RC23140 Declassified 11/16/2004 (W0403-060) March 8, 2004
Computer Science

IBM Research Report

Similarity-Based Alignment and Generalization: A New Paradigm for Programming by Demonstration

Daniel Oblinger, Vittorio Castelli, Tessa Lau, Lawrence D. Bergman
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Similarity-Based Alignment and Generalization: A New Paradigm for Programming By Demonstration

Daniel Oblinger OBLIO@US.IBM.COM
Vittorio Castelli VITTORIO@US.IBM.COM
Tessa Lau TESSALAU@US.IBM.COM
Lawrence D. Bergman BERGMANL@US.IBM.COM
IBM T.J. Watson Research, New York

Abstract

We present an approach to learning procedural knowledge by demonstration called *similarity-based alignment and generalization*. Key to our approach is the ability to induce complex procedure structure (loops and conditional branches) by aligning multiple unannotated demonstrations of a procedure. We present an implemented instance of a similarity-based alignment and generalization algorithm that relies on the known Input-Output Hidden Markov Models, and describe an extension, the *SimIOHMM*, that significantly improves the algorithm's performance. We present an empirical evaluation that demonstrates our system's scaling performance and quantifies the performance increase obtained through the use of the SimIOHMM extension.

1. Introduction

Knowledge-based organizations expend significant resources on capturing, disseminating, and reusing procedural knowledge. Procedural knowledge acquisition techniques, such as programming by demonstration (PBD) (Cypher, 1993b; Lieberman, 2001), hold great promise for such organizations. With PBD, one or more experts demonstrate the procedure and the system learns a model that can be used to execute that same procedure again on a different system. The key feature of PBD is automatically generalizing from one or more concrete demonstrations to an abstract model of the procedure.

In PBD systems, a demonstration is typically a recorded sequence of user actions and changes to the Graphical User Interface (GUI). One or more of these demonstrations is used to induce a model of the underlying procedure. For procedures executed on a GUI, a step typically represents a single user action, such as clicking on a button. Previous PBD systems work well when there is a fixed number of steps in the procedure (Lau et al., 2000) or when the procedure author can identify the specific step to be generalized (Maulsby & Witten, 1997). However, these assumptions are violated when a procedure grows to a large number of steps and contains complex structure such as conditional branches.

For example, consider the task of configuring an e-mail client for the employees of an organization. The exact sequence of actions taken may vary depending on the user's operating system configuration and on the configuration of the mail client. A procedure demonstrated in a particular environment may contain conditional steps that have no analogue in other demonstrations. Thus, a PBD system cannot simply assume that the ith step in one demonstration corresponds to the ith step in all the other demonstrations. One possible approach to this problem is to use action similarity to identify similar steps in the procedure (Mo, 1989). This has limited applicability, however. One may need to click the same button at several different points in a given procedure, yet it would be a mistake to simply map all of these actions to the same point in the induced procedure based only on that surface similarity.

For the purposes of this paper, we define *program-*

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

ming by demonstration as the problem of taking a set of demonstrations and generating a procedure model consistent with those demonstrations. Each demonstration is a sequence of events, including user actions as well as changes to the applications' GUI. A procedure model is consistent with a demonstration if it correctly predicts the actions in the sequence given the prior events in that sequence. This paper presents an approach to the problem of programming by demonstration inspired by sequence alignment algorithms. Specifically, in this paper:

- We apply traditional sequence alignment algorithms to the problem of programming by demonstration to learn procedures with complex structure.
- We define the *SimAlignGen* class of algorithms. This class extends traditional Hidden Markov Models by adding an additional source of bias: a similarity function over the inputs.
- We present an instance of an *SimAlignGen* class, called SimIOHMM, which has been implemented as part of a programming by demonstration system on the Windows platform.
- We provide an empirical evaluation of SimIOHMM's ability to learn a real-world procedure from demonstrations, and show that the addition of the similarity function results in a significant performance improvement.

2. PBD architecture

Before formally describing the learning algorithm underlying *SimAlignGen*, we will briefly discuss the main components and processes in our approach to PBD. Figure 1 provides a block diagram of our PBD system.

