Segmenting and Labeling Query Sequences in a Multidatabase Environment

Aybar C. Acar[1] and Amihai Motro[2]

[1] Bilkent University, Department of Computer Engineering,
Ankara 06800, Turkey
aacar@cs.bilkent.edu.tr

[2] George Mason University, Department of Computer Science,
Fairfax, VA 22030 USA
ami@gmu.edu

Abstract. When gathering information from multiple independent data sources, users will generally pose a sequence of queries to each source, combine (union) or cross-reference (join) the results in order to obtain the information they need. Furthermore, when gathering information, there is a fair bit of trial and error involved, where queries are recursively refined according to the results of a previous query in the sequence. From the point of view of an outside observer, the aim of such a sequence of queries may not be immediately obvious.

We investigate the problem of isolating and characterizing subsequences representing coherent information retrieval goals out of a sequence of queries sent by a user to different data sources over a period of time. The problem has two sub-problems: *segmenting* the sequence into subsequences, each representing a discrete goal; and *labeling* each query in these subsequences according to how they contribute to the goal. We propose a method in which a discriminative probabilistic model (a Conditional Random Field) is trained with pre-labeled sequences. We have tested the accuracy with which such a model can infer labels and segmentation on novel sequences. Results show that the approach is very accurate ($> 95\%$ accuracy) when there are no spurious queries in the sequence and moderately accurate even in the presence of substantial noise ($\sim$70% accuracy when 15% of queries in the sequence are spurious).

Keywords: Data Management, Information Integration, Query Processing.

1 Introduction

In many database applications users perform complex tasks by breaking them down to a series of smaller, simpler queries the results of which then become terms in a larger expression. While the user interface may be presenting this to the user as a simple button click, multiple queries are usually being evaluated at the DBMS level to satisfy the request. In some cases this is consciously done by the users themselves, usually when their user interface does not allow complex queries or if they are experimenting with the data in order to find the correct query.

Particularly in the area of virtual databases (multidatabases), this decomposition of queries into smaller components is a necessary part of the process. In a multidatabase the actual data resides in a multitude of data sources, each possibly with different schemas and access methods. Therefore a query posed to the multidatabase is broken into multiple components, each designed to retrieve a piece of the required information from the relevant local data source. For the purposes of this paper, we call the party posing the queries the 'user'. However, this user may just as well be an automated application integrating data from different sources (e.g. a mash-up).

Query consolidation [1] is the reversal of this query decomposition process. A consolidator attempts to predict the global query that produced a set of smaller component queries as a result of decomposition. Although it is not possible to uniquely identify a global query that produces a given set of component queries, it is possible to estimate some of the likely global queries. The research on consolidation so far assumes that the local queries are available as a sound and complete set.

However, in the real world, queries from a given user arrive as a sequence over time. There is no marker that indicates which queries in a sequence constitute a meaningful set. In the absence of such information, all the consolidating system has are logs of queries for local sources. The set of queries that are part of a global query must therefore be extracted from the sequence in the logs or collated in real time as these queries arrive. Furthermore it would be very useful to the consolidation effort to individually label the queries in a segment with their probable roles in the bigger picture. The purpose of this paper is the investigation of this problem.

In Section 2, some of the relevant literature and basic methods are introduced. Section 3 defines the exact problem, discusses relevant issues and the information available in the solution of the problem. Section 4 details our approach to the problem using conditional random fields. Section 5 presents experimental results on two different scenarios. Finally Section 6 concludes the paper with a summary of findings and future work.

2 Background

2.1 Session Identification

Most of the previous literature on session identification and analysis of query logs is geared towards Web server access log mining. In access log mining, sequences of document requests from a Web server are analyzed instead of database queries. A second form of server log analysis is the analysis of keyword and parameter queries sent to search engines and other information retrieval services (e.g., digital libraries, travel sites, &c.) This type of analysis is based on clustering queries in order to find trends [2] and other clues as to the general information demands of the users in order to better design services [10].

In terms of session identification, the most prevalent and simplest method of isolating user sessions is the *timeout* method. In the timeout method, the user is assumed to have started a new task after a given amount of time has passed without activity. For Web information retrieval the threshold is reported [8] to be 10 to 15 minutes, for optimal separation of subsequent user sessions. Another method proposed in [6] is based on the fact that the users spend less time on auxiliary pages (navigation pages &c.) than on content pages. The method assumes that a session is finished when a user navigates to content page and times out. The timeout method is applicable to the database query environment as well, as a supportive addition to the method proposed in the following section.

Finally, a method is proposed in [5] called *maximal forward reference*. In this method the user is assumed to be in the same session as long as the next page requested has a link from the current page. This approach is relevant to the problem at hand as it uses a metric of relevance between two requests in order to cluster them into the same session. A similar approach is used in the method proposed in the following section, to cluster relevant queries together.

As far as query log analysis in database environments, there is one relevant study [18] that uses methods based on statistical language analysis in order to do session identification. Their method is based on training a statistical model with previous examples, predicting session boundaries where a sequence of queries becomes improbable with respect to previous experience. The method differs from the one proposed by this paper in that it requires an in-order arrival. These concepts will be discussed further in the following section.

2.2 Conditional Random Fields

A common approach to segmenting sequence data and labeling the constituent parts has been the use of probabilistic graphical models. Particularly, *hidden Markov models* (HMMs) [16] have been used for sequence data such as natural language text, speech, bio-sequence data and event streams. An HMM defines the joint probability distribution $p(\mathbf{X}, \mathbf{Y})$, where X is a random variable over possible sequences of observations and Y is a random variable over the possible sequences of labels (i.e. the possible states of the process generating said observations) that can be assigned to the same input. Because they define a joint probability, HMMs are termed *generative* models.

