

User Interfaces for Process Modeling and Detection Systems

Yong Sheng

Thayer School of Engineering, Dartmouth College

8000 Cummings Hall, Hanover, NH 03755

yong.sheng@dartmouth.edu

ABSTRACT

User interfaces are important for process modeling and detection systems. This paper discusses the user interface design and implementation for the innovative Process Query System (PQS). Discussion focuses on the Hidden Markov Model (HMM) editor, model validation, process query types, and result visualization. A log likelihood indicator is used to evaluate and visualize the goodness of model-observation matching. A numerical method to measure the distinguishability between two HMMs is proposed and proved effective. The measurement is performed by estimating misdetection rate using the Monte Carlo simulation method. A visual HMM comparison tool using this method is implemented in the user interface.

Keywords: User Interface, modeling, process query system, hidden Markov model, model comparison, simulation

I. INTRODUCTION

1.1 Background

Process modeling is an effective computational methodology that is widely utilized in many application areas. The basic idea is to consider the behavior (observations) of dynamic systems as a result of the interoperation of correlated processes. Some processes are related to the intrinsic state of the object system, and others are noise. In order to study and understand the object system, one constructs the process models according to certain points of view and by certain modeling techniques. The constructed models can be used to identify, simulate, analyze, recognize, and detect object systems by computational algorithms corresponding to the modeling techniques.

We can only analyze the processes that generate observable outputs (can be characterized as observations). Processes may have various characteristics: a process can be stochastic or deterministic, stationary or non-stationary; the observations can be discrete or continuous in nature; the utilized sampling method may acquire observations synchronously or asynchronously. Many modeling techniques have been developed for processes with certain characteristics. Some well known techniques are Hidden Markov Models (HMMs)^[1, 7], Petri Nets^[2], Kalman Filters^[3], etc. All of these modeling techniques are effective for certain applications. Most of them are represented by abstract mathematical theories and sophisticated algorithms, however, which deters many potential users from applying these effective methods in their own studies.

The Process Query System (PQS)^[11] was developed to help researchers and engineers use these powerful methods without having to know the theories and algorithms extensively. The purpose of PQS is to provide integrated, scalable and customizable solutions for process modeling and detection applications. Just like database users can query specific record sets that satisfy certain criteria from massive and related data records, PQS users can “query” the processes that have the desired characteristics specified by certain models from massive and correlated observation sequences, with the built-in support for multiple major modeling techniques, algorithms, observation processing and management modules. Since most real world processes are stochastic, the query results of PQS are typically sets of hypothesis with a particular likelihood or confidence level, instead of deterministic record sets.

The user interfaces of PQS play an important role in achieving the overall objective of PQS. First, graphical user interfaces provides an intuitive way to construct, edit and understand models. Many types of models can be naturally represented as graphs, such as HMMs and Petri Nets. Second, a well designed user interface can help users understand and get used to the concise and intuitive concept of “process query”, and hide the arduous theories and algorithms behind the modeling techniques. Third, visualization of process query results is important to help users understand the query results deeply.

Currently, the PQS system and its client application support hidden Markov models only and all observation types must be converted or quantized to discrete values. We plan to support more modeling techniques, including but not limited to Petri Nets and Kalman filters in the near future. We also plan to support multiple observation types directly, including discrete symbols, continuous values, vectors, and custom-defined observation types. The reason to choose HMM as the first implemented modeling technique in PQS is that although HMM is quite simple and easy to implement, it is very powerful; most stochastic process modeling problems can be either reduced to or approximated by HMM with discrete observation.

The models in PQS act in the same way as criteria in traditional database queries. The criteria in database queries are typically boolean expressions, which are easy to construct, edit, understand and evaluate. The models in PQS are relatively abstract, intangible concepts and difficult to understand. For the example of HMMs, all the following questions are difficult to answer intuitively: what is the best HMM that describes a given observation sequence? How much difference will it make if we modify a parameter for a given HMM from 0.01 to 0.05? Are the two HMMs similar? A series of model manipulation tools for models are developed to help people find the answers qualitatively and quantitatively. These tools include an HMM editor, validation tool, training tool and comparison tool.

An innovative visual HMM comparison tool is implemented, which provides efficient statistical methods to measure the distinguishability between two HMMs. It utilizes the Monte Carlo method to simulate the two models and generate a large number of observation sequence samples for each model, respectively; meanwhile, it uses the two models to detect the samples and calculates the misdetection rate for each model's simulation respectively. By plotting the misdetection rate versus the length of observation sequences, one can get an intuitive and quantitative measurement of the distinguishability between the two models. This method is very effective in engineering fields because no matter what kind of topology and parameters the models have, it will produce meaningful results in relatively short time.