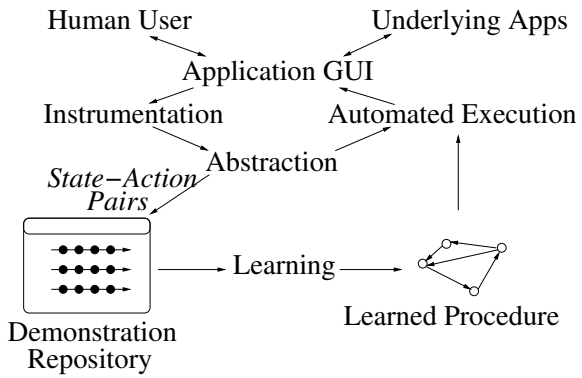

Figure 1. Block diagram of the *SimAlignGen* approach.

The complete data flow includes: recording procedure demonstrations, learning a procedure model from multiple demonstrations, and automatically executing steps of the induced procedure.

A human user demonstrates procedures by performing actions, such as clicking the mouse or pressing keyboard keys, on an application's graphical user interface. In response, the system performs actions, such as creating or deleting windows. The instrumentation component captures both user and system actions.

An abstraction component converts the stream of events recorded by the instrumentation into a sequence of *state-action pairs*, called a *trace*. Logically, a state-action pair represents the complete state of the GUI at a point in time, coupled with the action the user took in that state. This state-action representation is natural for use with inductive learning since it reduces the problem of learning a procedure to the problem of predicting the user action from the state of the GUI (and perhaps its history). The abstracted representation differs from the original event representation in four ways. First, the widget hierarchy is flattened in a similar manner to flattening of the DOM by XPath expressions (XML Path Language, 1999). Second, using heuristics, a subset of window features (such as the window title) is selected from the full set available. Third, the incremental changes to the GUI reflected in the event stream are converted into snapshots of the system state. Fourth, low-level events, such as "mouse down" are heuristically grouped into higher-level actions, such as "double click".

Abstracted demonstrations are stored in a demonstration repository and then provided as input to the learning component, described in detail in the next section. The executable procedure induced by this component can be thought of as a graph representing the procedure underlying the demonstrated traces, along with the decision support necessary to determine appropriate graph transitions and generate actions for each procedure step based on the current state of the GUI.

The induced procedure is used to drive automatic execution at playback. The playback component accepts a sequence of GUI states from the current execution environment (supplied by the instrumentation/abstraction components) and previously executed actions, and attempts to predict the next action in that sequence. Once this action is predicted it can be executed (at the user's discretion) on the GUI.

3. Similarity-Based Alignment and Generalization

The learning component takes as input a set of demonstrations and aligns them to produce an executable

procedure model.

Formally, the *alignment* of a set of demonstration traces is a partition of the state-action pairs in all the traces. A useful alignment for our purposes is one that groups together similar state-action pairs, such that each set of the partition corresponds to what a human would think of as a step in the procedure model.

The *SimAlignGen* approach induces a procedure model consistent with demonstrated training sequences by simultaneously employing three sources of bias in its search for this model.

1. The alignment of the steps in the demonstrations should preserve transitions between successive steps. For example, let demonstration 1 consist of step A followed by step B and demonstration 2 consist of step A' followed by B'; then aligning A with A' and B with B' is a good alignment, since it preserves the ordering of transitions within the demonstrations.
2. The alignment of state-action pairs should also yield sets that can be *generalized*—within a partition set, actions should be predictable from their corresponding states by an appropriately induced mapping function.
3. The states in aligned state-action pairs should be *similar* according to some domain-specific similarity metric.

Constructing a learner with the first two biases—transition preservation and generalization—is a difficult problem for which no optimal algorithm exists. One solution consists of iteratively alternating two steps: finding the best alignment of the training data consistent with a given transition and generalization structure, and finding the best transition and generalization structure consistent with a given alignment of the training data. If we represent the alignment by associating an integer with each state-action pair (the index of the partition set to which it belongs) or, more generally, a probability distribution over the partition sets, we can immediately reduce the iterative algorithm to the Baum-Welch algorithm (Baum et al., 1970). Baum-Welch is an expectation-maximization (E-M) algorithm used to induce discrete Hidden Markov Models from sequences of symbols (Rabiner & Juang, 1986). We interpret the expectation step as a way of determining the best alignment of the sequences relative to a given Markov model, and the maximization step as a way of inducing the best procedure model given an alignment.

