Similarity-Based Alignment and Generalization

Daniel Oblinger, Vittorio Castelli, Tessa Lau, and Lawrence D. Bergman

IBM T.J. Watson Research, New York
oblio,vittorio,tessalau,bergmanl@us.ibm.com

Abstract. We present a novel approach to learning predictive sequential models, called *similarity-based alignment and generalization*, which incorporates in the induction process a specific form of domain knowledge derived from a similarity metric of the points in the input space. When applied to Hidden Markov Models, our framework yields a new class of learning algorithms called *SimAlignGen*.
We discuss the application of our approach to the problem of programming by demonstration–the problem of learning a procedural model of a user's behavior by observing the interaction an application GUI.
We describe in detail the SimIOHMM, a specific instance of *SimAlignGen* that extends the known Input-Output Hidden Markov Model (IOHMM). We use the SimIOHMM in empirical evaluations that demonstrates the dependence of the prediction accuracy on the introduced similarity bias, as well as the computational gains over the IOHMM.

1 Introduction

Many domains require building predictive models from multiple observed data sequences. Examples from the biological domain include protein and DNA sequence alignment or prediction. Many domains with a temporal dimension involve building predictive models from sequential data, examples in the financial domain include market performance prediction, and risk analysis. In the computer networking domain, models of network performance or detection of illegal intrusions have also been learned from observed data sequences. Learning approaches for these domains (like Hidden Markov Model learning [1]) rely primarily on the sequence data itself for the learning and utilize little (if any) additional domain knowledge. In this paper we investigate a particular form of domain knowledge that we call similarity knowledge. We show how this knowledge can be employed in learning predictive models from sequential data, and empirically measure the impact of utilizing this additional source of knowledge.

A second thrust of this paper is to present a novel approach for using sequence modeling techniques like HMM learning for the problem of programming by demonstration [2, 3] described in the second section. In this application of sequence learning we will show that similarity knowledge is both available in the PBD domain, and that its use improves learning performance.

Contributions of this paper include:

- We present a novel application of traditional sequence alignment algorithms to the problem of programming by demonstration in order 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 Microsoft 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 An HMM approach to PBD

For the purposes of this paper, we define *programming 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 and changes to the application GUIs. A procedure model is consistent with a demonstration if it correctly predicts the actions in the sequence given the prior events in that sequence. Existing PBD systems work well when there is a fixed number of steps in the procedure [4] or when the procedure author can identify the specific step to be generalized [5]. However, these assumptions are violated when a procedure grows to a large number of steps and contains complex structure, such as conditional branches. Traditional sequence learning algorithms like HMMs seem appropriate in these cases since they are focus on the problem of identifying optimal alignment of sequences where no simple one-to-one fixed alignment is known a priori.

In this section we briefly outline the primary components necessary for applying these sequence alignment algorithms in the PBD context. Figure 1 provides a block diagram of our PBD system.

According to this Figure 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, that application performs actions, such as creating/deleting windows, or updating their contents. The instrumentation component captures both user and application actions.

An abstraction component converts the stream of events recorded by the instrumentation into a sequence of *snapshot-action pairs*, called a *trace*. Logically, a snapshot-action pair represents the complete content of the GUI at a point in time, coupled with the action the user took in that state. This snapshot-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 content 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

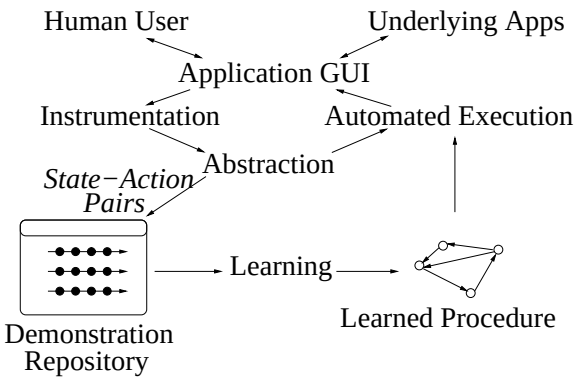

Fig. 1. Block diagram of the *SimAlignGen* approach.

by XPath expressions [6]. 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."

