

# Approximate Answers using Belief Networks

Berthier A. Ribeiro-Neto  
berthier@dcc.ufmg.br  
Computer Science Department  
Federal University of Minas Gerais  
Brazil  
CNPq grant 300188/95-1

Richard Muntz  
muntz@cs.ucla.edu  
Computer Science Department  
University of California, Los Angeles  
U.S.A.  
NASA grant NAG5-2225

## Abstract

We propose a *Bayesian belief network model* for dealing with approximations in the context of relational databases. Our model provides a general framework that can be successfully used to merge the notion of an approximation neighborhood with complex queries and a tuple ranking formula in consistent fashion. Through a mechanism of belief propagation, our model allows merging the influences of these different factors into a final ranking for tuples. We illustrate the main features of our model with examples.

## 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 [4].

An approximate 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 [1] 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 *conjunctions* and *disjunctions* would lead to complex probabilistic formulas with parameters that could not be accurately estimated in most cases. Rabitti and Savino [6] 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 [4] 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 Bayesian belief networks [5]. From fundamental assumptions we derive a formula for weighting tuples that is sensitive to the data distribution. Further, we also derive an expression for computing the similarity between a tuple and a user query. This expression adopts a normalization criterion based on the best neighbor which, in our belief, is intuitive from a user viewpoint. Further, such normalization allows the user to easily control the degree of approximation provided by the system.

The paper is organized as follows. Section 2 provides some background on Bayesian belief networks. Section 3 introduces a belief network model for a conventional database. This model does *not* include a ranking formula for tuples and allows only exact answers. In section 4 we extend the model with a ranking formula for tuples to incorporate information about the distribution of the data. In section 5 we extend the model with neighborhoods to allow approximate queries. In section 6 we highlight the main features of our model with examples. Our conclusions follow.

## 2 Bayesian Belief Networks

Bayesian belief networks are DAGs in which the nodes represent random variables, the arcs portray relationships between the linked variables, and the strengths of these influences are expressed by conditional probabilities. The *parents* of a node (which is then considered as a *child* node) are those judged to be direct *causes* for it. This causal relationship is represented in the DAG by a link directed from each parent node to the child node. The *roots* of the network are the nodes without parents.

Let  $x_i$  be a node in a Bayesian network  $G$  and  $\Gamma_{x_i}$  be the set of parent nodes of  $x_i$ . The influence of  $\Gamma_{x_i}$  on  $x_i$  can be specified by any set of functions  $F_i(x_i, \Gamma_{x_i})$  that satisfy

$$\sum_{\forall x_i} F_i(x_i, \Gamma_{x_i}) = 1 \quad (1)$$

$$0 \leq F_i(x_i, \Gamma_{x_i}) \leq 1 \quad (2)$$

This specification is complete and consistent because the product  $\prod_{\forall i} F_i(x_i, \Gamma_{x_i})$  constitutes a joint probability distribution for the nodes in  $G$ . The reader is referred to [5] for further details.

## 3 Belief Network Model for the Database

In this section we introduce a belief network model for a conventional relational database. We demonstrate the use of belief propagation in the network to generate exact answers. The model allows only exact queries and does not include a ranking formula for tuples. In sections 4 and 5, we extend the model to allow approximate answers.

**Definition 1** Let  $A_i$  be the  $i$ th attribute name in a given relation.  $D_i$  is the respective attribute domain. An attribute-value pair (avp)  $\vartheta_{i,j}$  is a reference to the  $j$ th distinct value in domain  $D_i$ .

**Definition 2** We model each distinct avp  $\vartheta_{i,j}$  as a binary random variable  $x_{i,j}$ . A value of  $x_{i,j}$  equal to 1 indicates that the avp is being used (by the system) to match a tuple and the query. In this case, we say that the avp is active. Otherwise, the value of  $x_{i,j}$  is 0.

**Definition 3** A query is modelled as a binary random variable  $q$ . The state of  $q$  is 1 if the set of active avps satisfies the query; otherwise, it is 0.

**Definition 4** We model each tuple by a binary random variable  $t$ . The state of  $t$  is 1 if the set of active avps matches the tuple; otherwise, it is 0.