For use in PBD, however, it is not sufficient to summarize the user's actions as a probability distribution over the space of actions as a traditional HMM would do. We cannot simply learn that two-thirds of users press the "Add" button and the remaining users press "Remove" at some particular procedure step; we must learn a function that predicts when each is appropriate. In our approach, we assume this choice may depend on features observed in the GUI, that is, both the next node and the next action conditionally depend on the current state of the GUI given the current node. Frasconi and Bengio (Frasconi & Bengio, 1994) introduced an extension to HMMs, called the Input-Output Hidden Markov Model, or IOHMM, that satisfies this assumption. IOHMMs predict the next node and the next output symbol as a function of the current node and of the current input symbol. In the PBD context the input symbol is the state, and the output symbol is the action in a state-action pair.

The third bias, similarity of aligned steps, is not employed by HMM or IOHMM algorithms, yet it is a natural form of knowledge in the PBD domain. This can be clearly seen when considering a procedure that involves actions taken on the Microsoft Window's control panel, as well as actions on a browser window. The control panel states and associated actions are clearly distinct from those in the browser; an induced procedure model that separates state-action pairs based on this surface similarity will tend to be better than one that does not. We capture this domain knowledge as a similarity bias in the form of a similarity metric over state-action pairs.

We have developed an algorithm, SimIOHMM, that extends the IOHMM by incorporating this similarity bias. SimIOHMM extends the Baum-Welch algorithm to incorporate a similarity metric over the space of state-action pairs. This additional bias is (1) readily available in the PBD context, (2) cannot be expressed within either the HMM or IOHMM frameworks, and (3) has a large impact on reducing training time as demonstrated in Section 5.

4. The SimIOHMM

In this section we first recall the Baum-Welch algorithm for classical HMMs, then we describe how to modify it for the SimIOHMM, to incorporate the third sources of bias. The following notational conventions are used throughout. Plain-text upper case characters denote random variables, while plain-text lower case characters denote values. The training set contains $\mathcal{M}$ traces. Let O_t be a shorthand notation for the state-action pair (S_t, A_t) observed at time t; the state S_t is the IOHMM input and the action A_t is the output. To specify the trace r to which an observation

belongs we use a superscript in parenthesis, as in $O^{(r)}$ (for sake of clarity, we will use this notation explicitly only when unclear from the context). We wish to stress that in this paper the term *state* is a description of the state of the application GUI used as input to the IOHMM, not the state of the HMM, for which we use the term *node*. Let O_a^b be the sequence of observations between time a and b. Let Q_t be the HMM node at time t, taking values in $[1, \mathcal{N}]$. Let $\hat{\theta}$ describe the set of parameters of the model obtained during the previous M-step (these parameters are explicit if the selected model is parametric and implicit otherwise. In our implementation, described later, these parameters are the internal variables of the action and transition classifiers). The length of a training sequence is denoted by $\mathcal{T}$. Using the standard notation for HMMs we define:

$\pi_n = P_{\hat{\theta}}\Big(Q_1 = n\Big)$
is the initial probability distribution over the nodes,

$b_n(o_t) = P_{\hat{\theta}}\Big(O_t = o_t \mid Q_t = n\Big)$
is the conditional probability of observing o_t given that the node is n,

$\alpha_n(t) = P_{\hat{\theta}}\Big(O_1^t = o_1^t, Q_t = n\Big)$
is the probability of observing o_1^t as the first t observations, and being in state n at time t,

$\beta_n(t) = P_{\hat{\theta}}\Big(O_{t+1}^{\mathcal{T}} = o_{t+1}^{\mathcal{T}} \mid Q_t = n\Big)$
is the conditional probability of observing $o_{t+1}^{\mathcal{T}}$ as the last $\mathcal{T} - t$ observations, given that the node at time t is n,

$\gamma_n(t) = P_{\hat{\theta}}\Big(Q_t = n \mid O_1^{\mathcal{T}} = o_1^{\mathcal{T}}\Big)$
is the conditional probability of being in node n at time t given that the entire sequence of observations is $o_1^{\mathcal{T}}$. This is the alignment.

$\xi_{i,j}(t) = P_{\hat{\theta}}\Big(Q_t = i, Q_{t+1} = j | O_1^{\mathcal{T}} = o_1^{\mathcal{T}}\Big)$
is the conditional probability that the node at time t is i and that the node at time $t+1$ is j given that the entire sequence of observations is $o_1^{\mathcal{T}}$.

