PIRE: An extensible IR engine based on probabilistic Datalog

Henrik Nottelmann

Institute of Informatics and Interactive Systems, University of Duisburg-Essen,
47048 Duisburg, Germany, nottelmann@uni-duisburg.de

Abstract. This paper introduces PIRE, a probabilistic IR engine. For both document indexing and retrieval, PIRE makes heavy use of probabilistic Datalog, a probabilistic extension of predicate Horn logics. Using such a logical framework together with probability theory allows for defining and using data types (e.g. text, names, numbers), different weighting schemes (e.g. normalised tf, tf.idf or BM25) and retrieval functions (e.g. uncertain inference, language models). Extending the system thus is reduced to adding new rules. Furthermore, this logical framework provide a powerful tool for including additional background knowledge into the retrieval process.

1 Introduction

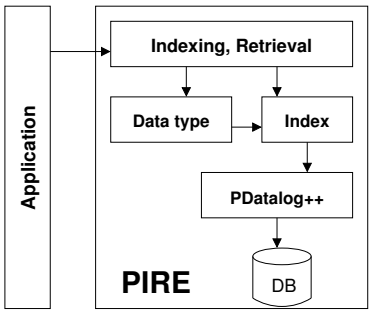
Information Retrieval has been investigated intensively within the last decades. Through all the years, researchers did not only aim at finding new techniques, one major focus always was on implementing the techniques and evaluating them with standard test collections, for example within the Text REtrieval Conference (TREC) or, more recently, the INEX evaluation initiative for XML retrieval.

Many IR techniques have been proposed. Besides heuristical approaches like the vector-space model, probabilistic IR can be justified theoretically following the Probability Ranking Principle (PRP) [14]. Thus, probabilistic IR approaches have become more and more popular, including the Binary Independence Retrieval (BIR) approach, the Darmstadt Indexing Approach (DIA), Language Models [13] or Uncertain Inference [18]. In combination with logics (e.g. probabilistic Datalog [6], a probabilistic extension of predicate Horn logics), probabilistic IR provides a powerful tool on a solid theoretical basis, which is extensible via additional rules, and which can take additional background knowledge into account.

The software presented in this paper, PIRE, is an extensible, general-purpose IR engine on the basis of probabilistic Datalog. Due to this logical foundation, PIRE has several advantages compared to common IR engines: The set of weighting schemes and retrieval functions is not fixed, but defined by logical rules. Using predicate logic increases the expressiveness, and enables to incorporate external knowledge, e.g. term associations from a thesaurus, into the retrieval process in a natural way.

PIRE uses the concept of data types. Each data type provides a number of operators which can be used for comparing document content with a query. As vagueness of query formulations is an important concepts of Information Retrieval (e.g. when a user

Fig. 1. PIRE architecture



is uncertain about the exact publication year of a document), these operators have a probabilistic interpretation (as proposed in [4]). Vagueness is required when a user is uncertain about the exact publication year of a document or the spelling of an author's name.

PIRE is implemented in Java and consists of five different components (see figure 1). The top component provides basic indexing and retrieval functions. It uses several indexes, each of them storing a specific part (called attributes) of the documents. In addition, operations that are specific for the data type like computing indexing weights are controlled by data type classes (e.g. for text, for names, for numbers). Probabilistic Datalog is employed for communication among the components, for defining the IR technique, and for storing the actual index (using a relational database system).

This paper is organised as follows. The next section summarises probabilistic Datalog and describes two major extensions which are required for Information Retrieval. Sections 3 and 4 explains how indexing and retrieval can be performed with probabilistic Datalog. Implementation details are presented in section 5, and further extensions are contained in section 6. The last section of this paper contains concluding remarks and an outlook into future work.

2 Probabilistic predicate logics

This section describes (probabilistic) Datalog and presents its extension, called pDatalog++.

2.1 Probabilistic Datalog

Datalog [16] is a variant of predicate logic based on function-free Horn clauses. An atom $p(t_1,\ldots,t_n) = p(\overline{t})$ is formed by a n-ary predicate p and terms $\overline{t}$ (constants or variables for constants). A literal is either an atom $p(\overline{t})$ (positive literal) or its negation $\neg p(\overline{t})$ (negative literal). A clause $\{\neg p_1(\overline{t}), \neg q(\overline{t}), r(\overline{t})\}$ is a set of literals with one positive literal. It can be seen as a disjunction $\neg p(\overline{t}) \wedge \neg q(\overline{t}) \wedge r(\overline{t})$ or the equivalent rule

$r(\bar{t}) \leftarrow p(\bar{t}), q(\bar{t})$. Here, the positive literal r denotes the head of the rule, the negative literals p and q form the rule body. Facts are rules with empty body.