The remainder of this paper is organized as follows. Subsection §1.2 gives a brief introduction to related works. Section 2 provides an extensive discussion about the user interface design and implementation. We start with a short introduction to the PQS architecture, followed by the overview discussion of the PQS user interface. Then we discuss in detail the design of the HMM editor, query types and result visualization, respectively. Section 3 discusses our method of measuring distinguishability between HMMs, which is utilized to implement the visual HMM comparison tool mentioned above. Some interesting experiment results are given and discussed at the end of this section. Section 4 contains the conclusions and a brief discussion about future work.

1.2 Related Works

There are many well established tools kits for the most frequently used modeling techniques. Many of them have already been applied extensively in many research areas. For hidden Markov models: Hidden Markov Model Toolkit (HTK) ^[4] is an extensive HMM manipulation library specially designed for speech recognition applications; HTK provides both C programming interfaces and a command level user interface; the General Hidden Markov Model library (GHMM) ^[6] is another well known C library, which implements efficient data structures and algorithms for basic and extended HMMs. GHMM also provides a lightweight graphical HMM editor. HMMER ^[5] is an implementation of profile Hidden Markov Models (profile HMMs). Both GHMM and HMMER are mainly designed for biological sequence analysis. There are also many freely available or commercial packages or toolsets for Petri Nets including graphical editors and analyzers. Most existing projects are designed for special applications, and support no more than one modeling technique, while PQS is designed for general purpose and its user interface is integrated with extensive visual tools.

II. DESIGN AND IMPLEMENTATION

2.1 PQS Architecture

PQS is a distributed computing system, which consists of a number of cooperating functional components over networks. These components include data sources (sensors), Semantics-based Message Oriented Middleware (SMOM), tracking engines and user interfaces (Fig 1). Each component can either be a standalone application or act as a logical module in a large functional block. Entities in the data source layer are sensors, which is a generalized abstraction of all kind of data acquisition devices. The Semantics-based Message-Oriented Middleware (SMOM) provides message publishing and subscription service for the information flows between sensors and engines. The message publish/subscribe architecture brings the PQS system reliability, scalability, flexibility and extensive connectivity. Powered by the DAML+OIL ^[8] semantic technology, the SMOM service can direct messages reported by sensors to appropriate users. The core component of this architecture is the PQS engine, which provides built-in algorithms and

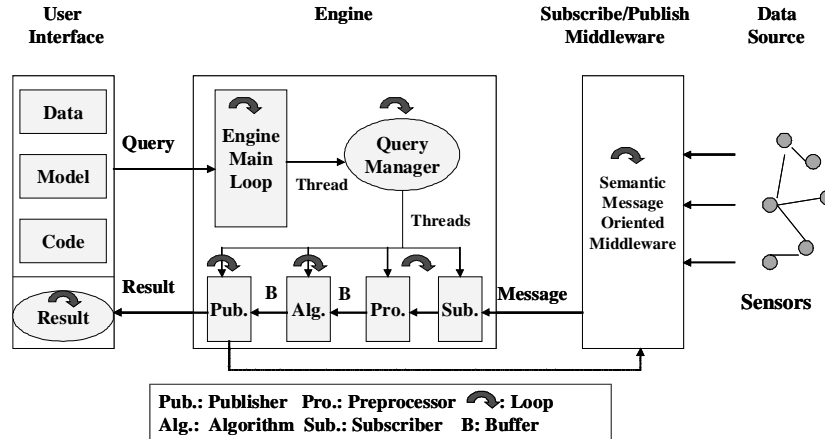


Fig 1 PQS Architecture

tools for the supported modeling techniques and data pre-processing modules. It handles the process queries submitted from the user side, subscribes to suitable data sources specified in the queries, executes the queries, reports the results to the user side, and publishes to the SMOM again if required by the query command. The user interface component is implemented as a standalone client application, which contains a hidden Markov model editor, SMOM server browser and data subscription editor, a preprocessor editor, query result visualizer and a series of model manipulation tools.

2.2 User Interface Design Overview

The basic work flow of a process query in the PQS client application is as follows. First, one can browse all available data sources (sensors) and generate data subscriptions for the desired data. Second, one designs suitable models that describe the characteristics of the desired processes. The structure and parameters of the models could be either specified manually, or generated automatically by training algorithms. Third, one should write a small piece of data preprocessing code that converts or quantizes sensors' output data to a standardized observation format that is acceptable to the engine algorithms. Fourth, one creates a process query, which contains the information of data subscriptions, models and preprocessing codes, and submits the query to the PQS engine. The engine will create a new thread to handle each incoming query and configure the thread with suitable subscriber, preprocessor, algorithm and publisher modules according to the query's requirement. By running the thread, the query is executed. The client application can get the query result sequence from the engine continuously until query execution is stopped or cancelled by the user. Finally, the query result will be visualized as tables, charts, and plots and represented to the user.