This ability to learn from independently recorded demonstrations is a feature of this approach that differentiates it from existing work in PBD.

3 The *SimAlignGen* family of algorithms

In the previous section we see that the learning component is provided as a set of action sequences just as one would expect for Hidden Markov Model learning, (actually a sequence of snapshot-action pairs as expected of Input/Output Hidden Markov Model learning.) might produce. In the PBD domain, however, the visual cues prevalent on Graphical User Interfaces provide an additional source of knowledge which can be used to augment the sequence data employed by traditional HMM learning. Unlike HMM processes found in nature, the features presented by a typically GUI were *explicitly designed* to illuminate the state of the underlying process generating those features. Further those features were designed to facilitate human mediated processes executed on that GUI. The GUI is designed to visually distinguish importantly different states in the underlying application, and make different steps in a procedure executed on that GUI visually distinct as well. In this paper we consider control panel-based configuration tasks in Microsoft Windows. In that domain for example, distinct steps in a typical configuration procedure will involve GUI windows that have different geometric configurations as well as distinctive title bars. Imagine several demonstrations of a process where the user browses the web to obtain their DNS server's IP address, and then enters that IP address into the appropriate control panel. At

a glance an onlooker could easily distinguish the web browser steps from the control panel steps by matching visual cues from the different demonstrations. We provide this visual similarity knowledge to our family of algorithms as a fixed similarity metric which returns a real-valued score measuring "similarity" between two captured GUI snapshot action pairs (S,A). Formally:

Similarity : $((S \times A) \times (S \times A)) \rightarrow \Re^{+}$.

Thus, instances of the *SimAlignGen* family of algorithms accept an input set of demonstrations as described in the previous section, and a fixed real-valued similarity function over the snapshot-action pairs in that dataset as described in this section. The algorithm then aligns the steps in the demonstrations in order to produce an executable model that can predict or reproduce user actions based on observed states in the GUI.

Formally, the input dataset is composed of a set of demonstrations (called *traces*). Each trace is a sequence of <*GUI-snapshot,User-Action*> pairs called *snapshot-action pairs*. The *alignment* of a set of demonstration traces is a partition of the snapshot-action pairs in all the traces. A useful alignment for our purposes is one that groups together similar snapshot-action pairs, such that each set of the partition corresponds to what a human would think of as a step in the procedure model. *SimAlignGen* algorithms align the snapshot-action pairs in those traces by simultaneously employing three sources of constraint in its search for an alignment. This partitioning of the input data into a set of procedure steps, in turn, drives the generation of an executable model of the underlying procedure being learned. These three sources of constraint are listed below:

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 snapshot-action pairs should also yield sets that can be *generalized*—within a partition set, actions should be predictable from their corresponding snapshots by an appropriately induced mapping function.
3. The snapshots in aligned snapshot-action pairs should be *similar* according to the provided 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 snapshot-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 [7]. Baum-Welch is an expectation-maximization (E-M) algorithm used

to induce discrete Hidden Markov Models from sequences of symbols [1]. We interpret the expectation step as a way of determining the best alignment of the sequences relative to a given HMM, 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 is typically dependant on features observed in the GUI, that is, both the next HMM state and the next action conditionally depend on the current content of the GUI given the current Markov state. Frasconi and Bengio [8] introduced an extension to HMMs, called the Input-Output Hidden Markov Model, or IOHMM, that satisfies this assumption. IOHMMs predict the next state and the next output symbol as a function of the current state and of the current input symbol. In the PBD context, the input symbol is the snapshot of the state of the GUI, and the output is the observed user action. Both are combined as a snapshot-action pair.