In the remainder, we use a more technical notation for Datalog rules, which also PIRE uses internally. In particular, variables start with an upper-case letter, and constants with a lowercase letter. E.g., the fact that Jo is parent of Mary, that Jo is a man, and that fathers are male parents, can be expressed by:

```
parent(jo,mary).
male(jo).
father(X,Y) :- parent(X,Y) & male(X).
```

An interpretation contains all facts which are considered to be true. A model of a Datalog program is an interpretation which is consistent with the given facts and rules. As the Datalog semantics are defined by well-founded models [17], every Datalog program has at most one model.

In probabilistic Datalog [6], every fact or rule has attached a probabilistic weight α, prefixed to the fact or rule (weights $\alpha = 1$ can be omitted):

```
0.5 male(X) :- person(X).
0.8 person(ed).
```

The intended meaning of a rule αr is that "the probability that any instantiation of rule r is true is α". Thus, the preceding example pDatalog program expresses the fact a person is male with a probability of 50%, and that Ed is a person with probability of 0.8. Thus, $Pr(\texttt{male}(\texttt{ed})) = 0.8 \cdot 0.5 = 0.4$.

The probabilities, and thus the semantics, are formally defined as follows: The pDatalog program is modelled as a probability distribution over the set of all "possible worlds". A possible world is the well-founded model of a possible deterministic program, which is formed by the deterministic part of the program and a subset of the indeterministic part. As for deterministic Datalog, only modularly stratified programs are allowed [15]. The formal definition of modular stratification is rather complicated, basically it states that no ground fact is allowed to depend negatively on itself.

By default, facts are assumed to be independent; so the probability that two facts are true equals the product of the probabilities of the two facts. In addition, computing the probability of a disjunction requires to use the incluse-exclusion formula.

As a consequence, the possible worlds in the example are:

$$\begin{aligned} Pr(W_1) &= 0.2 \quad & W_1 &:= \{\}\ , \\ Pr(W_2) &= 0.4 \quad & W_2 &:= \{\texttt{person}(\texttt{ed})\}\ , \\ Pr(W_3) &= 0.4 \quad & W_3 &:= \{\texttt{person}(\texttt{ed}), \texttt{male}(\texttt{ed})\}\ . \end{aligned}$$

The probability of a fact is then computed by summing up the probabilities of all worlds in which the fact is true. Thus, we obtain $Pr(\texttt{male}(\texttt{ed})) = 0.4$, and $Pr(\texttt{person}(\texttt{ed})) = 0.4 + 0.4 = 0.8$ as stated as a fact.

Alternatively, sets of facts (e.g. all tuples in one relation) can be defined to be disjoint, which means that the probability of the conjunction equals zero. In this case, the probability of a disjunction equals the sum of the underlying probabilities. This feature will be heavily used throughout this paper.

2.2 Probabilistic Datalog++

We will show that probabilistic Datalog is not sufficient for PIRE. Thus, we introduce its extension pDatalog++, with the following differences:

- Constants are now numbers (integers or decimals) or strings, and are optionally enclosed in `"..."` or in `'...'` (for constants which contain e.g. whitespace).
- Variables that are never used in another argument of the same rule can be replaced by _ (the underscore). This is syntactic sugar which prevents to introduce variables which are only used once.
- SQL-like aggregation operators are introduced.
- The independence assumption can be replaced by arbitrary functions for computing the probabilities for a rule.

Aggregation operators Aggregation operators like `sum` have proven to be useful in SQL, and are thus introduced into pDatalog++. An aggregation function $f : 2^D \mapsto D$ takes a bag of values in the domain D as input, and returns a single value in D. Currently, the functions `sum`, `count`, `avg`, `min` and `max` (with the typical meaning) are supported. Aggregation functions can be embedded in pDatalog++ by aggregation operators, which have the prototypical form:

$$op(A, Y_1, \ldots, Y_y, \{p(X_1, \ldots, X_x)\}) .$$

Here, op denotes the aggregation function and the variable A the aggregation result. Furthermore, the Y_i are variables, p is a predicate, and each variable X_j either equals one of the Y_i, the placeholder _, or `#`, which denotes the argument containing the values from the domain D which have to be aggregated (thus, the `#` appears exactly once). Such a literal defines the following (nameless) relation:

$$\{(a, \bar{y}) | \exists \bar{t} \colon S = \{v | (\bar{y}, \bar{t}, v) \in \mathtt{p}\}, S \neq \emptyset, a = \mathtt{op}(S).\}$$

Here, $\bar{y} = (y_1, \ldots, y_y)$ is bound to the variables Y_i; one aggregation value $a = a(\bar{y})$ is computed for each of these tuples $\bar{y}$. In addition, $\bar{t}$ stands for the occurrences of the underscore (the free variables), and v denotes the values which are aggregated. Obviously, this approach ignores any probabilities, and just considers any tuple with a non-zero probability.