A generative model can be used to generate new sequences obeying a certain distribution defined by the parameters of the model. These parameters are either coded directly or optimized by using training data. However, given a certain sequence of observations, finding the most likely sequence of labels involves using Bayes' Rule and knowledge of prior probabilities. Furthermore, defining a joint probability requires enumerating all possible sequences, both for observations and labels. Given that the number of possible sequences increases exponentially with the sequence length, this is generally intractable unless strong assumptions are made about the independence of sequence elements and long-range dependencies thereof.

In contrast, a different class of graphical models, known as *discriminative models* model the conditional distribution $p(\mathbf{Y}, \mathbf{x})$ for a particular sequence of observations, x and cannot generate the joint distributions. Since the problem at hand when trying to label a sequence of queries is finding the most probable sequence of labels given a sequence of observations (i.e. the observation sequence is already fixed), discriminative models are better suited to the task.

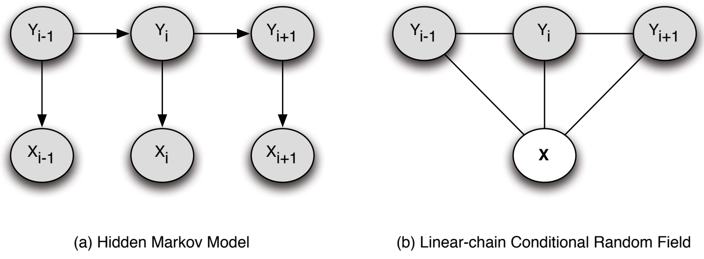

Fig. 1. Comparison of a first order hidden Markov model to a simple chain conditional random field

Such a discriminative model is the *Conditional Random Field* [12]. A conditional random field (CRF) can be regarded as an undirected graphical model, or Markov network [11], where each vertex represents a random variable. The edges of the graph represent dependencies between the variables. In the simplest case, the graph is a linear chain of state (label) variables Y_i dependent on the previous state and globally conditioned on the sequence of observations $\mathbf{X}$ (see Fig. 1b) but higher order and arbitrary graphs are possible. As long as each state variable Y_i obeys the Markov property (i.e., is conditionally dependent only on its clique in the graph), the graph is a conditional random field.

Given this structure, it is therefore possible to factorize the joint distribution of $\mathbf{Y}$ into potential functions, each one contained to a clique of vertices in the CRF. For example, in the case of a linear chain CRF such as that given in Fig. 1, the arguments of any such function will be Y_i, Y_{i-1}, and $\mathbf{X}$. As long as the feature functions are positive and real valued, the product will satisfy the axioms of probability, provided that it is normalized by a factor. Therefore the definition of any potential function f_k for a clique consisting of a particular observation sequence $\mathbf{x}$ and elements y_i and y_{i-1} of a particular state sequence $\mathbf{y}$ would be:

$$f_k(y_i, y_{i-1}, \mathbf{x}, i) = r \in \mathbb{R}, 0 \leq r \leq 1$$

The values of these potential functions can then be aggregated over the complete sequence to obtain the feature functions:

$$F_k(\mathbf{y}, \mathbf{x}) = \sum_{i=1}^{n} f_k(y_i, y_{i-1}, \mathbf{x}, i)$$

In trying to segment and label a sequence of queries, we are interested only in *decoding* a particular state sequence $\mathbf{y}$. In other words, given a particular sequence of observations we would like to find out the sequence $\mathbf{y}$ that would maximize $p(\mathbf{y}|\mathbf{x})$. This conditional distribution associated with the CRF is:

$$p(\mathbf{y}|\mathbf{x},\lambda) = \frac{1}{Z(\mathbf{x})} \exp \sum_{k=1}^{K} \lambda_k F_k(\mathbf{y},\mathbf{x}) \qquad (1)$$

where $Z(x)$ is the aforementioned normalization factor, and λ is the set of parameters (or weights) associated with each of the K feature functions. For any given application, the training of a CRF involves estimating these parameters. This parameter estimation is done using maximum likelihood training. Given a set of training sequences consisting of an observation sequence and the associated label sequence, $\{(\mathbf{x}^{(j)},\mathbf{y}^{(j)})\}$, the product of Eqn. 1 over all the j pairs of sequences is the likelihood. In maximum likelihood training the set λ that maximizes the log-likelihood function:

$$\mathcal{L}(\lambda) = \sum_{j} \left[\log \frac{1}{Z(\mathbf{x}^{(j)})} + \sum_{k} \lambda_k F_k(\mathbf{y}^{(j)},\mathbf{x}^{(j)}) \right] \qquad (2)$$

is found. The log-likelihood function is concave and differentiable. However, equating the differential to zero does not necessarily result in a closed-form in terms of the parameters. Therefore the parameters that maximize the likelihood cannot found analytically [17]. However the problem is amenable to numerical solution by iterative scaling [3,9] or quasi-Newtonian methods [14].

3 Methodology

3.1 Problem Overview and Assumptions

Throughout this paper, 'goal' will be used in the same sense as 'session'. In previous literature, the term 'session' is used in multiple meanings. Occasionally, it is used to indicate the range of transactions between a login-logout cycle or during a given timeframe. In the scope of our problem, however, a session is generally meant to denote a single information retrieval task of the user, either done in a single query or by combining multiple queries. In order to avoid this ambiguity, the set of queries that make up a task of the user will be referred to as a goal.

