RC23999 (W0607-024) July 12, 2006
Computer Science

IBM Research Report

Augmentation-Based Learning

Vittorio Castelli[1], Daniel Oblinger[2], Lawrence D. Bergman[3]

[1]IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

[2]Currently with DARPA/IPTO
3701 Fairfax Drive
Arlington, VA 22203

[3]IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Augmentation-Based Learning

Vittorio Castelli
IBM T.J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10520
vittorio@us.ibm.com

Daniel Oblinger[*]
DARPA / IPTO
3701 Fairfax Drive
Arlington, VA 22203
aniel.Oblinger@DARPA.MIL

Lawrence D. Bergman
IBM T.J. Watson Research Center
19 Skyline Drive
Hawthorne, NY 10532
bergmanl@us.ibm.com

Abstract

We describe Augmentation-Based Learning, a new learning algorithm for Programming-by-Demonstration that allows the user to explicitly edit the procedure model even while demonstrating a task. We discuss the problems faced by learning algorithms that support seamless alternation of editing and demonstrations, and show how Augmentation-Based Learning solves them, while at the same time capturing complex procedure models with no additional user intervention.

Introduction

We describe Augmentation-Based Learning (ABL), a new learning algorithm for programming-by-demonstration (PBD) that infers computer-based task models from observations of the users' interactions with software applications, while also supporting direct user edits of the models.

PBD systems allow users to create, maintain, and customize task models by demonstrating how computer-based tasks should be performed. These task models are used for automating procedures, guiding users through unfamiliar tasks, and providing in-context documentation. For PBD to be effective, the learning algorithm that creates the task model must be able to rely on a small number of examples; ideally, the user should obtain benefits even from the task model created with the first demonstration.

This small data-sample requirement poses interesting challenges: typical ML algorithms are not fine-tuned for such small training sets. Handling these small samples by restricting the bias of the learner will impose too many restrictions on the types of task models that the induction algorithms can produce. Conversely, relaxing the bias will regularly produce models that are unacceptable to most users.

In the PBD literature the problem is solved by partially relaxing the bias restrictions and relying on user input besides demonstrations. There is a trade-off between requiring more user intervention (and therefore allowing the learner to use a larger hypothesis spaces) and constructing a usable PBD system, that is, one that relies mostly on demonstrations.

[*]This work was done while the author was with the IBM T.J. Watson Research Center.

Manual editing of the induced model is an ideal solution to this trade off: by editing, the user effectively and concisely interacts with the learning algorithm, which therefore does not require strong bias restrictions. Editing is more expressive than all other feedback mechanisms encountered in PBD, and hence it minimizes the costs of the feedback interactions between user and learning algorithm. We believe that ABL is the first algorithm to support seamless interleaving of editing with induction. Existing PBD systems only allow editing as a post-processing step of induction.

In the rest of the paper we first formalize the notion of a procedure model and introduce our notation. Then, we describe different types of bias restrictions and of user feedback mechanisms encountered in PBD. In the main section we describe ABL, discuss its bias, and show how it supports manual edits by solving two fundamental problems, the precedence and consistency problems. Readers interested in bibliographic references and in a discussion of related work are referred to the full version of the present paper (Oblinger, Castelli, & Bergman 2006), and to (Bergman *et al.* 2005; Prabaker, Bergman, & Castelli 2006), which describe DocWizards, a PBD system based on ABL.

Background and Preliminaries

Procedure model and procedure demonstrations

A *procedure model* $(\mathcal{S}, \mathcal{E}, \mathcal{P})$ is an extension of a directed graph, defined as follows. $\mathcal{S} = \{s\}$ is a set of *procedure steps*; these are a pairs (n, α), where n is a node in $\mathcal{N}$ (the collection of nodes in the graph) and α, the *action skeleton*, is a generalized action that, evaluated in a particular context (defined later in this section), yields a completely specified executable action. $\mathcal{E} = \{e\}$ is a set of *directed edges* that denote sequential ordering of steps. $\mathcal{P}$ is a set containing a *predicate* p for every edge in $\mathcal{E}$. Evaluated during playback, p denotes whether the corresponding edge must be followed (among the predicates of the outgoing edges of a node, one and only one must evaluate to *true* for every context). The *procedure structure* of a procedure model is the directed graph $(\mathcal{N}, \mathcal{E})$ obtained by discarding the action skeletons and the predicates.

Inducing a procedure model means using data to identify a structure and *instantiating* the model. Instantiation consists of inducing a step for each node, and inferring the edge predicates. The data is a collection $\mathbf{T}$ of procedure demonstrations, called a *training set*. A *procedure demonstration*, or *trace* $\mathbf{t}$, is a sequence of *state-action pairs* (SAPs) $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^m$, recorded while a user performs a task. A *subtrace* is any contiguous subsequence of a trace. A *state* $\mathbf{x}$ is a representation of the content of the GUI just prior to the execution of an action. An *action* $\mathbf{y}$ is a detailed representation of a user's interaction with an application, for example, pressing the "Ok" button in a file-selection dialog.