The PQS user interface is designed according to this work flow. The PQS client application consists of the following major components: the HMM editor and model validation, training and comparison tools; the SMOM browser, which allow users to browse the available data sources, and generate data subscription in DAML format; the preprocessor editor, which helps users to create, compile and debug (not implemented in the current version) observation preprocessing code in the Java programming language; the query submission and session control module, which allows users to submit multiple queries concurrently and manage the communication links between client and PQS engine for each query session respectively; the query result visualization module, which provides meaningful visualization of the hypothesis set and likelihood; and the application frame module, which manages all the other modules and provides an integrated interactive platform.

We will discuss the HMM editor and validation tools, query types and result visualization, and the visualized HMM comparison tool in detail in the reminder of this section.

2.3 HMM Editor

The HMM editor is one of the major components of the PQS user interface. Its design follows the conventions of popular diagrams editing software. Fig. 2 is the hardcopy of a PQS client application window, and Fig. 3 is a zoomed hardcopy of an editing HMM graph. Each state of hidden Markov models is represented as a filled circle, which is a vertex of the graph. Transitions between states are represented as directional edges. The name of a state is shown at the center of the state element, and the value of the transition probability is noted beside the corresponding edge element. The HMM

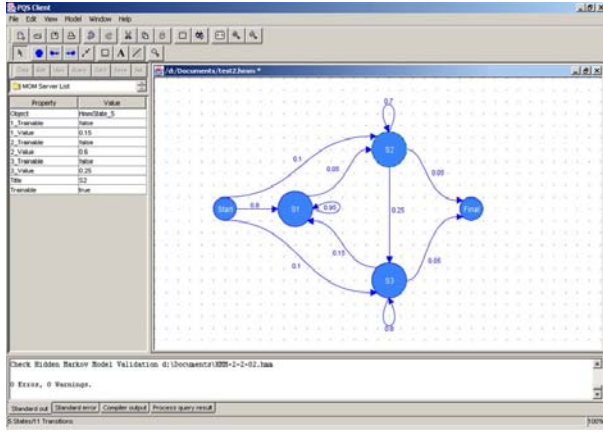


Fig 2 HMM Editor User Interface

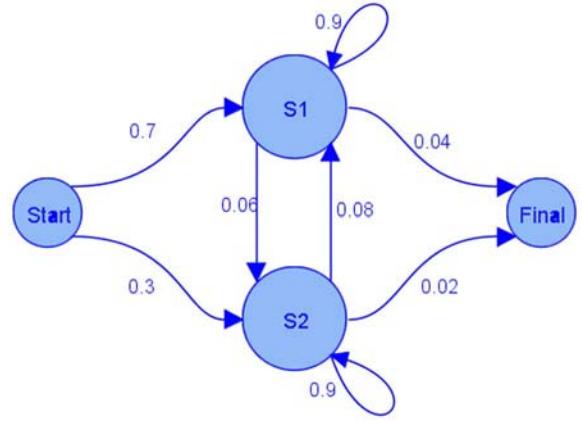


Fig 3 HMM with “start” and “final” states

editor supports many popular features, including copy and paste, undo and redo, zoom, drag and drop, show or hide grids and snap to grid points. A property editor is placed at the left side to the HMM editor window, which provides the means to specify the alphabet of observations for the whole model, and the emission probability distribution for each state.

2.3.1 “Start” and “Final” States

Beyond the core layout features, there are some special treatments designed for hidden Markov models only, such as the “start” and “final” states, which are introduced to define the initial state distribution and finite length sequence respectively. These states play specific roles in the HMM. A sample HMM with these special states is given in Fig 3.