We start by assuming that we can identify users by some identification mechanism such as a cookie, through authentication, from their network address &c. Therefore the query sequence we are observing is for a single user at a time. However, the sequence is not necessarily of queries sent to the same data source. This data can be collected in two ways. The user can be monitored directly, either using the logs of the client she is using directly, or through the logs of a proxy or gateway she uses to access the sources. In this case, the provenance of the queries in the sequence is more certain (unless someone else is using the terminal). Alternatively, given a common identifier (such as IP or a cookie) we can monitor the logs of the data sources the subject uses. In this case, the queries belonging to the same user are collected from the various data sources and are compiled into a single sequence, in order of time stamps.

In either case we assume the existence of a list of queries in chronological order, coming from the same user. The queries, for the rest of this paper, will be labelled Q_i in a sequence $\mathbf{Q} = Q_1...Q_n$ such that Q_i is chronologically earlier than Q_{i+1}.

For each query Q_i, we assume to have access to a set of *intrinsic features.* These are given in Table 3.1.

Table 1. Intrinsic Features of Query Q_i

Name	Notation	Description
Attributes	$\mathbf{\Pi}_i$	The set of fields projected by the query.
Dependencies	$\mathcal{F}_i^*$	The functional dependency closure of fields in the query.
Constraints	$\mathbf{C}_i$	The set of constraints associated with the query.
Source	D_i	The unique identifier of the data source the query was sent to.
Answer Set	$\mathbf{R}_i$	The set of tuples returned in response to the query.
Timestamp	t_i	The universal time at which the query was posed.

We assume that these pieces of information are available for every query in the sequence. These intrinsic features will be used to generate more detailed feature functions which will in turn be used to train and decode the CRF.

We only consider conjunctive queries. Hence, each query in the sequence is represented by a Horn clause. For example, consider a query sent to an online bookstore's data service, asking the ISBN, title and year of books by Dickens:

$$\begin{aligned} Q_{ex} = \{(ISBN, Title, Year) | \exists AuthorID(Author(''Dickens", AuthorID) \\ \wedge Book(ISBN, Title, Year, AuthorID))\} \end{aligned}$$

This query might then have the following intrinsic features:

$$\begin{aligned} \mathbf{\Pi}_{ex} &= \{ISBN, Title, Year\} \\ \mathcal{F}_{ex}^* &= \{ISBN \rightarrow Title, ISBN \rightarrow Year, ...\} \\ \mathbf{C}_{ex} &= \{(Author.Name = \text{``}Dickens''), (Book.AuthorID = Author.ID)\} \\ D_{ex} &= \text{http://soap.mybooks.com/Search.wsdl} \\ \mathbf{R}_{ex} &= \{(\text{0679783415, ``David Copperfield''}, 1850), ...\} \\ t_{ex} &= \text{2011-04-24T20:39Z} \end{aligned}$$

Given a query sequence and the intrinsic features of each member of the sequence, one needs to solve two distinct problems:

Segmenting. Otherwise known as *boundary detection*, this involves finding when one goal in the sequence ends and the other begins.

Labeling. Within each segment (goal) we also seek to find the function of each member query. This aids in reproducing the original goal exactly.

Both problems have analogues in natural language processing. The segmenting problem is analogous to *sentence boundary detection* and the labeling is very similar to *part-of-speech tagging.* In order to induce the most useful features for the task, we now consider a model for the sequence structure.

3.2 Assembly of Goals

In the simplest case, the goal, G, is the conjunction of the queries Q_i in the sequence. In terms of relational algebra, this is essentially the joining of the answer sets, R_i, into a single universal relation. However, it may be the case that the component queries do not offer a join path. Consider a sequence of two queries, Q_1 and Q_2 with attributes $\mathbf{\Pi}_1 = \{SSN, Name\}$ and $\mathbf{\Pi}_2 = \{Phone, Address\}$. The straightforward conjunction (join) of these will result in a cartesian product, which is meaningless. Therefore, perhaps it is better to consider these two queries as two separate goals of one query each, rather than a goal of two queries. However, the arrival of a third query Q_3 with attributes $A_3 = \{SSN, Phone\}$ would change the conclusion. The three queries can now be joined into a goal consisting of three queries.

The problem of isolating goals from a sequence of queries can be stated as follows: Given a sequence of queries $\mathbf{Q} = Q_1 ... Q_n$ coming from a user, find all user goals within the sequence that are *maximal and coherent.*

We define a goal, $\mathbf{G}$, therefore, as the largest subsequence, $\mathbf{G} \sqsubseteq \mathbf{Q}$, of the whole query sequence $\mathbf{Q}$, that is cohesive (i.e. that has a join path). We formalize this idea of coherence by defining the following feature between any two queries Q_i and Q_j:

$$joinable(i,j) = \begin{cases} 1 & \text{if } \exists \mathbf{X} \subseteq \mathbf{\Pi}_i, \mathbf{Y} \subseteq \mathbf{\Pi}_j \left((\mathbf{X} = \mathbf{Y}) \wedge (\mathbf{X} \rightarrow \mathbf{\Pi}_i \vee \mathbf{Y} \rightarrow \mathbf{\Pi}_j) \right) \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

The *joinable* function is thus 1 only if there is the possibility of a lossless join between the answer sets of the two queries. This lossless join property can be checked using the closures, $\mathcal{F}^+$, of the queries. Note that, in evaluating losslessness we only check functional dependencies and not inclusion. Considering the fact that constituent relations originate from independent sources, enforcing or requiring inclusion would not be realistic.