$a_{i,j}(t+1) = P_{\hat{\theta}}\Big(Q_{t+1} = j \mid Q_t = i, O_{t+1} = o_{t+1}\Big)$
is the transition probability, more specifically, the conditional probability of being in node j at time $t+1$ given that the node at time t is j and that the observation at time $t+1$ is o_{t+1}.

4.1. Baum-Welch algorithm for HMMs

The Baum-Welch algorithm alternates the E-step and the M-step until convergence.

The **E-step** consists of the backward-forward procedure, followed by the computation of the alignment of the demonstrations with the HMM graph. The forward procedure is governed by the equations

$\alpha_i(0) = \pi_i$
$\alpha_n(t+1) = b_n(O_{t+1}) \sum_{j=1}^{\mathcal{T}} \alpha_j(t) a_{j,n}(t+1)$,
where we assume that there is a unique starting node. The backward procedure follows the equations

$$\begin{aligned} \beta_i(\mathcal{T}) &= 1 \text{ for terminal nodes} \\ &= 0 \text{ otherwise} \\ \beta_i(t) &= \sum_{n=1}^{\mathcal{N}} \beta_n(t+1) a_{i,n}(t+1) b_n(O_{t+1}). \end{aligned}$$

Finally, the alignment is described by the set of probability distributions over the nodes, recomputed in each iteration as

$$\left\{ \gamma_n^{'}(t) = \frac{\alpha_n(t)\,\beta_n(t)}{\sum_{j=1}^{\mathcal{N}} \alpha_j(t)\,\beta_j(t)} \right\}_{t=1}^{\mathcal{T}},$$

where the quantities on the right hand side come from the previous iteration, and the transition probabilities, recomputed as

$$\left\{ a_{i,j}^{'} = \frac{\sum_{t=1}^{\mathcal{T}-1} \xi_{ij}^{'}(t)}{\sum_{t=1} \mathcal{T} - 1 \gamma_i^{'}(t)} \right\}_{i,j=1}^{\mathcal{N}}, \quad \text{where}$$

$$\xi_{i,j}^{'}(t) = \frac{\gamma_i(t)\, a_{i,j}(t+1)\, b_j(O_{t+1})\, \beta_j(t+1)}{\beta_i(t)}.$$

The **M-step** consists of computing the initial probability distribution and node transition probabilities that maximize the likelihood of the data given the alignment computed in the E-step.

4.2. The SimIOHMM

In a traditional HMM, o_t consist of just the output A_t, and the term $b_n(o_t)$ equals $P_{\hat{\theta}}\Big(A_t = a_t \mid Q_t = n\Big)$ In IOHMMs, where o_t contains both state and action, $b_n(o_t)$ is replaced by $P_{\hat{\theta}}\Big(A_t = a_t \mid S_t = s_t, Q_t = n\Big)$, and hence the algorithm models the *conditional* distribution of the possible actions (output) given the current state (input) and node. SimIOHMMs, on the contrary, estimate $b_n(o_t)$, and therefore models their *joint* distribution.

The **E-step** of a SimIOHMM learning algorithm differs from those of the HMM and IOHMM in the estimation of $b_n(o_t)$. Using Bayes rule we can write

$$P_{\hat{\theta}}\Big(S_t = s_t \,|\, Q_t = n\Big) = P_{\hat{\theta}}\Big(Q_t = n \,|\, S_t = s_t\Big) \frac{P_{\hat{\theta}}\Big(S_t = s_t\Big)}{P_{\hat{\theta}}\Big(Q_t = n\Big)}.$$

There are three terms on the right hand side:

$P_{\hat{\theta}}\Big(Q_t = n\Big)$ is the unconditional probability that the node at time t is node n and can be estimated as $P_{\hat{\theta}}\Big(Q_t = n\Big) = \frac{1}{M} \sum_{i=1}^{M} \gamma_n(t)$.

$P_{\hat{\theta}}\Big(S_t = s_t\Big)$ is the probability that the state at time t is s_t and can be estimated in two ways:

- as a constant, which then becomes immaterial in the derivation because of the normalization of the probabilities;
- as the fraction of training sequences where the state at time t is equal to s_t, namely, as

$$P_{\hat{\theta}}\Big(S_t = s_t\Big) = \frac{1}{\mathcal{M}} \sum_{r=1}^{\mathcal{M}} \delta\left(S_t^{(r)}, s_t\right),$$

where we have used the Dirac delta notation.