As background, a hidden Markov model λ , is defined as

$$\lambda = \langle \mathbf{M}, \mathbf{N}, \mathbf{A}, \mathbf{B}, \boldsymbol{\pi} \rangle \quad (1)$$

\mathbf{M} denotes the finite state set $\{s_i\}$ of model λ and $i = 1 \dots m$; \mathbf{N} denotes the finite alphabet of observations $\{o_j\}$ and $j = 1 \dots n$. $\mathbf{A} = \{a_{ij}\}$ is a $m \times m$ matrix, where a_{ij} denotes the transition probability from state s_i to s_j , given by

$$a_{ij} = P(S_{t+1} = s_j | S_t = s_i) \quad (2)$$

$\mathbf{B} = \{b_{ij}\}$ is a $m \times n$ matrix, where b_{ij} denotes the emission probability of observation o_j given state s_i , given by

$$b_{ij} = P(O_t = o_j | S_t = s_i) \quad (3)$$

$\boldsymbol{\pi} = \{\pi_i\}$ is an m -dimensional vector, where π_i is the initial state probability of state s_i , given by

$$\pi_i = P(S_1 = s_i) \quad (4)$$

To represent the initial state distribution $\boldsymbol{\pi}$ in the HMM graph, the special “start” state is introduced. The rules for “start” states are:

- a “start” state is not a real state in $\{s_i\}$. It neither emits any observations, nor accepts incoming transitions;
- at most one “start” state is allowed in an HMM graph;
- each transition from the “start” to state s_i provides the value of π_i ;
- the “start” state is optional. If the “start” state is absent in a model graph, the initial state probability will be uniformly distributed, that is $\pi_i = 1/m$ for any i .

Since some processes generate finite length sequences, the special “final” states, accompanied by a special delimiter symbol δ , can be introduced to represent models for finite processes. The delimiter should not be the same as any known symbols in \mathbf{N} . The rules for the “final” state are

- the “final” state is optional. A model without a “final” state generates infinite sequence;

- if at least one “final” state exists, a hidden delimiter symbol δ should be added to the observation alphabet \mathbf{N} , producing new observation alphabet $\mathbf{N}' = \{o_1, \dots, o_n, \delta\}$. The delimiter δ will not be shown in the graphically HMM editor so it is hidden from the user’s point of view.
- a “final” state is an absorbing state, so no outgoing transitions are allowed.
- a “final” state emits delimiter δ only, with probability 1.
- all “final” states are equivalent.

2.3.2 Model Validation

Model validation is an important feature to help users find careless typos or serious errors in their model, especially if the structure of the model is complex. It guarantees the integrity of models being submitted to the PQS engine. The validity check basically applies the following rules:

- Both the state set \mathbf{M} and the observation alphabet \mathbf{N} should not be empty;
- Each state s_i and observation symbol o_j must have a non-empty and unique name;
- At most one transition is allowed between same pair of source and target states;
- Emission and transition probabilities should be between 0 and 1;
- The sum of all outgoing transitions probabilities from a certain state $\sum_{j=1}^m a_{ij} \equiv 1$;
- The sum of all emission probabilities for a certain state $\sum_{j=1}^n b_{ij} \equiv 1$;
- If the “start” or “final” state exists, apply the rules for “state” and “final” state accordingly.

If any errors are detected, the corresponding states or transitions will be highlighted, and the reason for errors will be recorded in a validation log file. Users can follow the log and fix the problems one by one.

2.4 Visualizing the Query Result

The PQS user interface supports multiple process query types for different applications. Queries can be classified by the number of models and the number of data sources associated with the query. The simplest query type has one model and one data source. Besides that, multiple model queries and multiple data source queries are two useful query types.

A multiple model query can help users to solve the following question: given a series of HMMs $\lambda_1, \dots, \lambda_k$ and an observation sequence over time $\mathbf{O}^T = o_{t=1}o_{t=2} \dots o_{t=T}$, which model describes \mathbf{O}^T best. A log likelihood indicator

$$L(\mathbf{O}^T | \lambda) = \frac{1}{T} \log(P(\mathbf{O}^T | \lambda)) \quad (5)$$

is used as the criterion to evaluate how well model λ matches observation sequence \mathbf{O}^T . $L(\mathbf{O}^T | \lambda)$ has a non-positive value. The closer $L(\mathbf{O}^T | \lambda)$ is to zero, the better λ matches \mathbf{O}^T . Besides this property, there are two more reasons to choose it as the criterion. First, we can get $P(\mathbf{O}^T | \lambda)$ efficiently by the forward algorithm ^[1, 7] in linear time complexity over the sequence length T . Second, it is proved in ^[9] that the log likelihood converges to the entropy rate $H(\mathbf{O} | \lambda)$ as T goes to infinity, denoted as

$$H(\mathbf{O} | \lambda) = \lim_{T \rightarrow \infty} [L(\mathbf{O}^T | \lambda)] = \lim_{T \rightarrow \infty} \left[\frac{1}{T} \log(P(\mathbf{O}^T | \lambda)) \right] \quad (6)$$

Further, we can define the indicator as a series of functions of sequence length T for each model λ_i as

$$L_{\lambda_i}(T) = L(\mathbf{O}^T | \lambda_i) \quad \text{where } i = 1 \dots k \quad (7)$$

In the case of online process querying, the observation sequence length increases over time. For each new incoming observation, the PQS engine will report the log likelihood value for each model on the sequence of all symbols observed so far. The PQS client application visualizes the result by plotting $L_{\lambda_i}(T)$ for all λ_i over T on same chart (Fig 4.a). By comparing the curves, a user can find out which model best describes the data source.