So in this simple case, a goal goes on as long as the incoming queries are joinable to the existing ones in the goal. We call these queries "subgoals". Ultimately, all subgoals are joined to assemble the goals. We now consider subgoals that are composed of more than a single query.

3.3 Composite Subgoals

We define a composite subgoal as one that is created by the disjunction (union) of its constituent queries.

In constructing a composite subgoal, a user posing a sequence of queries to several sources is possibly collecting information from each of them in order to create a more comprehensive view of the information. An example would be to send a query selecting for the Italian restaurants in Washington D.C. to two or more different dining registries. The answers obtained are more likely to be unified rather than intersected since the aim here is to garner as many answers as possible.

Another case which requires aggregation rather than cross-referencing is when the complete answer cannot be obtained in a single query due to particular limitations of the data source. The most common reason for this are the *variable binding limitations* imposed by the query languages of the data sources.

As a simple example to this, consider an online telephone directory for some large organization. Generally, the service will not have the option of listing all the telephones at once. More likely, the directory will require the user to constrain (bind) at least one variable, such as the last name, department, &c. A user wanting the whole list will then be forced to ask for one department at a time, or worse, a name at a time. Monitoring such a series of queries, it would be wrong to label them as a conjunction when clearly the intention is to combine the answers into a large list.

The previous example also hints at a another aspect of the problem: We have to assume that the user knew or obtained the list of departments (a table itself) from somewhere, then iteratively expanded each item with the associated telephone listings. Which in turn means that some query sequences may involve recursive queries. As a more elaborate example, consider a bibliography database that limits binding by requiring that some information about a paper (e.g. title) be given. The only way a user can get all papers, or at least as many papers as possible, is to start with one paper he knows and to recursively expand the citations until no new papers can be found.

Recursive plans may be used in case of poorly structured data services as well. Consider an airline reservation system which only lists direct flights given a source or destination airport. A user trying to get from Washington to Havana will not be able to find a direct flight. Instead, the user will ask for flights out of Washington and will ask followup queries asking for possible destinations from those places, until he can recursively find a path to Havana with as few stopovers as possible (e.g. Washington-Toronto-Havana). Again, the aim here is not to join the answers to these queries, so they need to be treated differently.

Each of these examples illustrate a different form of recursion. In the case of the bibliography example, a depth-first traversal is the best option. In the case of the airline example, breadth-first or iterative deepening is needed as depth first recursion will probably be unnecessarily expensive. A third form of traversal is the case where the user recurses using some form of heuristic. Assume the user wants to reach a certain person in a social network service. Similar to the bibliography example, the only way she can accomplish this is to find a string of friends that can connect them. If the user knows that the person lives in a certain city, she may direct the recursion at each level to expand only people living in that city, knowing that this will result in finding a path more rapidly.

Ultimately, there is one common property of recursive query plans that allows and outside observer to identify them as such. Each query in such a sequence will bind one of its attributes to a value obtained from the result of a previous query in the sequence.We can formalize this using the intrinsic features of queries previously defined. The feature indicating that two queries Q_i and Q_j may belong to a recursive sequence is given as:

$$recursive(i,j) = \begin{cases} 1 & (i < j) \wedge \exists x \in (\mathbf{\Pi}_i \cap \mathbf{\Pi}_j); \exists y \in \pi_x(\mathbf{R}_i) \, [(x = y) \in \mathbf{C}_j] \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

In creating composite subgoals, a requirement is that the results of the queries, that are to be part of the subgoal, be *union-compatible* as defined in relational algebra. In other words, it is not possible to unify two results unless they have the same arity and indeed not meaningful unless they share the same attributes. Therefore, we need a measure to indicate the overlap in attributes between two queries. This can be simply stated as:

$$overlap(i,j) = \frac{\mathbf{\Pi}_i \cap \mathbf{\Pi}_j}{\mathbf{\Pi}_i \cup \mathbf{\Pi}_j} \tag{5}$$

The binary feature associated with overlap is the complete overlap:

$$completeoverlap(i,j) = \begin{cases} 1 & \text{if } overlap(i,j) = 1 \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

An analogous and useful feature would compare the overlap in the set of constraints between two queries:

$$constraintoverlap(i,j) = \frac{\mathbf{C}_i \cap \mathbf{C}_j}{\mathbf{C}_i \cup \mathbf{C}_j} \tag{7}$$

Thus far, the features we have considered have been totally about the queries themselves. When working in a multidatabase environment, the provenance of the queries is also very useful. In other words, the source to which the query has been sent and from whence the answer originated might further help in defining the role of the query. Particularly if two or more subsequent queries are sent to the same source, there is a higher likelihood that the user is attempting a unification or recursion. In terms of query optimization it is more efficient to ask everything in one query per source. However, if for some reason (e.g. binding limitations or a less expressive query language at that particular source) the user is not able to accomplish their subgoal in one query, they will be required to collect the data in pieces. For example, a complete overlap between two queries sent to the same source is a good indicator that the user is combining subsets (e.g.

phone numbers in different departments) of the same information. We therefore introduce the following feature:

$$samesource(i,j) = \begin{cases} 1 & \text{if } D_i = D_j \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

3.4 Application of Conditional Random Fields

Apart from the complexity of assigning relevance to goals there is some difficulty associated with building the goals in the first place. Several aspects of the sequence will give clues as to the generation of goals:

Do the goals arrive one at a time? The question here is whether the user is done with one task before she starts another one. If this is not true, parts of different goals may arrive interleaved with each other. Without interleaving, the problem of goal identification is simply a problem of boundary detection, namely finding the first query of each goal. Otherwise, goals have to be extracted explicitly.