$P_{\hat{\theta}}\Big(Q_t = n \mid S_t = s_t\Big)$ is the probability that the node at time t is node n given the state s_t, and its estimation is the main topic of this section.

Among the parameters $\hat{\theta}$ induced during the M-step, described later, are the representative states $\{r_k\}_{k=1}^{M}$ associated with the $\mathcal{N}$ nodes of the HMM. Let $d(\cdot,\cdot)$ be a distance function defined over the set of states, and let $K(\cdot)$ be a one-dimensional kernel function, such as a Gaussian density function. We use the Nadaraya-Watson kernel density estimation (Hastie et al., 2000, Ch. 5) to convert distances into conditional probabilities:

$$P_{\hat{\theta}}\Big(Q_t = n \mid S_t = s_t\Big) = \frac{K\Big(d(s_t, r_n)\Big)}{\sum_{i=1}^{\mathcal{N}} K\Big(d(s_t, r_i)\Big)}. \quad (1)$$

The **M-Step** for SimIOHMMs consists of building for each node a transition model, an action model, and a representative state using the alignment produced by the E-step. The IOHMM, in contrast, only requires the construction of the transition and action models. Currently, each node of the HMM has a traditional classifier for transition and one for actions. Classifiers compatible with the SimIOHMM learning algorithm must have two characteristics: they must allow weighted inputs and produce a probability distribution on the outputs. These classifiers roughly maximize the class-label posterior distribution given a weighted training set. This in general ensures that the objective function maximized by the E-M algorithm does not decrease from one iteration to the next.

For each trace r, the E-step computes the alignments $\gamma_n^{(r)}(t)$ and the transition probabilities $a_{i,j}^{(r)}(t+1)$. The training set for the action classifier of node n consists of all state-action pairs $O_t^{(r)}$ having $\gamma_n^{(r)}(t) \geq \epsilon/\mathcal{N}$, where $\mathcal{N}$ is the number of nodes in the HMM and ϵ is a value smaller than 1.[1] In practice, we found that values of ϵ between 0.3 and 0.001 yield similar desirable results. Each sample is assigned a weight proportional to the corresponding value of γ_n, where the proportionality constant is a large number chosen to match the way in which weighted inputs are managed by the classifier. The same samples are also used to construct the transition classifier. Here, the "class label" is the index of the next node, and therefore each observation can appear multiple times in the training set with different labels. More specifically, the training sample $(O_{t+1}^{(r)}, j)$ appears in the training set for the transition classifier of node n if $\gamma_n^{(r)}(t) \geq \epsilon/\mathcal{N}$, and $a_{n,j}(t+1) > 0$.

During the M-step the representative samples of the nodes are also updated. For each node n all the training traces are analyzed, and the state-action pair O^* is selected, that has the largest probability $\gamma_n(t)$ of being aligned with node n. After possible ties are broken randomly, O^* becomes the new representative state for node n.

[1] Values greater than 1 could result in the degenerate condition where all data is removed from the training set.

5. Experimental Results

We evaluate our SimIOHMM algorithm in two ways: via an empirical assessment of the system's scaling performance on a PBD problem as the amount of training data increases, and through an analysis of the improvement in training time over IOHMMs.

5.1. Materials and methods

The data for the experiments consists of eleven traces of one expert performing a procedure (shown in Figure 2) for modifying and verifying a system DNS networking configuration, described in detail in a web document taken from a corporate helpdesk website. The eleven traces were each recorded on a system with a different initial configuration; thus, each trace follows a different path through the procedure graph. In some cases, extraneous DNS servers had to be removed; in some, the correct server had to be added, and so on. Each trace is composed of 15 to 38 demonstrated actions, the average being just above 22. The SimIOHMM used in the experiments is been implemented in Java using the J48 classifier (an implementation of C4.5) from the Weka library (Witten & Frank, 1999). States are represented as vectors of categorical features. As a distance function we use the Hamming distance between pairs of states, and consider only the following attributes: the application owning the window with focus, and the title of the window with focus. The kernel density estimator used to convert distances into probabilities relies on the quadratic kernel function: $f(x) = (1-x)^2$ for $|x| < 1$, $f(x) = 0$ otherwise. The width of the kernel is set to 1.5, which implies that a state-action pair can be aligned with a node only if at least one of the two attributes used in the distance function matches