This is equivalent to `group by` in SQL, here:

`select` $A, Y_1, \ldots, Y_y$ `from` p `group by` $Y_1, \ldots, Y_y$

The following real-world examples compute the document length as the sum of the corresponding term frequencies, count all documents containing the term, and compute the average document length:

```
dl(D,DL) :- sum(DL,D,{tf(D,_,#)}).
df(T,DF) :- count(DF,T,{tf(#,T,_)}).
rd('avgdl',AVGDL) :- avg(AVGDL,{dl(_,#)}).
```

Computing probabilities Probabilistic Datalog is based on an independence assumption. In some cases, for example for most retrieval methods, arbitrary functions for computing the probability of facts derived by a single rule are required. These functions can use the probabilities of facts bound by literals, their product, and any variable occurring in the rule.

A probabilistic version of normalised tf.idf can be computed by these rules:

```
tmp_tf(D,T) :- tf(D,T,TF) & dl(D,DL) | TF/DL.
tmp_idf(T) :- df(T,DF) & numdocs(N) | log(N/DF)/log(N).
weight(D,T) :- tmp_tf(D,T) & tmp_idf(T) | PROB1*PROB2.
```

The part after the pipe symbol `|` denotes the function used for computing the probabilities of derived facts. The first rule computes the TF part, where the variables `TF` and `DL` (bound by the two subgoals) are used for computing the TF-based probability. Similarly, the second rule computes the IDF part.

The last rule combines the two probabilities; where `PROB1` refers to the probability of the first literal, and `PROB2` refers to the probability of a ground atom bound by the second literal. For simplicity, `PROB1*PROB2` is equivalent to `PROB` (always the product, following an independence assumption). As this is the default, it can be omitted.

3 Indexing documents with pDatalog++

This section describes the indexing part of PIRE. First, the document model is introduced (which is very similar to the one proposed in [5]). Then, indexing weights are defined for several operators. Finally, pDatalog++ relations and rules for computing indexing weights are presented.

3.1 Data types and operators

We first assume a finite set $\mathbf{D}$ of elementary data types, where each data type $d \in \mathbf{D}$ has a domain (set of values) $dom(d)$. Furthermore, we consider a set $\mathbf{O}$ of operators[1]. Given an interpretation I, an operator $o \in \mathbf{O}$ defines a binary relation (instantiation) $o^I \subseteq dom(d_1(o)) \times dom(d_2(o))$ with respect to two data types $d_1(o), d_2(o) \in \mathbf{D}$. In the remainder, $\mathbf{D}$ contains the data type `DOCID` which denotes the set of all document ids. There is no operator defined for `DOCID`, i.e. $\forall o \in \mathbf{O} : d_1(o), d_2(o) \neq$ `DOCID`.

Vagueness of query formulations is supported by probabilistic interpretations of operators, i.e. each fact $o(v_1, v_2)$ has attached a probabilistic weight which describes the probability that v_1 is a match for v_2. Sometimes, we use an operator as a function and refer to this probability as $o(v_1, v_2) \in [0, 1]$. Similar to [10], PIRE supports these data types:

"Text": For `Text`, one textual value (called d here, as it refers to the document content) is compared with a single term t. The operator `stemen` uses stemming (Porter stemmer for the English language) and stop-word removal, while `nostem` does not

[1] Also called "data type predicates" in the Digital Library field. We use the term "operator" here to avoid confusion with predicates in logics.

apply stemming. Both share the same modified BM25 weighting scheme (using the term frequency $tf(d,t)$, the document length $dl(d)$ and its average value *avgdl*, the number of documents N and the document frequency $df(t)$):

$$\texttt{stemen}(d,t) := \frac{tf(d,t)}{tf(d,t)+0.5+1.5\cdot\frac{dl(d)}{avgdl}} \cdot \frac{\log\frac{N+0.5}{df(t)}}{\log N+0.5}. \quad (1)$$

"Name": This data type for person names supports two Boolean operators `plainname` and `soundex`, i.e.

$$\texttt{plainname}(v_1,v_2), \texttt{soundex}(v_1,v_2) \in \{0,1\}.$$

As an example, we have $\texttt{soundex}(''\texttt{Jones}'', ''\texttt{Johnson}'')) = 1$.

"Number", "Year": The data type `Number` has the Boolean operators `=`, `<`, `>`, `<=` and `>=`, i.e. the indexing weight is in $\{0,1\}$. The data type `Year` is equivalent to `Number`, but intended for years and not for numbers.