Do goals evolve coherently? This is related to whether the queries within a goal arrive in the correct order. For the goal to be coherent, each arriving query has to be relevant to at least one other query that is already part of the goal. If the queries arrive out of order in the worst case it is possible that there will not exist a coherent goal until the last query of the goal arrives. As an example consider the queries $\mathbf{\Pi}_1 = \{A,B,C\}$, $\mathbf{\Pi}_2 = \{K,L,M\}$, $\mathbf{\Pi}_3 = \{X,Y,Z\}$ and $\mathbf{\Pi}_4 = \{A,K,X\}$. A goal consisting of these queries will evolve coherently only if Q_4 arrives first. Whether or not this is assumed to be true will determine the solution method.

Are queries used by multiple goals? This factor determines the number of possible goals that can be generated given a sequence of length N. If we assume that each query can be part of multiple goals, the set of possible goals in the query sequence $\mathbf{Q}$ will be the power set of $2^{\mathbf{Q}}$. If it is assumed that each query in $\mathbf{Q}$ belongs to just one goal, the set of possible goals becomes the partition of $\mathbf{Q}$. The number of parts in $\mathbf{Q}$ can be at most N, i.e. single-query goals.

In the simplest case, it can be assumed that queries arrive in order, in a non-interleaved fashion and that each query is part of a single goal only. In this case, the goal identification starts combining arriving queries into a goal. The first query to violate the coherence of the goal will be deemed the beginning of the next goal and the previous goal will be closed.

In the most complicated case, it is accepted that queries can arrive out of order, interleaved with queries of other goals and certain queries might be used by multiple goals. In this case the system will have to keep track of multiple goals at once, evolving some or all of them as new queries are received.

For the purposes of this paper, a middle ground between these two extremes is assumed. We assume that each query will be part of one goal only. Interleaving will be neglected. The in-order arrival of queries is not a requirement to the extent that the sequence can be kept in memory. As will be seen subsequently, we work in a sliding window on the sequence starting from the last query going back as far as the window size. A query arriving prematurely will still be added into its goal as long as it is still within this window.

We start by defining a finite state machine which will take one query at a time and transition from one state to another. The state to which the FSM transitions to after the arrival of a new query will become the label of that query. The outline of the state machine is given in Fig. 2

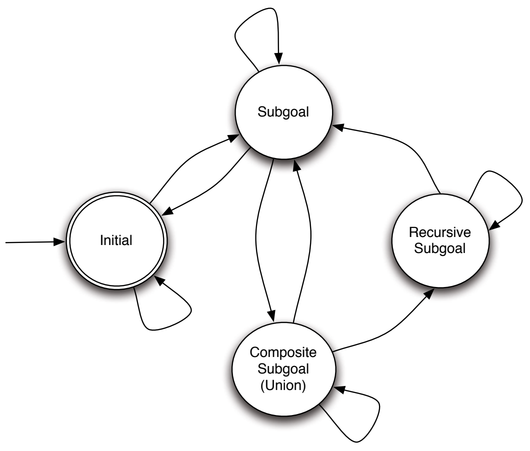

Fig. 2. Labeling Finite State Machine. Only relatively likely transitions are shown.

We now define this finite state machine as a probabilistic one, describing it with a conditional random field. The probability distribution is thus calculated as given in Eq. 1. In our case, the sequence $\mathbf{x}$ is the query sequence $\mathbf{Q}$ (i.e. the observations) and sequence $\mathbf{y}$ is the sequence of states, $\mathbf{L}$ the FSM traverses (i.e. the labels). Each transition of the FSM to the *Initial* state denotes the beginning of a new goal. After each arrival of a new query (observation) the CRF is decoded in order to find the state (label) sequence $\mathbf{L}$ that results in the highest conditional probability $P(\mathbf{L}|\mathbf{Q})$. Hence, each time a new query arrives, the conditional probability is recalculated and may therefore change the labels on earlier queries as well.

Figure 3 shows an example sequence. Also provided in the figure are the operations performed on the queries by the user. The latter information is naturally not available to the party monitoring the queries but serve to illustrate the example. Table 2 illustrates the labeling for the incoming queries incrementally as each one arrives. The first two queries are joinable, therefore they are labeled as the initial query and a conjunctive subgoal. The third query initially has no relationship to the preceding two. It is therefore labelled as the beginning of a new goal. The next few are found to be queries most likely to be combined (i.e. union-ed) with the third query and are thus labelled. The sixth query is again found to be unrelated to the existing goals and labelled as the beginning of a new goal. Query 7 happens to be the "keystone" query in that it has join paths to both query 6 and the previous goals. Therefore when query 7 arrives, the labels on the previous queries are changed, coalescing the three goals into one. This is possible since with each new query, the labels of all the previous queries are reexamined and a new Viterbi path[1] is calculated for the probabilistic FSM. Subsequently, query 8 arrives and is first deemed to be a single subgoal. However, as more evidence arrives, i.e. queries 9 and 10, it now becomes more probable that the three constitute a recursive subgoal and thus the labels are rearranged.