Sometimes users may use a multiple data source query to find out which of K data sources each generating $\mathbf{O}_1^T, \dots, \mathbf{O}_k^T$ is most likely to generate a sequence corresponding to a given model λ . Similar to a multiple model query, we still use the log likelihood indicator as the criterion. The curves to be plotted are the likelihood functions of sequence length T for

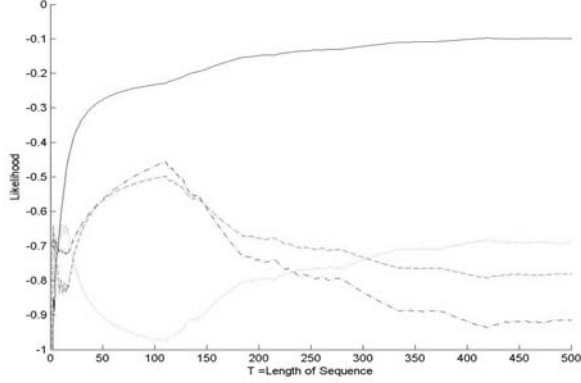


Fig 4.a Result of Multiple Model Query

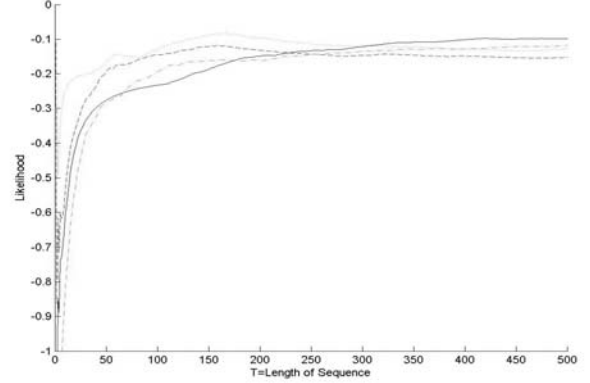


Fig 4.b Result of Multiple Data Source Query

each data source, given by

$$L_{O_i}(T) = L(\mathbf{O}_i^T | \lambda) \quad \text{where } i = 1 \dots k \quad (8)$$

Similarly, users can figure out the desired data sources from the visualized result plots (Fig 4.b) intuitively.

III. MEASURES OF DISTINGUISHABILITY BETWEEN HMMS

The interest of measuring the distinguishability between two hidden Markov models arises in many application areas in which HMMs are utilized to analyze sequences. The distinguishability between models is related to the misdetection rate. As with all probabilistic inference methods, the existence of misdetections is unavoidable while using HMMs, because noise exists in observations. The more distinguishable the HMMs are, the lower the misdetection rate will be if other conditions are the same. Various meaningful methods have been proposed in the literature to solve the problem analytically or numerically. The definition of the distance between model λ_1 and λ_2 is introduced by [10], in the sense of the difference of entropy rate

$$D(\lambda_1, \lambda_2) = \lim_{T \rightarrow \infty} \frac{1}{T} [\log(P(\mathbf{O}_{\lambda_1}^T | \lambda_1)) - \log(P(\mathbf{O}_{\lambda_1}^T | \lambda_2))] \quad (9)$$

where $\mathbf{O}_{\lambda_1}^T$ denotes sequences with length T generated by simulating model λ_1 . Similarly,

$$D(\lambda_2, \lambda_1) = \lim_{T \rightarrow \infty} \frac{1}{T} [\log(P(\mathbf{O}_{\lambda_2}^T | \lambda_2)) - \log(P(\mathbf{O}_{\lambda_2}^T | \lambda_1))] \quad (10)$$

And in general,

$$D(\lambda_1, \lambda_2) \neq D(\lambda_2, \lambda_1) \quad (11)$$

This definition gives theoretical guidance determining how different two HMMs are. The distance does not give a straightforward explanation to the misdetection rate, however, which concerns researchers and engineers more. Furthermore, in many cases we can obtain observation sequences with limited length only. For that typically the misdetection rate decreases as the length of the sequence grows, researchers are more concerned about how many observations are needed to fulfill performance requirements for the given HMMs λ_1 and λ_2 , than about how much the distance is between them. The PQS client application provides an efficient method to estimate the misdetection rate and represent it to users directly.