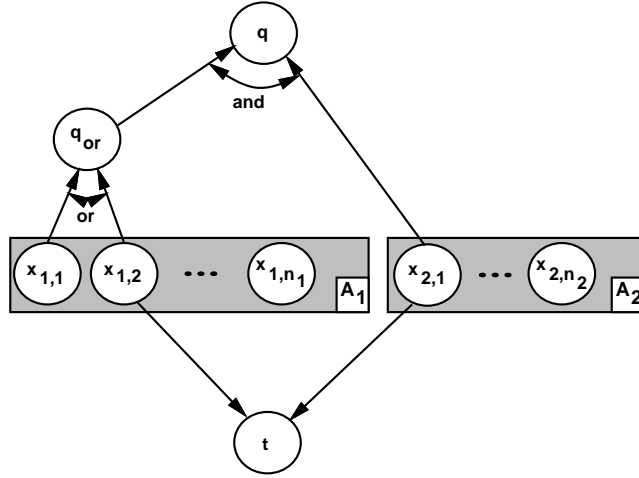


Figure 1: Bayesian belief network for the database.

Figure 1 illustrates our belief network for the database. Node  $q_{or}$  is an auxiliary variable modelling an *OR* condition. The query in this case is

$$q = (A_1 = \vartheta_{1,1} \vee A_1 = \vartheta_{1,2}) \wedge (A_2 = \vartheta_{2,1}) \quad (3)$$

The edges are directed towards  $q$  to indicate the dependency between the query and its avps as suggested by Pearl [5]. The node  $t$  models a tuple in the database that has value  $\vartheta_{1,2}$  for attribute  $A_1$  and value  $\vartheta_{2,1}$  for attribute  $A_2$ . The edges are directed towards  $t$  to indicate that its relevance (to the query) depends on the state of its components. Further, directing the edges towards the node  $t$  allows adding a tuple ranking formula to the model in natural fashion (as illustrated in section 4). The shaded boxes highlight attribute domains.

**Definition 5** Let  $\vec{x}$  denote the state  $(0,1)$  of all avp variables  $x_{i,j}$  (i.e., the root nodes).

The relevance of a tuple is determined by its *similarity* to the query. For a conventional database system, this similarity is 1 if the tuple satisfies the query; otherwise, it is 0. Our belief network model computes this similarity as the probability that there is a match for the tuple given that the query is satisfied, denoted  $P(+t|+q)$ . By conditioning on the state of the *root* nodes in the network (i.e.,  $\vec{x}$ ) we can write,

$$P(+t|+q) = \sum_{\forall \vec{x}} P(+t|+q, \vec{x}) \times P(\vec{x}|+q) \quad (4)$$

If the state of all parents of a node is known, that node is isolated from the rest of the network [5]. Thus, given  $\vec{x}$ ,  $t$  is independent of  $q$ . Further, we can convert  $P(\vec{x}|+q)$  into  $P(+q|\vec{x})$  by application of Bayes rule. Consequently,

$$P(+t|+q) = \sum_{\forall \vec{x}} P(+t|\vec{x}) \times \alpha \times P(+q|\vec{x}) \times P(\vec{x}) \quad (5)$$

where  $\alpha$  is simply a normalization constant corresponding to  $P(+q)$  [5].

Since we are interested only in exact answers, we define

$$P(+t|\vec{x}) = \begin{cases} 1 & \text{if, according to } \vec{x}, \text{ the parent avps of } t \text{ are active} \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

indicating that  $\vec{x}$  must match all avps in  $t$ .

For the query in figure 1 we have

$$P(+q|\vec{x}) = \begin{cases} 1 & \text{if, according to } \vec{x}, ((x_{1,1} \vee x_{1,2}) \wedge x_{2,1}) = 1 \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

indicating that  $\vec{x}$  must satisfy the query conditions.

Since a relational database is in *first normal form*, we restrict the state space for  $\vec{x}$ . Only states with exactly one active avp in each attribute domain are *valid* (i.e., receive a non-zero probability). For instance,  $[x_{1,2} = 1, x_{2,1} = 1, \text{ all others } 0]$  is a valid state for  $\vec{x}$  but  $[x_{1,1} = 1, x_{1,2} = 1, x_{2,1} = 1, \text{ all others } 0]$  is not (because two avps are active in domain  $D_1$ ). This guarantees that if there exists  $\vec{x}$  that satisfies the query and matches a tuple  $t$  then tuple  $t$  also satisfies the query.