When a user is uncertain about the exact publication year of a document and requests documents from the year 1999, a document from the year 2000 might also be relevant (although the probability is lower). Thus, vague operators $\sim$=, $\sim$< and $\sim$> are introduced:

$$\sim >(v_1,v_2) := \begin{cases} 1-\frac{v_2-v_1}{v_1} & , \quad v_1 < v_2 \\ 1 & , \quad else \end{cases},$$

$$\sim =(v_1,v_2) := 1-\left(\frac{v_1-v_2}{v_2}\right)^2.$$

3.2 Schemas and indexing weights

Each document in PIRE adheres to a schema, which defines a list of (potentially multi-valued) attributes A_i. This document model is mapped onto our logical framework: Each attribute A_i is modelled as a binary relation symbol with a data type $d_{A_i} \in \mathbf{D}$. The relations A_i can be uncertain, too, for modelling uncertain knowledge. For an interpretation I, each relation symbol A_i is mapped onto a relation instance $A_i^I \subseteq \texttt{DOCID} \times dom(d_{A_i})$ with the correct data types. The value (second argument) of a relation A_i for a specific document d is denoted by $A_i^I(d)$, abbreviated by $A_i(d)$.

Closely related to attributes and operators are indexing weights. They are the result of applying an operator $o \in \mathbf{O}$ to the content of a document attribute $A_i(d) \in dom(d_1(o))$ and a second value $v \in dom(d_2(o))$. The notion of "indexing weight" typically appears in the area of text retrieval, where a document/term pair in the index has assigned a weight (derived from the index). This weight is typically stored in the index for performance reasons. We generalise this idea and call the result of an operator also an indexing weight (even in cases where it is not stored explicitly, e.g. for a data type "Year" and a less-than operator).

3.3 PDatalog++ rules for indexing

This section describes how PIRE indexes documents via pDatalog++ relations and rules. For each attribute and each their possible operators, a separate "index" is created.

An index is a set of pDatalog++ relations which belong to the same attribute/operator pair. Indexing is performed locally in an index; retrieval requires the combination of several indexes.

Every index has a ternary relation `tf` (this name, like all others, is local to the index) which stores the token frequencies $tf(d,t)$, where the definition of a "token" depends on the operator (e.g. terms for `Text`, a first name or a last name for `Name`, and the complete number for `Year`). This relation is created first, by splitting the document content (using program code which depends on the data type). The deterministic unary relation `docid` contains all document ids. The final indexing weights are stored in a binary relation `weight`, where the indexing weight is the probability of the corresponding fact. Furthermore, each index has a binary relation `rd` for the "resource description". A resource description contains parameters which can be used in all stages of the IR process. Numerical data, e.g. parameters for the mapping functions, which are identified via textual keys. Some data types and operators use additional temporary relations for computing the indexing weights.

The following example shows how BM25 indexing weights (see equation 1) are computed for the operator `stemen`. In a preprocessing step, document frequency, document length, average document length and the number of documents are determined:

```
df(T,DF) :- count(DF,T,{tf(#,T,_)}).
dl(D,DL) :- sum(DL,D,{tf(D,_,#)}).
rd('avgdl',AVGDL) :- avg(AVGDL,{dl(_,#)}).
rd('numdocs',NUMDOCS) :- count(NUMDOCS,{tf(#,_,_)}).
```

Then, BM25 indexing weights can be computed. For simplicity, the computation is split into two parts (TF and IDF):

```
tmp_tf(D,T) :- tf(D,T,TF) & dl(D,DL) &_rd('avgdl',A) | TF/(TF+0.5+1.5*DL/A).
tmp_idf(T) :- df(T,DF) & rd('numdocs',N) | log((N+0.5)/DF)/log(N+0.5).
weight(D,T) :- tmp_tf(D,T) & tmp_idf(T)  | PROB1*PROB2.
```

4 Retrieval with PDatalog++

This section describes the retrieval part of PIRE. First, query syntax and semantics as well as mapping functions are defined. Then, pDatalog++ rules for actually performing retrieval are presented.

4.1 Queries

An abstract syntax is used for expressing queries, which will later be translated into sets of pDatalog++ rules. Each query refers to one schema $\mathbf{R}$, and returns document ids.

A query condition c consists of a schema attribute $A(c) \in \mathbf{R}$, an operator $o(c)$ and a comparison value $v(c)$ with matching data types, i.e. $d_{A(c)} = d_1(o(c))$ and $v(c) \in d_2(o(c))$. An example is:

`author soundex "nottelmann".`

Queries are formed by conditions. Two different types of queries are supported: weighted sums and Boolean-style queries.

Weighted sums have the form $\texttt{wsum}(w(c_1)\ c_1, \ldots, w(c_n)\ c_n)$, where the c_i are conditions, and the $w(c_i) \in [0,1]$ are probabilistic weights representing the importance of the conditions. The idea is that the comparison weight for a document w. r. t. the query is the sum of the comparison weights w. r. t. the conditions c_i, weighted by the $w(c_i)$; the precise semantics are described in section 4.2.