3.1 Methodology

The misdetection rate is defined as the expectation value of the probability that sequences (length= T) generated by one model λ_1 are detected as being generated by the other model λ_2 , which is given by:

$$e_{\lambda_2|\lambda_1}(T) = E[d_{\lambda_2|\lambda_1}(\mathbf{O}_{\lambda_1}^T)] = \sum_i P(\mathbf{O}_i^T|\lambda_1) \cdot d_{\lambda_2|\lambda_1}(\mathbf{O}_i^T) \quad (12)$$

where $\mathbf{O}_{\lambda_1}^T$ denotes the sequences (length= T) generated by λ_1 , and \mathbf{O}_i^T denotes the i^{th} of total n^T sequences with length T . $d_{\lambda_2|\lambda_1}$ is a judging function

$$d_{\lambda_2|\lambda_1}(\mathbf{O}) = \begin{cases} 1 & P(\mathbf{O}|\lambda_1) < P(\mathbf{O}|\lambda_2) \\ 0 & \text{otherwise} \end{cases} \quad (13)$$

Obviously we can not obtain the misdetection rate by enumerating all possible \mathbf{O}_i^T for large T directly, since the number of sequences with length T increases exponentially.

Given T and a large number K , our method is to simulate λ_1 and generate K sample sequences with length T using the Monte Carlo method, then estimate the misdetection rate by counting the occurrence of misdetections. The estimator

$$\hat{e}_{\lambda_2|\lambda_1}(T) = \frac{1}{K} \sum_{i=1}^K d_{\lambda_2|\lambda_1}(\mathbf{O}_{\lambda_1,i}^T) \quad (14)$$

where $\mathbf{O}_{\lambda_1,i}^T$ is the i^{th} of total K sample sequences simulated by λ_1 , is an unbiased estimator of $e_{\lambda_2|\lambda_1}(T)$ since

$$E[\hat{e}_{\lambda_2|\lambda_1}(T)] = E\left[\frac{1}{K} \sum_{i=1}^K d_{\lambda_2|\lambda_1}(\mathbf{O}_{\lambda_1,i}^T)\right] = \frac{1}{K} \sum_{i=1}^K E[d_{\lambda_2|\lambda_1}(\mathbf{O}_{\lambda_1,i}^T)] = E[d_{\lambda_2|\lambda_1}(\mathbf{O}_{\lambda_1}^T)] = e_{\lambda_2|\lambda_1}(T) \quad (15)$$

Obtaining $P(\mathbf{O}^T|\lambda_1)$ and $P(\mathbf{O}^T|\lambda_2)$ for each sample sequence causes the major computational complexity while estimating $e_{\lambda_2|\lambda_1}(T)$. The well known forward algorithm gives the solution to calculate $P(\mathbf{O}^T|\lambda)$, given by

$$P(\mathbf{O}^T|\lambda) = P_\lambda(\mathbf{O}^T) = \sum_{i=1}^m P_\lambda(\mathbf{O}^T, S_T = s_i) \quad (16)$$

$$\begin{aligned} P_\lambda(\mathbf{O}^T, S_T = s_i) &= P_\lambda(O_T = o_T | S_T = s_i) P_\lambda(\mathbf{O}^{T-1}, S_T = s_i) \\ &= b_{i,o_T} \sum_{j=1}^m P_\lambda(S_T = s_i | S_{T-1} = s_j) P_\lambda(\mathbf{O}^{T-1}, S_{T-1} = s_j) \\ &= b_{i,o_T} \sum_{j=1}^m a_{ij} P_\lambda(\mathbf{O}^{T-1}, S_{T-1} = s_j) \quad i = 1 \dots m \end{aligned} \quad (17)$$

Knowing that

$$P_\lambda(\mathbf{O}^1, S_1 = s_i) = \pi_i b_{i,o_1} \quad i = 1 \dots m \quad (18)$$

one can get $P(\mathbf{O}^T|\lambda)$ recursively.

Given L as the maximum sequence length to simulate, we need to calculate $\hat{e}_{\lambda_2|\lambda_1}(T)$ for each $T = 1 \dots L$. The straightforward method is: first, one simulates K sample sequences with 1 symbol only, and calculates $\hat{e}_{\lambda_2|\lambda_1}(1)$; then one simulates K sample sequences with 2 symbols independently, and gets $\hat{e}_{\lambda_2|\lambda_1}(2)$, and continuing until reach to the max length L . However, (16) and (17) show that we can use $P_\lambda(\mathbf{O}^{T-1}, S_{T-1} = s_j)$, the intermediate results for solving $P(\mathbf{O}^{T-1}|\lambda)$, to calculate the value of $P_\lambda(\mathbf{O}^T, S_T = s_j)$ and thus $P(\mathbf{O}^T|\lambda)$ easily. A large amount of computation can be saved by taking this advantage.