**Definition 6** *The only valid states for  $\vec{x}$  are those with exactly one active avp in each attribute domain. Let  $n_i$  be the number of distinct avps in attribute domain  $D_i$ . Then,*

$$P(\vec{x}) = \prod_{\forall D_i} \frac{1}{n_i} \quad (8)$$

This implies that the system considers all avps in a domain equally likely to be active. The reason is as follows.  $P(\vec{x})$  are the *prior probabilities* for the root nodes in the network. They specify our knowledge of the application *before* any evidence (e.g., a query) is collected. Our system does *not* enforce prior preferences for any avp in a domain. Lack of prior preferences translates to uniformly distributed priors as discussed by Jaynes [2].

A tuple  $t$  satisfies a query  $q$  iff there exists  $\vec{x}$  such that  $P(+t|\vec{x}) = 1$  and  $P(+q|\vec{x}) = 1$ . In this case,  $P(+t|+q) > 0$ . Otherwise,  $P(+t|+q) = 0$ . Thus, tuples that do *not* match the query receive a ranking equal to 0. Any tuple that matches the query receives a constant ranking given by  $\alpha \times P(\vec{x})$ .

## 4 A Probability Formula for Tuples

In this section we modify the probability  $P(+t|\vec{x})$  to make the ranking of tuples sensitive to the data distribution. Our choice of a particular formula is motivated by basic principles and an analogy with ideas from information theory and information retrieval.

**Definition 7** *Let  $t[j]$  be the value of attribute  $j$  according to tuple  $t$ . Let  $\vec{x}[j]$  be the value of attribute  $j$  according to  $\vec{x}$ . Further, let  $\vec{x}_t$  be a vector such that  $\forall j, \vec{x}_t[j] = t[j]$ .*

**Assumption 1** *If  $\vec{x} \neq \vec{x}_t$  then  $P(+t|\vec{x}) = 0$ .*

This implies that *no* partial matches are allowed in the tuple side of our belief network. Thus, the only vector  $\vec{x}$  which matters is  $\vec{x} = \vec{x}_t$ . The idea is to keep the model simple. In section 5 we introduce *neighborhoods* into the query side of the network to allow approximate answers.

Research in information retrieval shows that ranking formulas based on the frequency count of terms inside a collection perform well [8, 9]. Models based on a probabilistic interpretation of these frequencies also perform well [1, 3]. This empirical evidence motivates our next assumption.

**Assumption 2**  $P(+t|\vec{x}_t)$  is a function  $f$  of the probability that  $\vec{x}_t$  occurs in the database. That is,

$$P(+t|\vec{x}_t) \propto f(P(\vec{x}_t)) \quad (9)$$

Equation 9 is too generic to be useful. To constrain the form of  $f$ , we search for additional intuition.

The best index term weighting schemas in information retrieval emphasize the selectivity of the index term [8]. Index terms that appear in many documents are not very discriminatory and therefore receive smaller weights. This idea concurs with the concept, introduced by Shannon [10], of *amount of information* generated by an event. Ross calls it *surprise* [7]. Our approach is to associate an *amount of information* to each avp in the database.

Let  $f$  be as in equation 9 and  $r$  be a variable for probabilities such as  $P(\vec{x}_t)$ .

**Axiom 1**

$$f(1) = 0 \quad (10)$$

The intuition is that an avp that occurs in every tuple (i.e., with probability 1) provides no information.

**Axiom 2**  $f(r)$  is a strictly decreasing function of  $r$ . Thus,

$$r < s \implies f(r) > f(s). \quad (11)$$

This axiom states that avps that are more discriminatory boost the value of  $f$ . This is a standard assumption in information retrieval systems.

**Axiom 3**  $f(r)$  is a continuous function of  $r$ .

This axiom states that small variations on  $r$  cause small changes on  $f(r)$  which is a desired property.

**Axiom 4**

$$f(r \times s) = f(r) + f(s) \quad (12)$$

This axiom states that avps occur *independently* in the database. Let  $r = P(\vec{x}[i])$  be the probability that avp  $\vec{x}[i]$  occurs in the database and  $s = P(\vec{x}[j])$  be the probability that avp  $\vec{x}[j]$  occurs in the database, where  $i \neq j$ . Since the domains  $D_i$  and  $D_j$  are considered mutually independent  $P(\vec{x}[i] \wedge \vec{x}[j]) = P(\vec{x}[i]) \times P(\vec{x}[j]) = r \times s$ . Application of function  $f$  yields  $f(P(\vec{x}[i] \wedge \vec{x}[j])) = f(r \times s)$ . Consider that we first observe avp  $\vec{x}[i]$  and afterwards we observe both  $\vec{x}[i]$  and  $\vec{x}[j]$ . From one observation to another, the value of function  $f$  changes by  $f(r \times s) - f(r)$ . Since the occurrence of  $\vec{x}[j]$  is assumed independent of the occurrence of  $\vec{x}[i]$ , it has to be that  $f(s) = f(r \times s) - f(r)$ .