The third source of constraint above cannot be employed by either HMM or IOHMM algorithms, yet as we discussed it is a natural form of knowledge in the PBD domain. These algorithms do not take any form of explicit knowledge about the domain, rather they rely entirely on the dataset provided (and possibly parameter settings on the algorithms themselves).

SimAlignGen algorithms extend Hidden Markov Model induction by incorporating this similarity domain knowledge as a bias on the alignments which the algorithm considers in its search for a predictive model of the observed data.

In the next section we detail a specific instance of the *SimAlignGen* family, the SimIOHMM.

In this section we formally define the SimIOHMM by describing how it differs from the standard IOHMM. To this end, we refer to Bengio and Frasconi's paper [9] and follow their notational conventions. As customary, vectors are denoted by boldface characters, boldface symbols with subscripts are elements of a vector, and plain-text symbols denote scalar quantities. Random variables are in upper case, while values in lower case. The training set contains sequences of input-output pairs, or, equivalently, pairs consisting of an input sequence $\mathbf{u}$ and a corresponding output sequence $\mathbf{y}$.

3.1 Bengio and Frasconi's IOHMM

The goal of a IOHMM is to model the *conditional* distribution of $\mathbf{Y}$ given $\mathbf{U}$ as a finite-state process [10, Chapter 1] (also known as a Hidden Markov Model), namely, by postulating the existence of a hidden variable, the state X, taking value in a finite set $\mathcal{X}$, such that

$$\begin{aligned}\mathbb{P}\left[(\mathbf{U}_t, \mathbf{Y}_t, X_t) \mid \{(\mathbf{U}_j, \mathbf{Y}_j, X_j)\}_{j=1}^{t-1}\right] \\ = \mathbb{P}\left[(\mathbf{U}_t, \mathbf{Y}_t, X_t) \mid X_{t-1}\right].\end{aligned}$$

The same structure is inherited by the conditional distribution of the outputs given the inputs modeled by the IOHMM (namely, $\mathbb{P}\left[(\mathbf{Y}_t, X_t) \mid \mathbf{U}_t, \{(\mathbf{U}_j, \mathbf{Y}_j, X_j)\}_{j=1}^{t-1}\right]$) that can now be written as $\mathbb{P}\left[(\mathbf{Y}_t, X_t) \mid \mathbf{U}_t, X_{t-1}\right]$. Note that this probability can be further decomposed as

$$\begin{aligned}&\mathbb{P}\left[X_t \mid \mathbf{U}_t, X_{t-1}\right]\mathbb{P}\left[Y_t \mid X_t, \mathbf{U}_t, X_{t-1}\right]\\ &\quad = \mathbb{P}\left[X_t \mid \mathbf{U}_t, X_{t-1}\right]\mathbb{P}\left[Y_t \mid X_t, \mathbf{U}_t\right],\end{aligned}$$

namely, as a conditional transition probability to state X_t given the previous state and the input at time t, and a conditional probability of the output at time t given the state and input at time t. Hence, inducing a IOHMM is equivalent to estimating the initial probability distribution over the states, $\mathbb{P}\left[X_0\right]$ (note that the first input is observed at $t = 1$), the transition probabilities $\mathbb{P}\left[X_t \mid \mathbf{U}_t, X_{t-1}\right]$, and the conditional output probabilities $\mathbb{P}\left[Y_t \mid X_t, \mathbf{U}_t\right]$. The transition an output probabilities are assumed to be time-independent, namely,
$\mathbb{P}\left[X_t{=}a|\mathbf{U}_t{=}u, X_{t-1}{=}b\right]{=}\mathbb{P}\left[X_s{=}a|\mathbf{U}_s{=}u, X_{s-1}{=}b\right]$ and
$\mathbb{P}\left[Y_t{=}y|X_t{=}x, \mathbf{U}_t{=}u\right]{=}\mathbb{P}\left[Y_s = y|X_s = x, \mathbf{U}_s = u\right]$,
for every time instants s and t. These assumptions make the IOHMM very flexible and yet computationally manageable. It turns out that the IOHMM allows arbitrarily long time-dependence between the input-output pairs, and is therefore more powerful than a fixed-order Markov model. At the same time, the IOHMM can be efficiently induced from training data using the Baum-Welch algorithm, a specialization of the E-M algorithm. The Baum-Welch algorithm consists of two steps: an Expectation step, in which the training sequences are aligned to an existing model, and a Maximization step, in which the model is updated given the alignment.