Assuming that the sample set for length $T-1$ has already been generated as $\{\mathbf{O}_i^{T-1}\}$, we can generate the sample set for length T by adding one observation symbol to each sample in $\{\mathbf{O}_i^{T-1}\}$, so $\mathbf{O}_i^T = \langle \mathbf{O}_i^{T-1}, o_T \rangle$, where o_T is the new symbol

generated according to the Monte Carlo rule at time T . In this way, the intermediate results for \mathbf{O}_i^{T-1} can be utilized for \mathbf{O}_i^T and we can achieve linear complexity on the maximum length L .

Doing so will cause the sample set $\{\mathbf{O}_i^T\}$ to not be independent of $\{\mathbf{O}_i^{T-1}\}$. This consequence, however, does not affect the accuracy of estimator $\hat{e}_{\lambda_2|\lambda_1}(T)$, since all samples in $\{\mathbf{O}_i^T\}$ are still independent of each other.

3.2 Experiments and Discussions

Using the algorithm mentioned above, we choose two different HMMs and plot estimated $e_{\lambda_2|\lambda_1}(T)$ and $e_{\lambda_1|\lambda_2}(T)$ in linear (Fig 5.a) and logarithm (Fig 5.b) scale respectively. Fig 5 is a hardcopy of the distinguishability measurement chart generated by the PQS client. The X axis is the length of the sequence, and the Y axis is the misdetection rate. The result shows that both $e_{\lambda_2|\lambda_1}(T)$ (the lower curve on Fig 5.a and 5.b) and $e_{\lambda_1|\lambda_2}(T)$ converge to zero exponentially as the length of the observation sequence increases. $e_{\lambda_2|\lambda_1}(T)$ converges to zeros faster than $e_{\lambda_1|\lambda_2}(T)$, however. A reasonable explanation for this difference is that $D(\lambda_1, \lambda_2) \neq D(\lambda_2, \lambda_1)$, while the converging speed of $e_{\lambda_2|\lambda_1}(T)$ is determined by $D(\lambda_1, \lambda_2)$, and $e_{\lambda_1|\lambda_2}(T)$ by $D(\lambda_2, \lambda_1)$.