Boolean-style queries use the Boolean operators `and` (conjunction) and `or` (disjunction) as connectors of conditions. This example query will return documents published in 2003 by a person whose name sounds like "Doe":

$$(\texttt{year} >= 2003)\ \texttt{and}\ (\texttt{author soundex}\ ''\texttt{doe}'').$$

By abuse of notation, $c \in q$ denotes that condition c occurs in query q.

4.2 Uncertain Inference and Probabilities of relevance

Probabilistic Datalog adopts Rijsbergen's view of information retrieval as uncertain inference, a variant of the logical view on databases, where queries and document contents are treated as logical formulae, and a database only returns those documents d which logically imply the query q, i.e. it proves $q \leftarrow d$. For considering the intrinsic uncertainty of information retrieval, Rijsbergen interprets probabilistic IR as estimating the probability $Pr(q \leftarrow d) = Pr(q|d)$ that the document logically implies the query, and used this probability as the retrieval status value (RSV).

First, the probability that a document implies a single condition is considered:

$$Pr(c_i \leftarrow d) := o(c_i)(A(c_i)(d), v(c_i)).$$

Weighted sum queries consist of a set of conditions c_i with associated weights $Pr(q \leftarrow c_i) = w(c_i) \in [0,1]$. The underlying facts are defined to be disjoint, and the widely used linear retrieval function [20] is employed for computing the probability of inference:

$$Pr(q \leftarrow d) = \sum_{c \in q} Pr(q \leftarrow c) \cdot Pr(c \leftarrow d).$$

For Boolean-style queries, the inclusion-exclusion formula has to be applied for disjunctions, and the probabilities have to be multiplied for conjunctions (independence assumption):

$$Pr((c_1 \wedge c_2) \leftarrow d) = Pr(c_1 \leftarrow d) \cdot Pr(c_2 \leftarrow d).$$

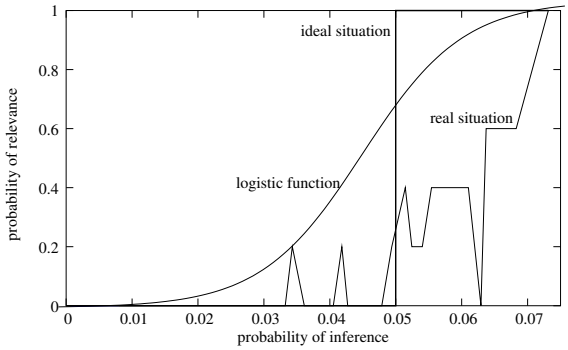
For advanced applications like combining different operators, the RSVs have to be transformed (using a mapping function) into the probability $Pr(\text{rel}|d,q)$ that document d is relevant to a user query q ("probability of relevance") [12]:

$$f : I\!R \mapsto [0,1], \quad f(Pr(q \leftarrow d)) \approx Pr(\text{rel}|q,d).$$

Each operator has its own mapping function. Thus, we split the query q into sub-queries $q_{A,o}$ which only refer to one attribute A and operator o combination.[2] The retrieval status

[2] When a Boolean-style query is split, several sub-queries can refer to the same attribute/operator pair, depending on the overall query structure

Fig. 2. Ideal and real case



values $Pr(q_{A,o} \leftarrow d)$ for this sub-query are then converted into probabilities of relevance using an attribute- and operator-specific mapping function $f_{A,o}$. The overall probability of relevance is derived by combining the probabilities of relevance for the sub-queries according to the original query structure.

In this paper, we consider linear and logistic mapping functions:

$$
\begin{aligned}
f_{lin}(x) &:= c_1 \cdot x, \\
f_{alin}(x) &:= c_0 + c_1 \cdot x, \\
f_{log}(x) &:= \frac{\exp(b_0 + b_1 \cdot x)}{1 + \exp(b_0 + b_1 \cdot x)}, \\
f_{max}(x) &:= \frac{x}{\max x'}.
\end{aligned}
$$

Both linear functions have the same drawback. They do not ensure that the results are between 0 and 1 in the general case of $c_0, c_1 \in \mathbb{R}$. In other words, the result cannot necessarily be regarded as a probability. However, linear mapping functions can be justified in the context of uncertain inference [19].

A better alternative is the logistic function [2, 3], which has been used in different application areas within IR for quite some time [7, 1, 9]. It can be seen as a continuous approximation of the step function in the ideal situation, where exactly the documents in the ranks $1, \ldots, l$ are relevant, and the documents in the remaining ranks $l+1, \ldots$ are irrelevant (see figure 2).

In PIRE, the operators of the data type `Text` use a combination of f_{max} (applied first, so that the top-ranked document has a weight of one) and the logistic mapping function (with default parameters $b_0 = -4$ and $b_1 = 12$, other parameters can be set as well in the resource description). The same mapping functions are defined for the operators $\sim<$, $\sim>$ and $\sim=$ for the data type `Year`. The operators of the other data types return values in $\{0, 1\}$, and thus employ the identity mapping function $f_{id}(x) = x$.