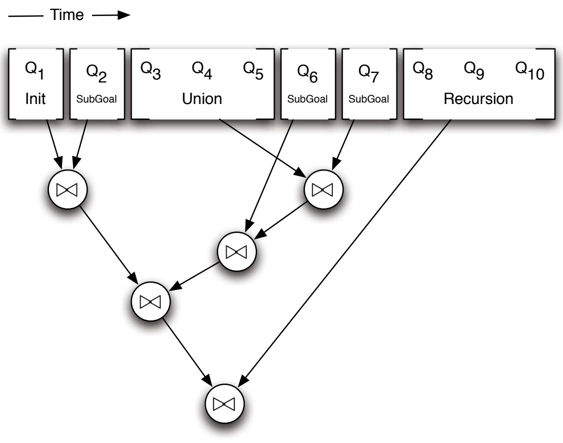

Fig. 3. Consolidation of Example Sequence

3.5 Features of the Conditional Random Field

In order to complete the definition of the CRF that represents this probabilistic FSM, the set of features $F_k(\mathbf{L}, \mathbf{Q})$ have to be defined. There are two forms of features, namely *state* and *transition* features. The state features are defined in terms of identity functions. There is an identity function for each label type and each position in the sequence. For example the identity function defining whether the label L_i is the initial state would be:

$$initial(i) = \begin{cases} 1 & \text{if } L_i = INITIAL \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

[1] The sequence of transitions with highest probability.

Table 2. Evolution of Example Sequence

Time	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_{10}
1	**I**									
2	I	**S**								
3	I	S	**I**							
4	I	S	I	**U**						
5	I	S	I	U	**U**					
6	I	S	I	U	U	**I**				
7	I	S	**U**	U	U	**S**	**S**			
8	I	S	U	U	U	S	S	**S**		
9	I	S	U	U	U	S	S	**R**	**R**	
10	I	S	U	U	U	S	S	R	R	**R**

I: Init, S: Subgoal, U: Union, R: Recursion
New or updated labels are shown in boldface

There will be one such identity function for each label per query. For example, given a sequence of 10 queries and the 4 labels defined in the FSM (i.e. Initial, Subgoal, Union, and Recursion), there will be 40 identity functions. Rather, we say that there are 4 state features per clique in the CRF.

The basic transition features have already been given in equations 3 through 8. Since conditional random fields are log-linear models, in order to capture the query model correctly we need to use conjunctions of the basic features (eqns. 3-8). For example, a compound feature particularly useful in determining recursion is ($completeoverlap \wedge recursive \wedge samesource$). Up to this point, for each clique in the CRF, we have 9 basic features (those given in Eqns. 3-8 and two features for absolute and relative timeout) and 4 state identity features. Along with the possible conjunctions (i.e. the powerset of the 13 features) we may consider up to 4095 features in evaluating the CRF.

We also need to define the scope of each of the features. Recall that a feature function is $f_k(L_i, L_{i-1}, \mathbf{Q}, i)$. However, the functions we have defined so far take two arguments (queries) to be compared according to some criterion (e.g., joinability, recursive nature &c.). For each clique, one of these queries is the current one (i.e., Q_i). What this query is being compared to in the sequence is dependent on the scope. We propose two different scopes, namely local and global. The global scope effectively compares all the queries in the range to Q_i and succeeds if there is at least one successful comparison. For example the feature $f^{global}_{samesource}(L_i, L_{i-1}, \mathbf{Q}, i)$ would succeed if there is at least one other query in the sequence that has the same source as Q_i. Therefore given a basic feature $g_k(Q_i, Q_j)$ a global feature is then:

$$f^{global}_k(L_i, L_{i-1}, \mathbf{Q}, i) = \max_{\forall j \neq i}(g_k(Q_i, Q_j))$$

In contrast, a local feature only checks the queries proximal in the sequence to Q_i. Hence, for example for a window of 5, the local feature would be:

$$f_k^{local}(L_i, L_{i-1}, \mathbf{Q}, i) = \max(\max_{i-5<j<i}(g_k(i,j)), \max_{i<j<i+5}(g_k(i,j)))$$

Note, however, that the state features previously mentioned and any conjunctions containing them can only be local within a window of 2. Otherwise, the probability of a given label would not be independent of previous states except the neighboring one, violating the Markov property.

The final enrichment to the feature space involves the *time-shifting* of features, such that clique potentials can be affected by previous and future observations. A time shifted feature $f_i^{<j>}$ would be defined as:

$$f_i^{<j>} = f(Q_{i+j}) \tag{10}$$

So, a feature $f_i^{<-2>}$ would be the said feature for the observation two queries before Q_i used to evaluate the potential for the i-th clique in the CRF. Again this is limited to the transition features since the potential of a given clique cannot be a function of the label of state not neighboring it. Therefore, features containing any of the state identity functions can only be shifted by -1. According to these restrictions and assuming we use time shifts of -2, -1, 0, 1, 2, global and local forms for all the possible features, we may have up to 12278 features per clique.

3.6 Gain-Based Feature Selection

Note that the previous estimate is the combinatorial maximum of features obtained in the model we have thus far built. Some of these features will not be predictive at all and we thus need to prune those that have little utility in order to keep the model manageable and more importantly to avoid over-fitting the training data.