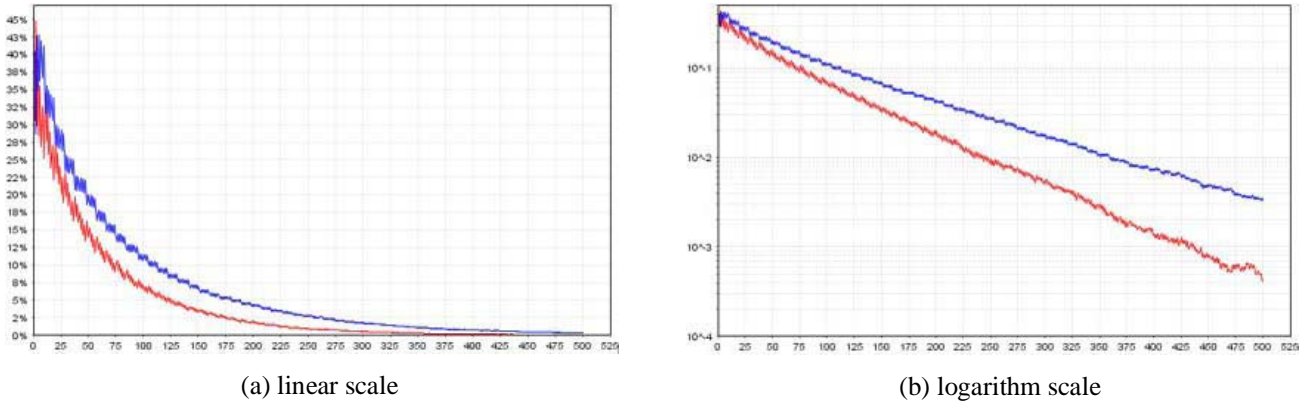


Fig 5 The Comparison of Two HMMs

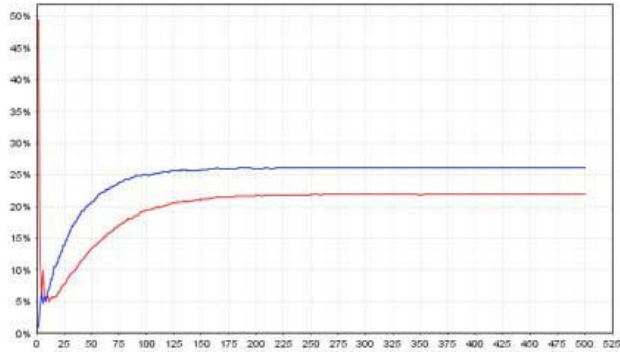


Fig 6 Misdetection Rates Plot for Zero Distance Models

Another experiment is done by choosing two different models with $D(\lambda_1, \lambda_2) = D(\lambda_2, \lambda_1) = 0$. This experiment gives out interesting results that the misdetection rates converge to different nonzero values (Fig 6). This is strong support for the explanation above since the misdetection rate stays constant for long sequences, while the distance between models is zero. This result also shows that the two models are still somehow distinguishable, however, which implies the two models are not equivalent even if the distance is zero. The reason is that the model distance defined in (9) and (10) is effective for stationary processes only, while the initial conditions may affect the overall distinguishability significantly.

IV. CONCLUSIONS AND FUTURE WORK

We have shown the importance of user interfaces for process modeling and detection systems, and discussed the design and implementation of the graphical user interface for PQS extensively. In the future, we are going to keep exploring the innovative concept of process queries, and how to enhance the user interfaces to help PQS achieve its goal. For example, we will support more modeling techniques, observation types, query types, tools and reports, etc.

The query result visualization is still a challenging issue, especially for the hypothesis set. The hypothesis set has complex and multidimensional structure. A good visualization of the hypothesis set in a meaningful way is crucial to help users understand the results and make decisions according to them.

The visual HMM comparison tool generates interesting results. In general, the misdetection rate plot is an effective and intuitive visual measurement of the distinguishability between HMMs. This method provides meaningful results for arbitrary pairs of HMMs in linear time of the maximum simulated sequence length. It is an interesting subject for the future study how stationary model distances and initial conditions affect the overall distinguishability between HMMs.

ACKNOWLEDGEMENTS

The author wishes to thank George Cybenko for his guidance during this work and for the fruitful discussions that resulted in most of the ideas presented in this paper. The author also wishes to thank Robert S. Gray and Diego Hernando and the reviewers for their valuable suggestions.

This research was supported in part by ARDA Grant F30602-03-C-0248, DARPA Projects F30602-00-2-0585, and National Institute of Justice, Department of Justice award number 2000-DT-CX-K001.

Points of view in this document are those of the author and do not necessarily represent the official position of the sponsoring agencies or the U.S. Government.

REFERENCES

- [1] L.R. Rabiner, A Tutorial on Hidden Markov Model and Selected Applications in Speech Recognition, *Proceedings of the IEEE*, Vol 77, No. 2 1989 Feb: p257-286
- [2] J. L. Peterson. Petri Net Theory and the Modeling of Systems. *Prentice-Hall International*, 1981
- [3] R.E. Kalman, A New Approach to Linear Filtering and Prediction Problems, *Transactions of the ASME—Journal of Basic Engineering*, 82 (Series D), 1960: p35-45
- [4] S.J. Young, *The HTK Hidden Markov Model Toolkit: Design and Philosophy*, Cambridge University Engineering Department, UK, 1993.
- [5] Krogh, A. An introduction to hidden Markov models for biological sequences. *Computational Methods in Molecular Biology*, 1998, Elsevier. p45–63.
- [6] A. Schliep, A. Schönhuth, C. Steinhoff. Using Hidden Markov Models to Analyze Gene Expression Time Course Data. *Bioinformatics*. 2003 Jul;19 Suppl 1:p255-263
- [7] Y. Ephraim and N. Merhav, Hidden Markov Processes, *IEEE Transactions on Information Theory*, Vol. 48, No. 6, 2002 Jun: p257-286
- [8] Dan Connolly et al, DAML+OIL Reference Description, *W3C Note* 2001 Dec, <http://www.w3.org/TR/daml+oil-reference>
- [9] T.M. Cover, J.A. Thomas, *Elements of Information Theory*, John Wiley & Sons, Inc. 1991
- [10] B.-H. Juang and L. Rabiner. A probabilistic distance measure for hidden Markov models. *AT&T Technical Journal*, 64(2):p391--408, 1985
- [11] V. Berk, W. Chung, et al., Process Query Systems for Surveillance and Awareness, in *7th World Conference on Systemics, Cybernetics on Informatics (SCI2003)*, July 2003