The state contains the attributes for inferring edge predicates, and, during playback, provides the context for instantiating executable actions from action skeletons. For example, the action skeleton "*Select the first item in the 'Package Explorer' list*" is instantiated to "*Select project1*" when the state shows that the first item in that list is called "*project1*".

Definition 1 *The* **alignment** *of a subtrace* $(\mathbf{x}_1, \mathbf{y}_1)$, $\ldots$, $(\mathbf{x}_m, \mathbf{y}_m)$ *with a sequence of nodes* $n_1, \ldots, n_m$ *in* $(\mathcal{N}, \mathcal{E})$ *is a one-to-one correspondence that maps* $(\mathbf{x}_i, \mathbf{y}_i)$ *to* n_i *for* $i = 1, \ldots, n$. *We say that the subtrace is* aligned *with the procedure structure.*

Thus, aligning means assigning steps as labels to SAPs.

Definition 2 *A subtrace* $(\mathbf{x}_1, \mathbf{y}_1)$, $\ldots$, $(\mathbf{x}_n, \mathbf{y}_m)$ *is* **consistent** *with a model if:*

- *the subtrace is aligned with a sequence of nodes* $n_1, \ldots, n_m$*, where* n_i *corresponds to step* s_i.
- *Sequentially instantiating the action skeletons* $\alpha_1, \ldots, \alpha_m$ *using* $\mathbf{x}_1, \ldots, \mathbf{x}_m$ *yields* $\mathbf{y}_1, \ldots, \mathbf{y}_m$.
- *For each* $i = 2$ *to* m*, there exists an edge* $_i$ *from* n_{i-1} *to* n_i *such that evaluating the associated predicate* p_i *using* $\mathbf{x}_i$ *yields true.*
- *If* $(\mathbf{x}_1, \mathbf{y}_1)$ *is the first SAP in a full procedure demonstration,* s_1 *must be an initial step, namely, a step without incoming edges.*

We say that *a model is consistent* with a collection of subtraces if each subtrace is consistent with the model.

Related Work: Biasing the Learner in PBD

In this section, we describe some common bias restrictions for learning algorithms used in PBD. In general, we distinguish two classes of problems solved by these algorithms: the generalization problem and the combined alignment-and-generalization problem.

The *generalization problem* is the problem of inducing a procedure model given a procedure structure and an alignment of the training data. Many existing PBD systems solve only the generalization problem, and the user must provide the structure and the alignment either directly (e.g., by labeling actions) or indirectly (via feedback mechanisms). For example, algorithms that produce procedure models consisting of an outer loop containing a fixed sequence of steps having user-specified length solve the generalization problem.

When the alignment is not given, the learning algorithm must build a procedure structure using a collection of traces and align these traces to the procedure structure before solving the generalization problem. We call this the *alignment problem*. A learner that builds a procedure structure, aligns it with the training data, and induces the model based on the alignment is said to solve the *combined alignment-and-generalization problem*.

Different bias restrictions are encountered in systems that solve the combined alignment-and-generalization problem. At one end of the spectrum, the learner is constrained to a hypothesis space containing a very restricted class of procedure structures. For example, there are algorithms that induce an outer loop enclosing a fixed sequence of steps of unspecified length. At the other end of the spectrum, algorithms are allowed to use a very large hypothesis space. For example, SimIOHMM solves a probabilistic version of the general combined alignment-and-generalization problem, but is by its nature a batch algorithm. ABL solves the general combined alignment-and-generalization problem, relies on a hypothesis space that contains arbitrarily complex procedure models, and operates on-line.

Related Work: User In The Loop

In this section, we briefly describe some of the common methods used in PBD to leverage user feedback.

The most common feedback mechanism is the explicit identification of the beginning and the end of each task demonstration. Another simple mechanism consists of notifying the learning algorithm when a suggestion offered during playback is not appropriate; in this case, the learning algorithm produces a different suggestion until the user is satisfied or no additional alternative exists. In a related approach, the learner maintains a collection of alternative procedure models and allows the user to select the most appropriate. This mechanism is commonly used as a post-processing step, but it has also been proposed to control the induction of individual steps during recording. User-in-the-loop techniques have also found application in support of predicate inference. Some systems, for example, allow the user to highlight UI widgets that contain information relevant to the inference of decision predicates. Active learning has also been used, where the algorithm prompts the user for additional information when unsure.

Direct editing of the procedure model provides the user with better control over the learning process than any of the mechanisms described above, but is also difficult to integrate with the induction process. PBD systems that allow the user to edit the script produced by the learning algorithm restrict editing to be a post-processing step. We believe that ABL is the first learning algorithm to support the interleaving of editing operations with demonstrations, and to use user edits to constrain subsequent induction.