The IOHMM E-step efficiently computes a probability distribution over the state space for each input-output pair by means of two steps, called forward recursion and backward recursion. The forward recursion computes, for each input-output pair, the joint conditional distributions of the current state, the current output, and all preceding outputs, given the current input and all the preceding inputs. The backward recursion computes, for each input-output pair, the conditional probability of the subsequent outputs given the current state, the current input, and the subsequent inputs. The probability distributions over the states are then obtained by multiplying and normalizing the results of the forward and backward recursions for the corresponding time instant. The results of the forward and backward recursion are also appropriately combined (via multiplication and normalization) to estimate the posterior transition probabilities, namely, the probabilities of transitioning from state i at time t to state j at time $t + 1$ given the entire sequence of inputs and outputs.

The M-step efficiently recomputes the initial probability distribution over the states, the conditional transition probabilities given the current input, and the conditional distributions of the output given the current state and input either by maximum likelihood (in which case the Baum-Welch is a bona-fide E-M algorithm), or by a generic likelihood estimation method (in which case the Baum-Welch is a Generalized E-M algorithm). Bengio and Frasconi followed

the latter approach, and used neural networks in the maximization step. The efficiency of the M-step stems from the fact that the *global* maximization of the likelihood is performed by *separately* maximizing the likelihoods at each individual state, namely, by finding the parameters that maximize the transition probabilities and output probabilities at each state given the results of the expectation step.

3.2 The SimIOHMM

The SimIOHMM extends the IOHMM by further incorporating the bias described earlier in this Section. To this end, each hidden Markov state is associated with one or more representative inputs, as well as with a transition and output distribution. The representative inputs come into play during the E-step and are updated during the M-step, as follows.

The E-Step. The forward recursion is

$$\alpha_{i,t} = \mathbb{P}\left[\mathbf{y}_t \mid X_t = i, \mathbf{u}_t\right] \mathbb{S}\left(\mathbf{u}_t, \mathbf{v}_i\right) \sum_{\ell \in \mathcal{X}} \phi_{i,\ell}(\mathbf{u}_t) \alpha_{\ell,\, t-1}, \tag{1}$$

which adds to [9, Equation(20)] the additional term $\mathbb{S}\left(\mathbf{u}_t, \mathbf{v}_i\right)$, the similarly score between the input $\mathbf{u}_t$ and the representative sample $\mathbf{v}_i$ of state i, that provides the required bias.

Similarity, the backward recursion is

$$\begin{aligned} \beta_{i,t} = \sum_{\ell in \mathcal{X}} \mathbb{S}\left(\mathbf{u}_{t+1}, \mathbf{v}_\ell\right) \mathbb{P}\left[\mathbf{y}_{t+1} \mid X_{t+1} = \ell, \mathbf{u}_{t+1}\right] \cdot \\ \cdot \phi_{\ell,i}(\mathbf{u}_{t+1}) \beta_{\ell,\, t+1}, \end{aligned} \tag{2}$$

which again adds a bias term to [9, Equation(22)].[1]