**Theorem 1** If a function  $f$  satisfies axioms 1 through 4, then

$$f(r) = -C \log_2 r \quad (13)$$

where  $C$  is a positive integer.

**Proof** Refer to [7].

By substituting equation 13 into equation 9, we obtain

$$P(+t|\vec{x}_t) \propto -\log_2 P(\vec{x}_t) \quad (14)$$

Applying the independence assumption of axiom 4, we write

$$P(+t|\vec{x}_t) \propto -\log_2 \prod_{\forall i} P(\vec{x}_t[i]) \quad (15)$$

$$\propto \sum_{\forall i} \log_2 \frac{1}{P(\vec{x}_t[i])} \quad (16)$$

**Definition 8** Let  $n_{\vec{x}_t[i]}$  be the number of tuples in which avp  $\vec{x}_t[i]$  occurs and  $N$  be the total number of tuples in the respective relation (or in the resultant join relation in case of queries involving join operations). Then,

$$P(\vec{x}_t[i]) = \frac{n_{\vec{x}_t[i]}}{N} \quad (17)$$

The smallest value for this probability is  $\frac{1}{N}$ . Substituting 17 into 16 and normalizing properly (i.e.,  $0 \leq P(+t|\vec{x}_t) \leq 1$ ), we finally obtain

$$P(+t|\vec{x}_t) = \frac{\sum_{\forall i} \log_2 \frac{N}{n_{\vec{x}_t[i]}}}{\sum_{\forall i} \log_2 N} \quad (18)$$

$$P(-t|\vec{x}_t) = 1 - P(+t|\vec{x}_t) \quad (19)$$

This completes the specification of our probability formula for tuples. Avps that occur *less* frequently in the database are more selective and contribute with higher weight to the ranking.

## 5 Neighborhoods = Approximate Answers

In this section we introduce *neighborhood* nodes into the query side of our belief network. Neighborhood nodes allow partial matches (i.e., approximation) between  $\vec{x}$  and the query node  $q$ . If  $\vec{x}$  matches (exactly) a tuple  $t$ , this tuple receives a rank higher than zero. We say that tuple  $t$  *approximates* query  $q$ .

To be able to specify proximity or nearness, the user needs a *proximity* operator. Motro calls this operator *similar-to* [4]. We refer to it by either *similar-to* or  $\sim$ .

**Definition 9** A neighborhood  $\mathcal{N}_{i,v}$  is the set of all avps that can be used to approximate the avp  $\vartheta_{i,v}$ . Avp  $\vartheta_{i,v}$  provides an exact match for  $\mathcal{N}_{i,v}$ . The other avps provide approximate matches and are called *neighbor avps*.

**Definition 10** We model each neighborhood  $\mathcal{N}_{i,v}$  as a binary random variable  $\Upsilon_{i,v}$ . It is 1 to indicate a match (possibly partial) between an active avp and  $\mathcal{N}_{i,v}$ ; otherwise, it is 0. We adopt the simplified notation  $\mathcal{N}_v$  and  $\Upsilon_v$ , whenever the attribute domain  $D_i$  is implicitly clear. The respective avp and avp variable are referred to simply as  $\vartheta_v$  and  $x_v$ .

Measuring similarity requires the design of *data metrics*. A data metric is a function  $M : D \times D \rightarrow R$  that specifies a *semantic* distance between any two avps in the same attribute domain [4]. For instance,  $M(\text{drama}, \text{suspense}) = 2$  indicates that the film categories *drama* and *suspense* are

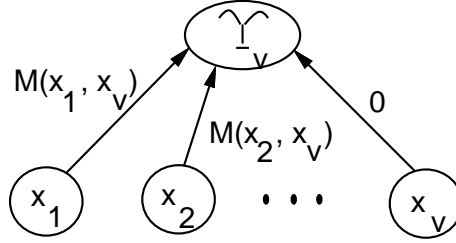


Figure 2: Belief network for a neighborhood  $\Upsilon_v$ .

separated by a semantic distance of 2. Avps in different attribute domains are considered unrelated (i.e., separated by an infinite distance). In this paper, we do not concern ourselves with the design of data metrics for a database (Motro [4] discusses data metrics in detail and suggests techniques for generating them automatically). We simply assume that they are provided.