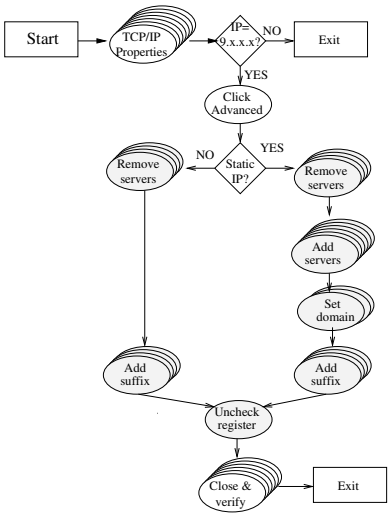

Figure 2. Network configuration procedure. Ovals denote user actions; stacked ovals represent sequences of repeated actions; shaded ovals denote actions that are executed only for certain initial configurations.

the corresponding attribute of the representative state of the node. All the HMMs constructed in the experiments have 25 nodes, the value that optimizes the training time of the IOHMM while maintaining zero classification error. The parameter ϵ used to select the training sets for the action and transition classifiers are 0.2, but the algorithm is somewhat insensitive to the value of this parameter.

The experiments were run on a machine with two 2.4 GHz Xeon processors, with 4 GB of main memory, running under the Linux operating system.

5.2. Experiment 1 – *SimAlignGen* efficacy

We evaluate the performance of our system by randomly dividing the corpus into a training set and a test set, and study how the accuracy of the algorithm increases as the size of the training set increases. For each split, we train our system on the training set and we compute the accuracy as the ratio of the number of correctly predicted actions in all test traces over the total number of actions in those traces. More specifically, to compute the prediction accuracy on a specific trace, we present the next state to the SimIOHMM, obtain a prediction of the next action, compare it with the action taken by the expert, and notify the SimIOHMM of the action taken by the expert. For each training set size, we perform 10 independent repetitions. The initial alignments of the SimIOHMM are computed by a similarity-based algorithm not described in this paper.

Figure 3 shows the results of this experiment. For each split, the prediction accuracy is represented by a "$*$" symbol. For each training set size, the average accuracy over the 10 repetitions is also plotted using the "$\circ$" symbol, and these averages are connected by a line.

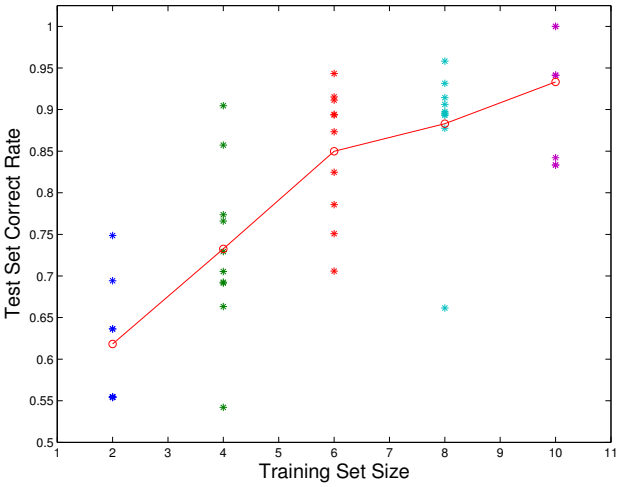

Figure 3. Correct prediction rates on test set as a function of the training set size. For each training set size, 10 independent repetitions are plotted. The averages of the 10 repetitions are plotted connected by a solid line.

The figure shows that accuracy is an increasing function of the number of traces in the training set. This is not surprising, since the coverage of steps in the testing set increases with the training set size. However, of the eleven traces, only two are completely predictable given the remaining 10. Hence, only when 10 traces are used for training (i.e., when leave-one-out cross-validation is used) do we observe 100% accuracy. The relatively high accuracy observed with small training sets indicates that a sizable fraction of the actions in the procedure are common to all procedures, such as the clicks required to open particular control panel dialogs.

We note that there are often outliers, namely, cases where the average correct prediction rate is unusually low (evident, for example, when the training set size is 4 or 8). This is due to the fact that the SimIOHMM becomes confused when presented with an expert action that was not observed in the training set: here, the SimIOHMM produces a warning notification that the future predictions are generally unreliable. If a previously unseen action occurs early in a test trace, the error rate for that trace is usually very high.