4.3 PDatalog++ rules for retrieval

For each sub-query q_i (with the sub-query id $i \geq 0$), a temporary predicate `rsv[query id]_[i]` stores the RSV $Pr(q_i \leftarrow d)$. The corresponding temporary predicate `prob[query id]_[i]` is used for computing $Pr(\text{rel}|q_i,d)$ from $Pr(q_i \leftarrow d)$. In a final step, the probabilities from the relations `prob[query id]_[i]` are combined in a relation `prob[query id]`.

For weighted sum queries, each sub-query contains all conditions for one combination of an attribute and an operator. The condition weights in the sub-query are normalised so that their sum equals one. Thus, the query

```
q := wsum(0.1 ti stemen ''hello'',0.3 ti stemen ''world'',
          0.6 ab stemen ''java'')
```

with the attributes `ti` and `ab` is split into two sub-queries

```
q1 := wsum(0.25 ti stemen ''hello'',0.75 ti stemen ''world'')
q2 := wsum(1,ab stemen ''java'')
```

The probabilities of relevance $Pr(\text{rel}|q_i,d)$ are computed by these rules. For simplicity, we use f_{max} as a mapping function, where 0.25 and 0.33 are arbitrarily chosen maximum RSV just for illustrating the retrieval process:

```
rsv42_0(D) :- weight(D,'hello') | (0.1/0.4)*PROB.
rsv42_0(D) :- weight(D,'world') | (0.3/0.4)*PROB.
prob42_0(D) :- rsv42_0(D) | PROB/0.25.

rsv42_1(D) :- weight(D,'java') | (0.6/0.6)*PROB.
prob42_1(D) :- rsv42_1(D) | PROB/0.33.
```

The resulting probabilities of relevance for the sub-queries are then combined by a weighted sum:

$$Pr(\text{rel}|q,d) = \sum_i \left(\sum_{c \in q_i} Pr(q_i \leftarrow c) \right) \cdot Pr(\text{rel}|q_i,d).$$

This can easily be converted into two pDatalog++ rules, assuming disjointness of the facts in `prob42_0` and those in `prob42_1`:

```
prob42(D) :- prob42_0(D) | 0.4*PROB.
prob42(D) :- prob42_1(D) | 0.6*PROB.
```

Boolean-style queries are transformed into disjunctive form (a disjunction of conjunctions of conditions). Each single condition forms a sub-query, and their results are combined straight-forward. Thus, the query

```
q := (ti stemen ``hello'' and ti stemen ``world'') or ab stemen ``java''
```

is transformed into these rules:

```
rsv42_0(D) :- weight(D,'hello').
prob42_0(D) :- rsv42_0(D) | PROB/0.2.

rsv42_1(D) :- weight(D,'world').
```

```
prob42_1(D) :- rsv42_1(D) | PROB/0.4.

rsv42_2(D) :- weight(D,'java').
prob42_2(D) :- rsv42_2(D) | PROB/0.1.

prob42(D) :- prob42_0(D) & prob42_1(D).
prob42(D) :- prob42_2(D).
```

Here, no disjointness is assumed for the facts of the relations `prob42_0`, `prob42_1` and `prob42_2`.

5 Implementation

PIRE itself is fully implemented in Java, but in its current state heavily uses a relational database. PIRE is available as Open Source.[3]

A specialised index class is used for managing the IR indexes (one for every attribute/operator combination, as mentioned before). In addition, each data type has its own class so that operations which depend on the data type can easily be separated from general code. A third, thin component (named "Indexing, Retrieval" in figure 1) glues together index and data type classes and can be called from outside with high-level methods. E.g., indexing documents (for simplicity, only with one attribute and operator) requires these few lines of pseudo code (each line corresponds to exactly one Java call):

```
create new PIRE instance
for all attributes:
  register attribute with operators
init all indexes
for all document:
  add document id to index
  for all attributes in document:
    add attribute content to index
compute indexing weights
```

PDatalog++ rules as described in this paper are used for communication between these classes, e.g. for the rules for computing indexing weights.

The architecture is flexible so that this index class can easily be exchanged by a new implementation (see below for a concrete implementation). The current implementation passes facts and rules to a pDatalog++ layer. This component stores facts (e.g. the `weight` facts) in relational tables. The arguments of a pDatalog++ predicate correspond to the columns `arg0`,...,`argn` in the table; an additional column `prob` stores the probability of the tuples. Rules are converted into SQL statements. As an example, the rule

```
rsv42_0(D) :- weight(D,'hello') | (0.1/0.4)*PROB.
```

will be transformed into

```
insert into rsv42_0 select arg0,0.1/0.4*prob from weight where arg1='hello'
```

[3] http://www.is.informatik.uni-duisburg.de/projects/pire/

The details of this mapping procedure are beyond the scope of this paper. Different relational database management systems can be used via JDBC. In particular, HSQLDB[4] provides in-memory tables, so that no I/O effort is required. This is particularly useful where indexing and retrieval has be done on the fly, e.g. for ranking the final results in distributed IR.