Figure 2 illustrates the belief network corresponding to the neighborhood  $\Upsilon_v$ . The semantic *distance* between each avp (or avp variable) and the neighborhood node is given by the respective data metric value. For instance, the semantic distance from avp variable  $x_2$  to  $\Upsilon_v$  is  $M(x_2, x_v)$ . This distance is modelled in the belief network as a weight associated to the link connecting  $x_2$  to  $\Upsilon_v$ . The edges are oriented towards  $\Upsilon_v$  to indicate that the belief in the neighborhood depends on the state of its neighbor avps.

To complete our network we need to specify the probabilities  $P(+\Upsilon_v|\vec{x})$ . These probabilities quantify the *degree of similarity* between  $\vec{x}$  and  $\Upsilon_v$  and reflect general user preferences regarding  $\Upsilon_v$ . From basic assumptions, we derive an expression for  $P(+\Upsilon_v|\vec{x})$ . For simplicity, the notation  $+x_v$  is used to designate the states of  $\vec{x}$  for which  $\vartheta_v$  is the active avp in its domain.

**Assumption 3** *Any avp is completely similar to itself. In a scale from 0 to 1,  $P(+\Upsilon_v|+x_v) = 1$ .*

This ensures that avps that match the neighborhood exactly provide maximum contribution to the rank.

**Assumption 4** *The metric associated to an attribute domain reflects user preferences. If  $M(x_1, x_v) > M(x_2, x_v)$  then the user prefers  $x_2$  to  $x_1$  as an approximation for  $\Upsilon_v$ .*

This ensures that there is basic concordance between the metric and the user preferences. Given a metric, we assume that the user is willing to accept the *rulings* originated from that metric as suggested in [4]. In case the user desires a different set of approximate answers, he should submit a modified query or provide the system with feedback that can be used for tuning the answer set.

**Assumption 5** *In a neighborhood  $\Upsilon_v$ , the appealing avp for approximation is the closest neighbor to  $\Upsilon_v$ . We refer to the closest neighbor avp variable as  $\tau_v$ . The user evaluates preferences relative to  $\tau_v$ .*

This implies that  $M(\tau_v, x_v)$ , smallest distance for any neighbor avp, is used for normalization. We could have adopted a different normalization strategy (e.g., use the mean, the median, or the largest distance). We chose the smallest distance because, as exemplified in section 6, it facilitates an intuitive interpretation of the ranking by the user.

**Definition 11** *Let  $P(+\Upsilon_v|+\tau_v) = \kappa_v$ , where  $0 \leq \kappa_v < 1$ . We refer to  $\kappa_v$  as the similarity factor for  $\Upsilon_v$ .*

The similarity factor  $\kappa_v$  defines the similarity of the best neighbor  $\tau_v$ . It allows the user to control the *degree of approximation* provided by the best neighbor which affects the similarity of any neighbor in the domain (because of assumption 5). We are now ready to define the similarity between a generic avp and a neighborhood.

**Definition 12** Let  $x_w$  be a neighbor avp variable of  $\Upsilon_v$  ( $x_w \neq x_v$ ). Then,

$$P(+\Upsilon_v | +x_w) = \kappa_v \times \frac{M(\tau_v, x_v)}{M(x_w, x_v)} \quad (20)$$

The similarity of an avp variable  $x_w$  to a neighborhood  $\Upsilon_v$  is a fraction of the similarity factor  $\kappa_v$ . According to assumption 4, this fraction is inversely proportional to  $M(x_w, x_v)$ . According to assumption 5,  $M(\tau_v, x_v)$  is used as normalization factor.

We summarize this discussion in the following.

**Definition 13** Let  $\Upsilon_v$  be a neighborhood for  $x_v$ . The similarity between  $\Upsilon_v$  and  $x_v$  is 1. That is,

$$P(+\Upsilon_v | \dots, +x_v, \dots) = 1 \quad (21)$$

$$P(-\Upsilon_v | \dots, +x_v, \dots) = 0 \quad (22)$$

Let  $\tau_v$  be the closest neighbor of  $x_v$  according to a data metric for that domain. Let  $x_w$  be any neighbor of  $x_v$  ( $x_w \neq x_v$ ). Then,

$$P(+\Upsilon_v | \dots, +x_w, \dots) = \kappa_v \times \frac{M(\tau_v, x_v)}{M(x_w, x_v)} \quad (23)$$

$$P(-\Upsilon_v | \dots, +x_w, \dots) = 1 - P(+\Upsilon_v | \dots, +x_w, \dots) \quad (24)$$

where  $\kappa_v$ ,  $0 \leq \kappa_v \leq 1$ , is the similarity factor (i.e., similarity of the best neighbor).