In order to do this, we use gain based feature selection. Recall that the training of a CRF involves optimizing the weight parameter vector λ in the log-likelihood function (Eqn. 2) in order to obtain the maximum likelihood. This is done in supervised way by iteratively maximizing for a set of training data. Each training sequence of queries has an associated sequence of correct labels. Thus we are able to optimize the parameters, λ, to obtain the weights that will have the least training error. Since some of our features are expected to be less discerning, the changes in the associated weights will have less impact on the training error. We can therefore use the method described in [15]. The gain of a feature f is defined as:

$$G_\Lambda(f) = \max_\mu(\mathcal{L}_{\Lambda+f\mu} - \mathcal{L}_\Lambda)$$

where $\mathcal{L}_{\Lambda+f\mu}$ and $\mathcal{L}_\Lambda$ are the log-likelihoods of the CRF with or without the feature f, given the training data. If done naively, this gain calculation will be very time consuming, given the large number of features. Fortunately, there are several optimizations including mean field approximation and limiting the gain calculation only to mislabeled outputs (cf. [15] for details). Using this gain-based pruning we show in the next section that we are able to use approximately a tenth of the features while gaining better accuracy and better generalization.

4 Experimentation

We performed some simple experiments on the approach thus far explained in order to validate its accuracy and examine its behaviour. Two publicly available benchmark databases,TPC-H[2] and Mondial[3], were used. Of these, TPC-H is the simulated database of a business including supplier and customer information, invoices, orders and so forth. It has 8 tables. These 8 tables, for the purposes of our experiment, are treated as different data sources. Furthermore, the larger tables have been vertically sliced into smaller parts, each one as a different source. This allows us to simulate vertically distributed data in a multidatabase environment. Therefore, with the subdivisions our TPC-H test scenario simulates 17 different sources in a simulated global virtual database (multidatabase) schema.

The Mondial database has a larger schema and is populated with real-world geopolitical data from several resources (e.g. the CIA World Fact Book). Originally it has 23 tables, the larger of which were likewise vertically divided in order to obtain 30 simulated data sources.

Some parts of the divided tables were given binding limitations (e.g., the Countries table in Mondial can only be searched by setting name of the country).

In order to create training and test data for the CRF, we then automatically generate 500 query sequences for each. This was done by first creating a universal relation for each test scenario. Random views of these universal relations were then set as targets and a query plan (i.e., a query sequence) was generated for each target, along with the labels for each of the components. Due to the differences of the schemas and the random nature of the target selection, the lengths of the query sequences are distributed over a range of values. For Mondial, the sequences were between 1 and 36 queries long, with a median of 10. For TPC-H the sequences were betweem 1 and 23 queries long, with a median of 7.

The 500 sequences where randomly concatenated into groups of 10, each group becoming a simulated query log for one user. The CRF was trained and tested using leave-one-out cross-validation. Each database scenario was run 50 different times with different training and test cases.

Three different CRF scenarios were experimented with. The CRF experiment is the basic approach with all the possible features. The PrunedCRF experiment is the CRF scenario trained with gain-based pruning of attributes during training. The training and testing programs were implemented in Java and the experiments were run on a 2.2 MHz Athlon 64 system with 1 GB of memory operating under FreeBSD 6. Each cross-validation experiment (i.e. 50 runs) was completed in times on the order of 1-2 hours, on this platform, depending on the number of features. The inference (labeling) of the average test group took around 1.5 seconds. The unpruned method (CRF) has exactly 12,768 features for both data sets, the pruned CRF had 1,065 and 983 features for Mondial and TPC-H, respectively.

[2] http://www.tpc.org/tpch/
[3] http://www.dbis.informatik.uni-goettingen.de/Mondial/

The training was done using gradient ascent, particularly the L-BFGS [14] method. To avoid overfitting, the likelihood function was regularized with a Gaussian prior with variance 5. After each round of training with 49 groups the remaining test group was labelled and the labels were compared to the true labels which were decided during the generation of the query sequences. The accuracy of a test is defined as the percentage of labels that are the same with the true labels. Table 3 presents the average accuracy of each cross-validation run, one per scenario.

Table 3. Labeling Accuracy Results

		Test Accuracy				Test Accuracy	
Noise Level	Method	TPC-H	Mondial	Noise Level	Method	TPC-H	Mondial
0 %	CRF	94.6%	91.2%	10 %	CRF	73.2%	70.6%
	PrunedCRF	96.3%	95.4%		PrunedCRF	82.8%	79.5%
5 %	CRF	86.3%	88.1%	15 %	CRF	60.3%	65.1%
	PrunedCRF	95.2%	92.3%		PrunedCRF	68.6%	69.3%

The robustness of the CRF method to noise was also tested during these experiments. For each cross-validation run, after the training of the CRF, the test sequence was tested for accuracy initially as it was. Subsequently the test cases were each “corrupted” by inverting the values of random observation features in the test sequence. For example, if the one query was actually joinable with another, the noise induction might flip the value showing the queries as non-joinable. With this method, we aim to simulate user and observation errors. The tests were run at noise levels of 5, 10, and 15%, relating to that fraction of observation features being corrupted. Table 3 shows the results.

The results are encouraging in that even at relatively high noise levels the feature pruned CRF and the content enriched CRF are still reasonably accurate (on average above 95% accuracy at 1-in-20 error in observation). The basic CRF seems to suffer immediately from the noise, probably due to the tendency of the unpruned features to over-fit the training data. Therefore, we see that pruned feature spaces have the additional benefit of lower generalization errors, even in the case of noisy input.

As a subset of the preceeding analysis, we have also investigated the boundary detection capabilities of each model. This purely to test whether the system segments the sequences correctly, regardless of the correct labeling of these segments.

The F1 scores of the boundary detection are given in Table 4. The segmentation behavior of the CRF is much better than the more complicated task of labeling, with very high accuracies up to very high noise levels. As with the labeling task, the basic, unpruned CRF suffers as noise increases, for the same reasons.