The bias term $\mathbb{S}\left(\mathbf{u}_t, \mathbf{v}_i\right)$, a similarity score, is designed to concentrate the distribution over the states at time t onto those states having representative samples similar to $\mathbf{u}_t$. For sake of simplicity, assume that each state has a unique representative sample. Then $\mathbb{S}\left(\mathbf{u}_t, \mathbf{v}_i\right)$ is computed as follows: first, the distances $\{d(\mathbf{u}_t, \mathbf{v}_i)\}_{i=1}^{|\mathcal{X}|}$ between $\mathbf{u}_t$ and the representatives of the states are computed; then these distances are converted into a similarity score by means of a kernel $K(\cdot)$ (for example, a finite-support decreasing function or a Gaussian):

$$\mathbb{S}\left(\mathbf{u}_t, \mathbf{v}_i\right) = K\left(d(\mathbf{u}_t, \mathbf{v}_i)\right).$$

The definition of $\mathbb{S}\left(\cdot, \cdot\right)$ can be obviously extended to capture the similarities between input-output pairs, rather than between inputs.

Adding the bias term substantially improves the training time and can improve the classification accuracy, as illustrated in the experiments section.[2]

[1] The small discrepancies with the actual Equation [9, Equation(22)] are due to typographical errors in the original paper.

[2] An attentive reader would notice that Equations (1) and (2) produce scaled probabilities computed in the traditional Baum-Welch algorithm (namely, quantities that

The M-step. The M-step of the SimIOHMM is identical to that of the IOHMM, with the additional recomputation of representative samples. For each HMM state, the sample(s) with highest alignment probability becomes the new representative state.

4 Experimental Results

We evaluate our *SimAlignGen* in two ways. We evaluate the utility of using a similarity bias for learning HMM models in a general setting using synthetic data where we can vary the "correctness" of similarity data presented. We then evaluate the predictive performance of this approach in a practical PBD setting implemented on the Microsoft Windows GUI where we measure the effectiveness of the *SimAlignGen* approach as well as utility of the similarity bias in this context.

4.1 Experiment 1: Effectiveness of similarity knowledge

We measure the performance of *SimAlignGen* relative to traditional IOHMM performance as a function of the correctness of the similarity information presented. We expect the performance of *SimAlignGen* algorithms to vary as a function of how well the similarity knowledge matches the process underlying the generated data. We quantify this degree of match by introducing a *similarity correctness* score given (1) an underlying generating process, (2) a dataset generated from that process where each point in the dataset is labeled by the state that generated it, and (3) a similarity metric that returns a real-valued similarity score for each pair of points in the dataset. The similarity correctness score is a number between 0 and 1 that represents the fraction of dataset points whose nearest neighbor (according to the similarity metric) was also generated by the same state. Thus a score of 100% would imply the similarity metric correctly separates all of the dataset points into groups perfectly aligned with the underlying generating process. Lesser scores are associated with increasing levels of noise, or misinformation regarding the underlying generating process.

For this experiment, we randomly generate Hidden Markov Models by instantiating a generic HMM template. The template has ten hidden states, each state's distribution is composed of two separate randomly generated distributions of features, actions, and transitions. The output actions are a binary feature, and the inputs are a pair of binary features. Because of the sparseness in both the input and output representation, even this relatively modestly sized HMM process is difficult to learn. We add a feature to the generated dataset used in computing the similarity between generated snapshot-action pairs. This feature is the index number for the state generating each data point. We inject a variable amount of noise or misinformation into the feature, perturbing each with additive Gaussian noise with zero mean and some chosen variance.

do not sum to one). This fact, however, is immaterial because even in the traditional Baum-Welch the results of the forward and backward regression are combined through a normalization step, which takes care of this scaling issue.