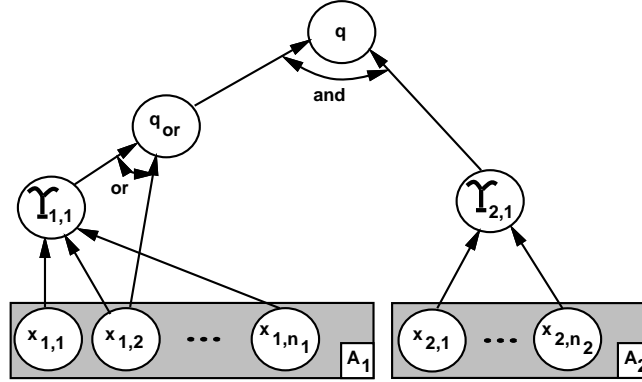


Figure 3: Belief network for  $q = (A_1 \sim \vartheta_{1,1} \vee A_1 = \vartheta_{1,2}) \wedge (A_2 \sim \vartheta_{2,1})$ .

The user specifies an *approximate* query by explicitly referring to a neighborhood. Consider the query example in section 3,  $q = (A_1 = \vartheta_{1,1} \vee A_1 = \vartheta_{1,2}) \wedge (A_2 = \vartheta_{2,1})$ . If the user is willing to accept approximations for  $\vartheta_{1,1}$  and  $\vartheta_{2,1}$ , he can rewrite this query as

$$q = (A_1 \sim \vartheta_{1,1} \vee A_1 = \vartheta_{1,2}) \wedge (A_2 \sim \vartheta_{2,1}) \quad (25)$$

Figure 3 illustrates the corresponding belief network. Given  $\vec{x}$ , the belief in the query is calculated as

$$P(+q | \vec{x}) = P(+q | q_{or}, \Upsilon_{2,1}) \times P(+\Upsilon_{2,1} | \vec{x}) \times P(+q_{or} | \vec{x}) \quad (26)$$

FILM						
Id	Title	Director	Category	Theater	Location	Rating
$t_1$	Four_Feathers	Korda	Adventure	Music_Hall	Beverly_Hills	3.5
$t_2$	Modern_Times	Chaplin	Comedy	Rialto	Downtown	4.0
$t_3$	Psycho	Hitchcock	Suspense	Music_Hall	Beverly_Hills	3.5
$t_4$	Rear_Window	Hitchcock	Suspense	Egyptian	Westwood	4.0
$t_5$	Robbery	Yates	Suspense	Odeon	Sta_Monica	3.0
$t_6$	Star_Wars	Lucas	Adventure	Chinese	Hollywood	3.5
$t_7$	Surf_Party	Dexter	Drama	Village	Westwood	0.0

Figure 4: A database for films.

where

$$P(+q|q_{or}, \Upsilon_{2,1}) = \begin{cases} 1 & \text{if } (q_{or} \wedge \Upsilon_{2,1}) = 1 \\ 0 & \text{otherwise} \end{cases} \quad (27)$$

$$P(+q_{or}|\vec{x}) = 1 - P(-\Upsilon_{1,1}|\vec{x}) \times P(-x_{1,2}|\vec{x}) \quad (28)$$

These equations reflect the dependencies implied by the network in figure 3. The probabilities  $P(+\Upsilon_{2,1}|\vec{x})$  and  $P(-\Upsilon_{1,1}|\vec{x})$  are given by equations 23 and 24. The expression for  $P(+q_{or}|\vec{x})$  excludes states that negate both  $\Upsilon_{1,1}$  and  $x_{1,2}$ .