For efficiency reasons, PIRE currently uses extensional semantics [6], which means that probabilities are derived directly from the probabilities of the underlying facts. In some cases (not with the rules used so far), the probability of the same fact is considered twice. This problem can be solved by switching to intensional semantics which keeps tracks of all underlying facts. The drawback of intensional semantics is their exponential time complexity.

6 Possible extensions

As PIRE is based on logical rules, it can easily be extended towards other retrieval models (e.g. BIR or language models). It is also easy to incorporate external knowledge like a thesaurus, or to include hyper links in the retrieval process.

6.1 Language models

So far, we only investigated weight sums and Boolean-style queries. The flexibility of pDatalog++ allows for further retrieval models, e.g. language models. Here, the probabilities $Pr(t|d) = Pr(t \leftarrow d)$ (the normalised frequency of term t in document d) and $Pr(t|G)$ (normalised frequency of t in the background knowledge, e.g. the complete collections) are combined with parameter λ; queries are sets of words:

$$Pr(q|d) = \prod_{t \in q} (\lambda Pr(t|d) + (1-\lambda) Pr(t|C)) \ .$$

This can easily be modelled in pDatalog++. First the probabilities $Pr(t|d)$ (stored in the relation `weight`) and the collection-specific probabilities $Pr(t|C)$ (stored in the relation `cweight`) have to be computed (the latter could also be specified manually through facts):

```
df(T,DF) :- count(DF,T,{tf(#,T,_)}).
dl(D,DL) :- sum(DL,D,{tf(D,_,#)}).
numdocs(N) :- count(N,{docid(#)}).
tfc(T,TF) :- sum(TF,T,{tf(_,T,#)}).
cl(DL) :- sum(DL,{dl(_,#)}).
weight(D,T) :- tf(D,T,TF) & dl(D,DL) | TF/DL.
cweight(T) :- tfc(T,TFC) & cl(DLC) | TFC/DLC.
```

Then, for each query term, retrieval status values have to be computed, and combined in the final result (if required, also probabilities of relevance could be computed):

```
rsv42_i(D) :- weight(D,'hello') | 0.3*PROB.
rsv42_i(D) :- cweight('hello') | 0.7*PROB.
...
rsv42(D) :- rsv42_0(D) & rsv42_1(D) & ... & rsv42_n(D).
```

[4] `http://hsqldb.sourceforge.net`

6.2 External knowledge

One of the major advantages of logical frameworks in IR is that external knowledge can easily be incorporated into the retrieval process. For example, term associations from a thesaurus like WordNet[5] can be used for query expansion.

WordNet already contains a set of facts which can be directly be used in pDatalog. E.g. "synsets" group synonyms together:

```
s(100012748,1,'animal',n,1,67).
s(100012748,2,'animate_being',n,1,0).
s(100012748,3,'beast',n,1,4).
s(100012748,4,'brute',n,2,0).
s(100012748,5,'creature',n,1,16).
s(100012748,6,'fauna',n,2,0).
```

The first argument is the synset id, the third one the term. Thus, among other terms, "animal", "beast" and "fauna" are seen as synonyms. These synsets can be exploited for retrieval, by searching for documents which contain a synonym of a search term:

```
rsv42_0(D) :- weight(D,'hello').
rsv42_0(D) :- weight(D,T) & s(N,_,T,_,_,_) & s(N,_,'hello',_,_,_).
```

Of course, more complex situations can be incorporated as well.

7 Conclusion and outlook

This paper introduced PIRE, a probabilistic IR engine. PIRE is based on probabilistic Datalog, provides a theoretically founded basis for information retrieval. PIRE is simple to use in applications, and is already integrated in the federated Digital Library system Daffodil[6] for retrieval on locally available collections as well as for ranking the merged result list. For indexing and retrieval of documents, probabilistic Datalog has been extended by aggregation operators and by arbitrary functions for computing probabilities. This extension is called pDatalog++.

PIRE can be easily extended towards new application areas. Besides some glue code and a small amount of document preprocessing like splitting a document value into tokens (for which a variety of existing classes can be used), everything is captured in rules. This makes it easy to integrate new data types, operators, weighting schemes and retrieval functions.

We did not mention an additional feature of PIRE, namely computing moments (expectation and variance) of indexing weights, that allows for using the PIRE infrastructure also for the decision-theoretic framework of resource selection [11].

With logics, it is straight-forward to integrate additional background knowledge into account, e.g. a thesaurus, or to include hyper links between documents (or parts of them), annotations or contextual information into the retrieval process. In the near future, we will add this feature to PIRE.