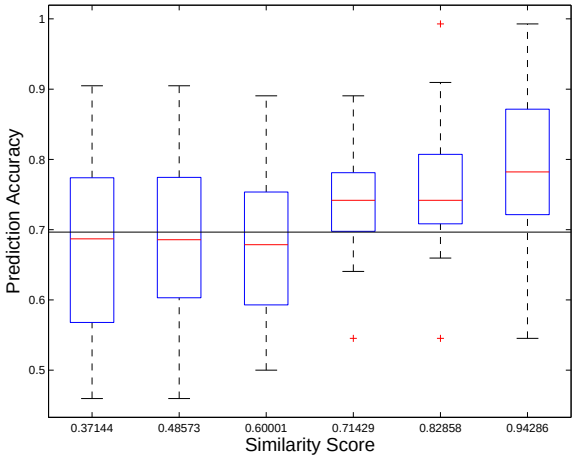

Fig. 2. Accuracy as a function of similarity score

Figure 2 shows the accuracy of SimIOHMM as a function of the correctness of the provided similarity knowledge. The graph is generated from 180 runs of the SimIOHMM obtained from three-fold cross validation of 60 randomly generated HMM models. Each dataset is composed of 400 data-points (20 traces with 20 snapshot-action pairs in each) along with a varying Gaussian noise added to the similarity feature. A similarity correctness score and an accuracy were computed for each run. The resulting points are binned according to similarity score and then averaged to produce the predictive accuracies reported in the box plot. The box plot shows the median, upper and lower quartile as well as the maximum and minimum accuracies obtained at each level.

The horizontal line just below 70% represents the average comparative IOHMM performance (which does not take into account similarity knowledge). As expected *SimAlignGen* outperforms IOHMMs when the similarity bias provides an accurate model of the underlying HMM. Conversely, learning performance is degraded by a similarity bias that does not represent well the model generating the data. In this experiment the cross-over point between the performance of the two algorithms is between 60% and 70% similarity correctness.

This result raises the question of what kind of correctness scores can we expect in practical applications to programming by demonstration? To test this we presented a set of eleven Microsoft Window's users with documentation obtained from a corporate help desk describing a procedure for modifying their DNS network configuration parameters. We recorded their actions using our PBD system, and then manually annotated each snapshot-action pair with a label that specifies the documentation step it corresponds to (if any). Using the Microsoft Windows-specific similarity measure implemented in our system, we computed the similarity correctness score for this dataset. This similarity metric

combines several factors, including the previous action taken by the user, and the text on the title bar of the window with focus, etc. We obtained a similarity correctness of 88% for this procedure in the Windows domain, much above the 60-70% cross over point for the two algorithms. We expect similarity metrics will often such give strong hints in the PDB domain where GUIs are involved. Different parts of an underlying application will intentionally have many redundantly distinguishing GUI features as cues to the user of the application. We are simply leveraging those redundant cues as a similarity metric in learning the underlying HMM. Indeed, it was this intuition that GUI interface have such redundant features that drove us to consider this new class of HMM learning algorithms in the first place.

4.2 Experiment 2: SimIOHMM training time

In our second set of experiments we measure the performance of the SimIOHMM as part of a PBD system for capturing procedures on the Microsoft Windows GUI. In addition to the gains in accuracy demonstrated above, we show that SimIOHMM can result in substantial reductions in training time and better scalability as a function of training set size when compared with IOHMMs. Figure 3 shows the average training time verses number of training traces.

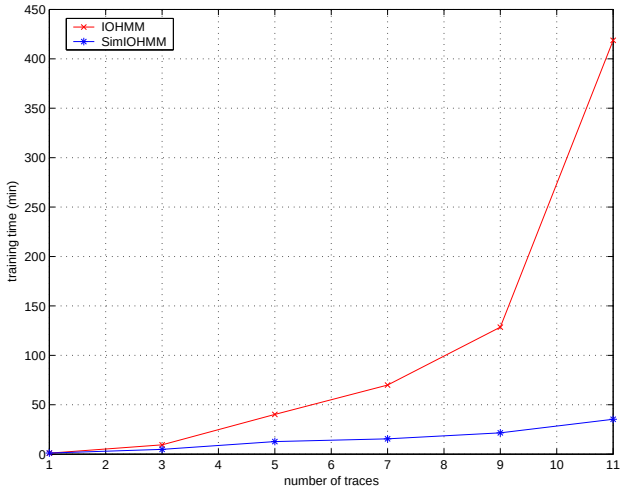

Fig. 3. Training time for IOHMM and SimIOHMM as a function of the number of training traces.