**Lemma 1** *If  $\kappa_v = 0$  for every neighborhood  $\Upsilon_v$ , only exact answers are generated.*

**Proof** According to equation 23,  $\kappa_v = 0$  implies that the belief in the neighborhood  $\Upsilon_v$  is 0 for all neighbor avps. Thus, any state  $\vec{x}$  that *approximates* the query conditions will necessarily generate a belief in the query node  $q$  equal to 0. According to equation 5,  $P(+q|\vec{x}) = 0$  implies  $P(+t|+q) = 0$ . Thus, approximate answers are excluded from the rank.  $\square$

By setting the similarity factors  $\kappa_v$  to 0, the user can make our network model behave like a conventional database. Further, by adjusting them in the interval  $0 < \kappa_v \leq 1$ , he can control the similarity of neighbor avps (equation 20). The user has the flexibility to adjust the similarity factors for each neighborhood individually. However, this might require considerable effort. A useful alternative is to adopt a unique value of  $\kappa$  for all neighborhoods in a same attribute domain.

## 6 Examples

In this section, we use examples to highlight the main features of our belief network model. We consider a single value of  $\kappa$  for all neighborhoods in a same attribute domain.

### 6.1 The Sample Database

Figure 4 illustrates a database for films adapted from [4]. A film  $t_k$  has a title, a director, a category, a theater of exhibition, theater location, and a rating. For simplicity, our sample database contains a single relation. However, our network approach is also applicable to join queries involving multiple relations. One only has to consider that the tuples to be ranked are taken from the relation which results from the query processing (i.e., the resultant relation *after* the join has been computed).

Figure 5 illustrates the respective metrics for the attributes *category* and *location* (also from [4]). The metric for category reflects *semantic* distances. The metric for location uses physical distances

CATEGORY_METRIC		
Value_1	Value_2	Distance
Comedy	Drama	2
Comedy	Adventure	3
Comedy	Suspense	3
Drama	Adventure	2
Drama	Suspense	2
Adventure	Suspense	1

LOCATION_METRIC		
Value_1	Value_2	Distance
Beverly_Hills	Downtown	15
Beverly_Hills	Hollywood	8
Beverly_Hills	Santa_Monica	10
Beverly_Hills	Westwood	5
Downtown	Hollywood	10
Downtown	Santa_Monica	20
Downtown	Westwood	18
Hollywood	Santa_Monica	15
Hollywood	Westwood	10
Santa_Monica	Westwood	5

Figure 5: Metrics for *category* and *location*.

(in miles). Using these metrics we can define neighborhoods for each avp in the attribute domains *category* and *location*. Thus, the user can refer to these neighborhoods in his queries. The other attribute domains have no pre-defined metrics and do *not* allow approximations.

## 6.2 Example 1

Consider the query

```

Q :      select  title,theater,category
        from    film
        where   category ~ drama

```

which asks for films and the respective theaters of display such that the film category is *similar-to* drama. Tuple  $t_7$  provides the only exact match for  $\Upsilon_{drama}$ . Let  $\kappa_{cat}$  refer to the similarity factor for all neighborhoods in the attribute domain *category*.

Case 1:  $\kappa_{cat} = 0$

Ranking:  $t_7$

Case 2:  $0 < \kappa_{cat} < 1$

Ranking:  $t_7 > t_2 > (t_1 = t_6) > (t_3 = t_4 = t_5)$

In case 1,  $\kappa_{cat} = 0$  eliminates approximate answers and thus, only tuple 7 is relevant. In case 2, tuple 7 is the most relevant answer. The neighbor avps for  $\Upsilon_{drama}$  are *adventure*, *comedy*, and *suspense* and they all have the same semantic distance to *drama*. This implies that  $P(+q|\vec{x})$  is the same for all neighbor avps. Thus, the rank is determined by  $P(+t|\vec{x})$ . *comedy* is the most specific neighbor avp and thus,  $t_2$  receives a high rank. *suspense* is the least specific avp and thus, tuples  $t_3$ ,  $t_4$ , and  $t_5$  appear at the bottom of the rank.

This example illustrates two points: (a) our belief network can be naturally adapted to behave as a conventional database; and (b) if two avps have the same degree of similarity to the neighborhood, the most specific one contributes with higher weight to the rank. This second effect becomes important when the set of approximate answers is large. A large number of closely ranked approximate answers is usually overwhelming to the user. Our ranking strategy is able to distinguish the most specific avps pushing the respective tuples higher in the rank.

### 6.3 Example 2

Consider the query

```
Q :      select  title,theater,category
        from    film
        where   category ~ suspense
```

which asks for films and the respective theaters of display such that the film category is *similar-to* suspense. Tuples  $t_3$ ,  $t_4$ , and  $t_5$  provide exact matches for  $\Upsilon_{suspense}$ .