[5] http://www.cogsci.princeton.edu/~wn/

[6] http://www.daffodil.de

We also plan to tune the pDatalog++ processing component to enhance speed, by improving the transformation of pDatalog++ rules into SQL statements.

PIRE can also be extended towards other directions. E.g., we did not use negation throughout this paper, but the limited usage of negation in pDatalog can quite easily be integrated into the system. Documents could also be described by logical expressions, allowing for a richer document model. Partial representations of documents are possible when switching to four-valued probabilistic Datalog [8] (with additional probabilities for the truth values "unknown" and "inconsistent").

Finally, XML retrieval should be supported. A primitive approach which is already implemented is to map sub-trees of the XML document (defined e.g. by XPath expressions) onto attributes. We plan to turn PIRE into a fully-fledged XML retrieval engine which explicitly takes the hierarchical structure of the documents into account. Logics seem to be an excellent starting point for this.

8 Acknowledgements

This work is supported in part by the DFG (grant BIB47 DOuv 02-01, PEPPER).

References

[1] W. S. Cooper, F. C. Gey, and D. P. Dabney. Probabilistic retrieval based on staged logistic regression. In N. J. Belkin, P. Ingwersen, and A. M. Pejtersen, editors, *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Copenhagen, Denmark, June 21-24, 1992*, pages 198–210, New York, 1992. ACM.

[2] S. Fienberg. *The Analysis of Cross-Classified Categorical Data*. MIT Press, Cambridge, Mass., 2. edition, 1980.

[3] D. H. Freeman. *Applied Categorical Data Analysis*. Dekker, New York, 1987.

[4] N. Fuhr. A probabilistic framework for vague queries and imprecise information in databases. In *Proceedings of the 16th International Conference on Very Large Databases*, pages 696–707, Los Altos, California, 1990. Morgan Kaufman.

[5] N. Fuhr. Towards data abstraction in networked information retrieval systems. *Information Processing and Management*, 35(2):101–119, 1999.

[6] N. Fuhr. Probabilistic Datalog: Implementing logical information retrieval for advanced applications. *Journal of the American Society for Information Science*, 51(2):95–110, 2000.

[7] N. Fuhr and U. Pfeifer. Combining model-oriented and description-oriented approaches for probabilistic indexing. In *Proceedings of the Fourteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 46–56, New York, 1991. ACM.

[8] N. Fuhr and T. Rölleke. HySpirit – a probabilistic inference engine for hypermedia retrieval in large databases. In *Proceedings of the 6th International Conference on Extending Database Technology (EDBT)*, pages 24–38, Heidelberg et al., 1998. Springer.

[9] F. C. Gey. Inferring probability of relevance using the method of logistic regression. In B. W. Croft and C. J. van Rijsbergen, editors, *Proceedings of the Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 222–231, London, et al., 1994. Springer-Verlag.

[10] H. Nottelmann and N. Fuhr. Decision-theoretic resource selection for different data types in MIND. In J. Callan, F. Crestani, and M. Sanderson, editors, *Recent research in multimedia distributed information retrieval. Proceedings of the ACM SIGIR 2003 Workshop on Distributed Information Retrieval, Toronto, Canada. (Lecture Notes in Computer Science, 2924)*., Heidelberg et al., 2003. Springer.

[11] H. Nottelmann and N. Fuhr. Evaluating different methods of estimating retrieval quality for resource selection. In J. Callan, G. Cormack, C. Clarke, D. Hawking, and A. Smeaton, editors, *Proceedings of the 26st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New York, 2003. ACM.

[12] H. Nottelmann and N. Fuhr. From retrieval status values to probabilities of relevance for advanced IR applications. *Information Retrieval*, 6(4), 2003.

[13] J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. In W. B. Croft, A. Moffat, C. J. van Rijsbergen, R. Wilkinson, and J. Zobel, editors, *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 275–281, New York, 1998. ACM.

[14] S. E. Robertson. The probability ranking principle in IR. *Journal of Documentation*, 33:294–304, 1977.

[15] K. Ross. Modular stratification and magic sets for Datalog programs with negation. *Journal of the ACM*, 41(6):1216–1266, Nov. 1994.

[16] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume I. Computer Science Press, Rockville (Md.), 1988.

[17] A. van Gelder, K. Ross, and J. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, July 1991.

[18] C. J. van Rijsbergen. A non-classical logic for information retrieval. *The Computer Journal*, 29(6):481–485, 1986.

[19] C. J. van Rijsbergen. Probabilistic retrieval revisited. *The Computer Journal*, 35(3):291–298, 1992.

[20] S. K. M. Wong and Y. Y. Yao. On modeling information retrieval with probabilistic inference. *ACM Transactions on Information Systems*, 13(1):38–68, 1995.