Table 4. Segmenting Accuracy Results

Noise Level	Method	Accuracy (F-1) TPC-H	Accuracy (F-1) Mondial	Noise Level	Method	Accuracy (F-1) TPC-H	Accuracy (F-1) Mondial
0 %	CRF	.992	.997	10 %	CRF	.971	.971
	PrunedCRF	.999	.999		PrunedCRF	.982	.988
5 %	CRF	.984	.973	15 %	CRF	.912	.903
	PrunedCRF	.987	.982		PrunedCRF	.938	.962

5 Conclusion

This paper investigates the problem of isolating sets of queries denoting independent user goals from a continuous sequence, either in real time or from a log. The complexities involved can be classified into two categories. The first category of difficulties is defining the boundaries of a goal. The second difficulty is the determination of the individual parts within a goal, regarding their function in forming the goal.

Even if one has a totally objective way of evaluating a given set of queries as to whether they contain an interesting goal, there are still 'technical' difficulties. Given a sequence of queries the set denoting a goal is not necessarily an uninterrupted sub-sequence. Two or more goals may be interleaved with each other or with noise. Some queries in the sequence may be replacements or refinements of older queries. Furthermore, it is possible that the queries needed to build a goal do not arrive in the order they are used. This defeats machine learning methods (e.g. [18]) which are sensitive to the sequence in which queries arrive. Another question is whether the user being monitored is using the same query for multiple goals or one. This distinction affects the number of goals to be evaluated greatly[4].

A probabilistic and noise-tolerant method of handling the problem has been introduced. The method can accept realtime streams, constantly evolving a working scenario as new queries arrive. Scenarios that have stopped evolving are dropped for fresh ones as new evidence arrive. Furthermore, the method proposed here leverages the evidence available exclusively in a virtual database environment, namely the source information, to guide its decisions.

A future direction is the addition of semantic feature based on the data. Such a method will require semantic information about the application domain either in a latent fashion or explicitly using the constraints of the schema along with a logical inference engine. Such an approach has met with some success in the area of semantic query caching [7] and optimization [4,13]. The same principle can be applied to the present problem as well.

[4] If the component queries are assumed to be re-used, the number of possible goals increases exponentially, as opposed to a linear growth otherwise.

References

1. Acar, A.C., Motro, A.: Inferring user goals from sets of independent queries in a multidatabase environment. In: Ras, Z., Tsay, L.-S. (eds.) Advances in Intelligent Information Systems. SCI, vol. 265, pp. 225–243. Springer, Heidelberg (2010)
2. Beeferman, D., Berger, A.: Agglomerative clustering of a search engine query log. In: Proceedings of Knowledge Discovery and Data Mining, pp. 407–416 (2000)
3. Bilmes, J.: A gentle tutorial on the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. Technical Report ICSI-TR-97-021, University of Berkeley (1997)
4. Cardiff, J., Catarci, T., Santucci, G.: Semantic query processing in a heterogeneous database environment. Journal of Intelligent and Cooperative Information Systems 6(2), 151–192 (1997)
5. Chen, M.-S., Park, J.S., Yu, P.S.: Efficient data mining for path traversal patterns. Knowledge and Data Engineering 10(2), 209–221 (1998)
6. Cooley, R., Mobasher, B., Srivastava, J.: Data preparation for mining world wide web browsing patterns. Knowledge and Information Systems 1(1), 5–32 (1999)
7. Godfrey, P., Gryz, J.: Semantic query caching for heterogeneous databases. In: Proceedings of Knowledge Representation Meets Databases, pp. 6.1–6.6 (1997)
8. He, D., Goker, A.: Detecting session boundaries from web user logs. In: Proceedings of the BCS-IRSG 22nd Annual Colloquium on Information Retrieval (2000)
9. Jin, R., Yan, R., Zhang, J., Hauptmann, A.: A Faster Iterative Scaling Algorithm for Conditional Exponential Model. In: Proceedings of the 20th Int. Conf. on Machine Learning, pp. 282–289 (2003)
10. Joachims, T.: Unbiased evaluation of retrieval quality using clickthrough data. Technical report, Cornell University, Department of Computer Science (2002)
11. Kindermann, R., Snell, J.: Markov random fields and their applications. American Mathematical Society, Providence (1980)
12. Lafferty, J., McCallum, A., Pereira, F.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: Proceedings of the 18th Int. Conf. on Machine Learning, pp. 282–289 (2001)
13. Levy, A.Y., Sagiv, Y.: Semantic query optimization in datalog programs. In: Proceedings of Principles of Database Systems, pp. 163–173 (1992)
14. Liu, D., Nocedal, J.: On the Limited Memory BFGS Method for Large Scale Optimization. Mathematical Programming 45(1), 503–528 (1989)
15. McCallum, A.: Efficiently inducing features of conditional random fields. In: Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI 2003), pp. 403–411 (2003)
16. Rabiner, L.R.: A tutorial on hidden markov models and selected applications in speech recognition. Proceedings of the IEEE 77(2), 257–286 (1989)
17. Wallach, H.: Efficient Training of Conditional Random Fields. Master's thesis, University of Edinburgh (2002)
18. Yao, Q., Huang, X., An, A.: A Machine Learning Approach to Identifying Database Sessions Using Unlabeled Data. In: Tjoa, A.M., Trujillo, J. (eds.) DaWaK 2005. LNCS, vol. 3589, pp. 254–264. Springer, Heidelberg (2005)