Case 1:  $\kappa_{cat} < 0.68$

Ranking:  $(t_3 = t_4 = t_5) > (t_1 = t_6) > t_7 > t_2$

Case 2:  $0.68 < \kappa_{cat} < 0.88$

Ranking:  $(t_1 = t_6) > (t_3 = t_4 = t_5) > t_7 > t_2$

Case 3:  $\kappa_{cat} > 0.88$

Ranking:  $(t_1 = t_6) > t_7 > (t_3 = t_4 = t_5) > t_2$

In case 1, the values for  $\kappa_{cat}$  restrict the contribution of neighbor avps to the rank. Thus, tuples with exact matches ( $t_3$ ,  $t_4$ , and  $t_5$ ) prevail even if they are less specific. *Adventure* is the closest neighbor of  $\Upsilon_{suspense}$  and thus, tuples  $t_1$  and  $t_6$  follow in the ranking. In case 2, the larger values for  $\kappa_{cat}$  make *adventure* more similar to  $\Upsilon_{suspense}$ . Also, *adventure* is more specific than *suspense*. The combination of this two effects pushes tuples  $t_1$  and  $t_6$  to the top of the rank even if they are *not* exact matches. In case 3, the values for  $\kappa_{cat}$  are larger than 0.88 which is sufficient to push tuple  $t_7$  up in the rank (past the exact matches  $t_3$ ,  $t_4$ , and  $t_5$ ), because its category (*drama*) is far more specific than *suspense*.

This example elucidates two important *rules of thumb* related to the ordering of tuples at the top of the rank.

Rule of Thumb 1 : By lowering the value of  $\kappa$  for a neighborhood, the user decreases the similarity of neighbor avps what isolates exact answers at the top of the rank (even if they are less specific).

Rule of Thumb 2 : By increasing the value of  $\kappa$  for a neighborhood, the user increases the similarity of neighbor avps what favors more specific avps (even if they are not exact matches).

These two basic rules allow the user to gain intuition about the ordering of tuples at the top of the rank even in the presence of complex queries (i.e., queries involving two or more vague conditions). We avoid discussing this issue here for lack of space.

## 7 Conclusions

We proposed a Bayesian belief network as a tool for ranking approximate answers in a relational database system. We first introduced a belief network model for a conventional database. We then extended this model with a probability formula for tuples to include information on the distribution of the data. Following, we introduced the concept of neighborhoods to allow approximate answers.

The probability formula for tuples makes our ranking sensitive to the data distribution in the system. This is particularly important for large sets of approximate answers. If many approximate answers are indistinguishable using semantic distance only, the user is overwhelmed with too many closely ranked answers. The probability formula avoids this inconvenience by distinguishing the answers which are most specific.

The approximation neighborhoods we introduced adopt a normalization strategy based on the best query neighbor. Such normalization strategy has two advantages. First, it allows the user to control the degree of approximation at the top of the rank in simple fashion. Second, it provides the user with an intuitive interpretation of the ordering of tuples at the top of the ranking (our two *rules of thumb*).

Through belief propagation, our network model combines the influences of the distribution of the data and of the degrees of approximation for the tuple attribute values (relative to the user query) into a final rank. Such ranking strategy was illustrated with examples.

## References

- [1] Norbert Fuhr. A probabilistic framework for vague queries and imprecise information in databases. In *Proceedings of 16th VLDB Conference*, Brisbane, Australia, 1990.
- [2] E.T. Jaynes. Prior probabilities. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4:227–241, 1968.
- [3] K.L. Kwock. Experiments with a component theory of probabilistic information retrieval based on single terms as document components. *ACM Transactions on Information Systems*, 8(4):363–386, October 1990.
- [4] Amihai Motro. Vague: A user interface to relational databases that permits vague queries. *ACM Transactions on Office Information Systems*, 6(3):187–214, July 1988.
- [5] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, Inc., 1988.
- [6] F. Rabitti and P. Savino. An information retrieval approach for image databases. In *Proceedings of 18th VLDB Conference*, Vancouver, British Columbia, Canada, 1992.
- [7] S. Ross. *A First Course In Probability*. Macmillan Publishing Company, 3rd edition, 1988. Chapter 9.
- [8] G. Salton and C. Buckley. Term-weighting approaches in automatic retrieval. *Information Processing & Management*, 24(5):513–523, 1988.
- [9] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill Book Co., New York, 1983.
- [10] C.E. Shannon. *The Mathematical Theory of Communication*. The University of Illinois Press, Urbana, U.S.A., 1949.