5.3. Experiment 2: SimIOHMM training time

A benefit of SimIOHMM over IOHMMs is a substantial reduction in training time and better scalability as a function of training set size. Figure 4 shows the average training time as a function of the number of training traces. Each point is the average of

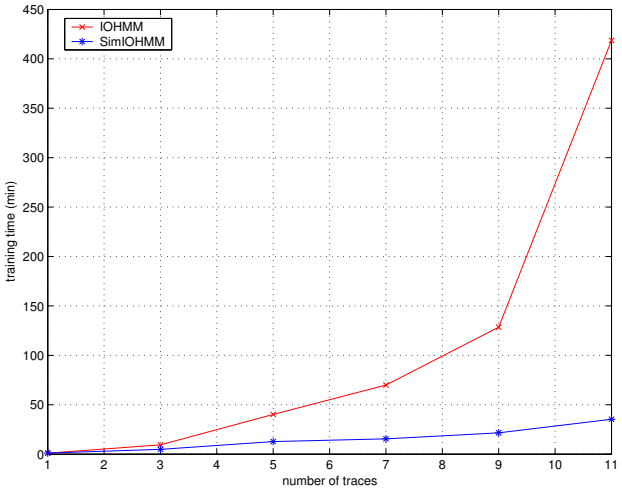

Figure 4. Training time for IOHMM and SimIOHMM as a function of the number of training traces.

10 independent repetitions; the variance was too small to be worth plotting. Here, the initial alignment of both SimIOHMM and IOHMM are computed using the same randomized algorithm.

In the figure, it is apparent that the from the viewpoint of training time, the SimIOHMM scales much better than the IOHMM, and that the ratio of the IOHMM training time to the SimIOHMM training time is superlinear in the number of training traces.

The faster training time of the SimIOHMM is due to the fact that the training sets used to train the node classifiers tend to be smaller. The main reason is that a state-action pair can be aligned only with nodes having similar representative samples. A way of measuring this effect is by analyzing the dispersion of the alignment distributions $\gamma_n(t)$ for the training traces once convergence is reached. A measure of dispersion of a probability distribution is its entropy. Figure 5 shows the average entropy (in bits) of the alignment at convergence as a function of the number of training traces. The experiments are the same used for Figure 4. Due to the similarity bias, the SimIOHMM yields substantially more concentrated alignment probabilities than the IOHMM, and the difference between these averages entropies is an increasing function of the number of traces. As the number of traces required to learn a procedure grows, inaccuracies in the induced model stem from both incomplete training datasets and misalignments during training. We anticipate that the more concentrated alignment probabilities produced by the SimIOHMM will reduce these misalignments and thus improve predictive accuracy as well as training performance.

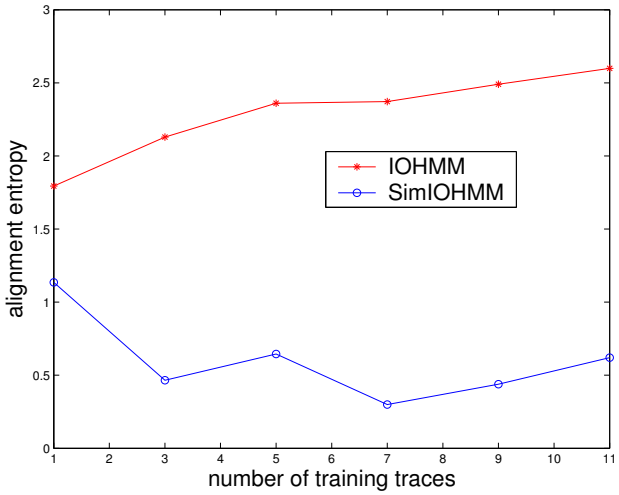

Figure 5. Alignment entropy (in bits) for IOHMM and SimIOHMM as a function of the number of training traces.