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 classifiers tend to be smaller. The main reason is that a snapshot-action pair can be aligned only with states 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 4 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 3. Due to the similarity bias, the SimIOHMM yields substantially more concentrated alignment probabilities than the IOHMM, and the difference between these entropies is an increasing function of the number of traces. These findings are also confirmed by the analysis of simulated data.

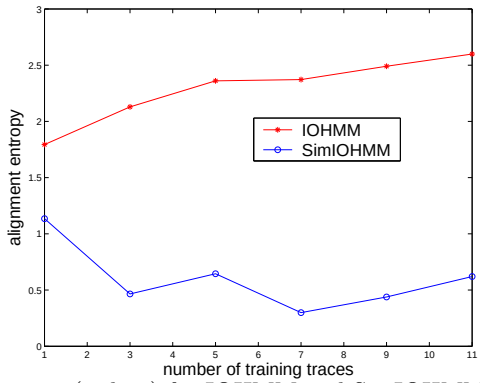

Fig. 4. Alignment entropy (in bits) for IOHMM and SimIOHMM as a function of the number of training traces.

5 Related work

Fasconi and Bengio [8] 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 [11], and temporal sequences in domains such as speech recognition. [12]

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 [4] and Eager [13] 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 [14] 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.

6 Conclusions

This paper presents an approach to the PBD 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 showing accuracy improvements as a function of synthetic similarity data, and large efficiency improvements over traces collected from a real-world procedure.

References

1. Rabiner, L.R., Juang, B.H.: An introduction to Hidden Markov Models. IEEE ASSP Magazine (1986) 4–15
2. Cypher, A., ed.: Watch what I do: Programming by demonstration. MIT Press, Cambridge, MA (1993)
3. Lieberman, H., ed.: Your Wish is My Command: Giving Users the Power to Instruct their Software. Morgan Kaufmann (2001)
4. Lau, T., Domingos, P., Weld, D.S.: Version space algebra and its application to programming by demonstration. In: Proc. Sevententh Int. Conf. on Machine Learning. (2000) 527–534
5. Maulsby, D., Witten, I.H.: Cima: an interactive concept learning system for end-user applications. Applied Artificial Intelligence **11** (1997) 653–671
6. XML Path Language: http://www.w3.org/tr/xpath (1999)
7. Baum, L.E., Petrie, T., Soules, G., Weiss, N.: A maximization technique occuring in the statstical analysis of probabilistic functions of Markov chains. Annals of Math. Statistics **41** (1970) 164–171
8. Frasconi, P., Bengio, Y.: An EM approach to grammatical inference: Input/Output HMMs. In: Proc. IEEE Int. Conf. Pattern Recognition, ICPR '94, Jerusalem (1994) 289–294
9. Bengio, Y., Frasconi, P.: Input-Output HMM's for sequence processing. IEEE Trans. Neural Networks **7** (1996) 1231–1249
10. Shields, P.: The Ergodic Theory of Discrete Sample Paths. Americal Mathematical Society (1996)
11. Krogh, A., Brown, M., Mian, I.S., Sjolander, K., Haussler, D.: Hidden Markov Models in computational biology: applications to protein modeling. Technical Report UCSC-CRL-93-32 (1993)
12. Rabiner, L.: A tutorial on Hidden Markov Models and selected applications in speech recognition. Proc. IEEE **77** (1989) 257–286
13. Cypher, A.: Eager: Programming repetitive tasks by demonstration. In Cypher, A., ed.: Watch What I Do: Programming by Demonstration. MIT Press, Cambridge, MA (1993) 205–217
14. Lau, T., Domingos, P., Weld, D.S.: Learning programs from traces using version space algebra. In: Proc. 2nd Int. Conf. on Knowledge Capture. (2003)