6. Related work

Fasconi and Bengio (Frasconi & Bengio, 1994) introduced the concept of IOHMM. Our work differs from this and later work on IOHMMs by introducing the use of a similarity metric. Sequence alignment algorithms have been widely developed and applied to spatial sequences in domains such as computational biology (Krogh et al., 1993), and temporal sequences in domains such as speech recognition. (Rabiner, 1989)

Our work differs from prior work in the field of programming by demonstration primarily in the way that alignment information is obtained. Approaches such as SMARTedit (Lau et al., 2000) and Eager (Cypher, 1993a) implicitly align demonstrated steps using position within the sequence itself. In these systems, the procedure is assumed to consist of one or more iterations of the same fixed-length loop body. Thus, the alignment can be trivially determined using each step's position within the sequence. A later approach (Lau et al., 2003) uses version space algebra to find an alignment when the length of the loop body is not known.

However, none of these approaches are able to learn procedures with conditionally performed steps.

7. Conclusions and future work

Programming by demonstration promises to be an effective method for acquiring procedural knowledge. A challenge in this field is learning complex procedures that include conditional branches, in which the demonstrations contain different steps depending on the paths followed. This paper presents an approach to this problem based on the idea of similarity-based alignment and generalization, and makes the following contributions:

- A novel approach to programming by demonstration based on similarity-based alignment and generalization;
- The *SimAlignGen* class of algorithms that extend traditional sequence alignment algorithms by the addition of a third bias based on a *similarity metric*;
- An instance of an *SimAlignGen* algorithm, called SimIOHMM, which has been implemented as part of a programming by demonstration system on the Windows platform; and
- An empirical evaluation of SimIOHMM's performance which demonstrated its effectiveness on traces gathered from a real-world procedure, and results showing significant speedup in training time due to the addition of the similarity metric.

Future work includes the exploration of properties of the algorithm itself, including noise tolerance, scaling of accuracy and of training time, etc. We are investigating the sensitivity of the algorithm to the choice of distance function, of kernel, and of kernel width. We are extending the SimIOHMM in several directions by constructing algorithms for initialization, automatic selection of the number of nodes, and adaptive node-dependent feature selections for the classifiers. We are also investigating new types of learning algorithms that would replace the current statistical action classifiers.
Finally, we believe that this similarity knowledge is not unique to PBD, but may serve as a useful bias in many other domains.

References

Baum, L. E., Petrie, T., Soules, G., & Weiss, N. (1970). A maximization technique occuring in the statstical analysis of probabilistic functions of Markov chains. *Annals of Math. Statistics*, *41*, 164–171.

Cypher, A. (1993a). Eager: Programming repetitive tasks by demonstration. In A. Cypher (Ed.), *Watch What I Do: Programming by Demonstration*, 205–217. Cambridge, MA: MIT Press.

Cypher, A. (Ed.). (1993b). *Watch what I do: Programming by demonstration*. Cambridge, MA: MIT Press.

Frasconi, P., & Bengio, Y. (1994). An EM approach to grammatical inference: Input/Output HMMs. *Proc. IEEE Int. Conf. Pattern Recognition, ICPR '94* (pp. 289–294). Jerusalem.

Hastie, T., Tibshirani, R., & Friedman, J. (2000). *The elements of statistical learning*. Springer Series in Statistics.

Krogh, A., Brown, M., Mian, I. S., Sjolander, K., & Haussler, D. (1993). *Hidden Markov Models in computational biology: applications to protein modeling* (Technical Report UCSC-CRL-93-32).

Lau, T., Domingos, P., & Weld, D. S. (2000). Version space algebra and its application to programming by demonstration. *Proc. Sevententh Int. Conf. on Machine Learning* (pp. 527–534).

Lau, T., Domingos, P., & Weld, D. S. (2003). Learning programs from traces using version space algebra. *Proc. 2nd Int. Conf. on Knowledge Capture*.

Lieberman, H. (Ed.). (2001). *Your wish is my command: Giving users the power to instruct their software*. Morgan Kaufmann.

Maulsby, D., & Witten, I. H. (1997). Cima: an interactive concept learning system for end-user applications. *Applied Artificial Intelligence*, *11*, 653–671.

Mo, D. H. (1989). Learning Text Editing Procedures from Examples. Master's thesis, Univ. of Calgary.

Rabiner, L. (1989). A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proc. IEEE*, *77*, 257–286.

Rabiner, L. R., & Juang, B. H. (1986). An introduction to Hidden Markov Models. *IEEE ASSP Magazine*, 4–15.

Witten, I. H., & Frank, E. (1999). *Data mining: Practical machine learning tools and techniques with Java implementations*. Morgan Kaufmann.

XML Path Language (1999). http://www.w3.org/tr/xpath.