The ABL Algorithm

ABL solves the combined aligned-and-generalization problem by incrementally constructing a procedure model while simultaneously ensuring the alignment of the data with the model. Editing operations, allowed at any point, destroy this alignment, making the data and the procedure structure inconsistent. ABL solves the inconsistencies by carefully tracking the parts of the alignment that are not affected by editing. ABL also gives precedence to the edits over previous examples and ensures that data observed before an edit is not used to undo the edit effects. In this section, we first formalize the notion of editing operations and define the problems that a learning algorithm that allows edits must solve. Then we describe in detail how ABL attains the behavior described above.

Definition 3 *An* **editing operation** *is a manual transformation of a procedure structure* $(\mathcal{N}, \mathcal{E})$ *into a different procedure structure* $(\mathcal{N}', \mathcal{E}')$ *through moving, copying, adding, or deleting nodes and edges.*

To support edits, a learning algorithm must produce a human-readable representation of the procedure model. More importantly, it has to solve the *precedence* problem and the *consistency* problem. The precedence problem arises when the learning algorithm inadvertently counteracts a manual edit using data observed before that edit. This problem is solved by just discarding all the data observed before the edit, as one can easily see by constructing simple examples. The consistency problem arises because user edits can produce procedure structures that are not consistent with previous demonstrations, and the learner is then asked to induce predicates with inconsistent data.

Incremental Procedure Induction in ABL

We now describe the three main components of ABL: the incremental construction of the procedure structure, the induction of steps, and the inference of predicates.

Procedure Structure Construction. ABL starts with an empty procedure and incrementally modifies it in real-time as new examples are observed. When presented a new SAP, ABL modifies the existing procedure structure using transformations called *augmentations*. Intuitively, an augmentation is any transformation that modifies the structure of a procedure only by adding nodes and edges; more precisely:

Definition 4 *An* **augmentation** *associated with a SAP* σ *is a transformation from a procedure structure* $(\mathcal{N}, \mathcal{E})$ *to a procedure structure* $(\mathcal{N}', \mathcal{E}')$ *satisfying* $\mathcal{N} \subseteq \mathcal{N}'$ *and* $\mathcal{E} \subseteq \mathcal{E}'$.

Hence, applying an augmentation produces a new procedure structure that contains all the nodes and the edges of the original one. Examples of augmentations are the introduction of a branch, the conversion of a sequence of steps into a loop, and the extension of an unterminated block by a new step.

ABL maintains a collection $\mathcal{M}$ of procedure models consistent with the observed demonstrations (and therefore the training traces are aligned with each model in $\mathcal{M}$). When a new SAP σ is observed, ABL updates $\mathcal{M}$ by finding all the augmentations that produce new models consistent with the past observations and with σ. More specifically, let μ be a model in $\mathcal{M}$, let n_0 be its (unique) node aligned with the last SAP observed before σ. Given σ, ABL determines the set of steps in μ that are consistent with σ, constructs the set $\mathcal{N}(\sigma)$ containing the corresponding nodes, and adds to $\mathcal{N}(\sigma)$ a brand new node. Then, for each $n \in \mathcal{N}(\sigma)$ ABL identifies the collection $\mathcal{A}_\mu(n)$ of augmentations that would create in μ an edge from n_0 to n.

Different augmentations have different *costs*: for example, if an edge from n_0 to n already exists in μ, no change to the structure of μ is needed, and the corresponding "null" augmentation has small cost. Conversely, creating a branching edge from n_0 to a brand new node has a high cost. Each augmentation in $\mathcal{A}_\mu(n)$ yields both a new model and a new alignment of the training set. In this alignment, each previous observation retains its existing correspondence with a node in μ, while the new observation is aligned with n. Given the alignment produced by an augmentation $\mathfrak{A}$, ABL tries to produce a new model, by inducing steps and inferring predicates. This operation can fail, in which case $\mathfrak{A}$ is discarded, or succeed, in which case an instantiation cost is computed that captures the difficulty of inducing the steps and inferring the predicates. The surviving augmentations applied to the corresponding models yield a new collection of models $\mathcal{M}'(\sigma)$. Each model in $\mathcal{M}'(\sigma)$ has an overall cost, which captures the complexity of the model, computed by combining the cost of augmentations and the cost of instantiation. The model in $\mathcal{M}'(\sigma)$ with the smallest cost is presented to the user.

Step Induction. Inducing a step means constructing a generalized action α, which can be represented as a quadruple $(\mathfrak{y}, \mathbb{S}, \mathbb{D}, \mathfrak{t})$. Here, $\mathfrak{y}$ is an action type (e.g., "uncheck a check box"), $\mathbb{S}$ is a set of source widgets, $\mathbb{D}$ is a set of destination widgets, and $\mathfrak{t}$ is a text entry. For example, the action of dragging a set of files from one directory and dropping them in another is described as follows: $\mathfrak{y}$ is "drag-and-drop", $\mathbb{S}$ is the collection of icons in the source directory that are dragged, $\mathbb{D}$ is the destination directory, and $\mathfrak{t}$ is empty. The induction of a step s in a model $\mu \in \mathcal{M}$ is the process of constructing (*generalizing*) the relevant elements of the quadruple using the SAPs aligned with s. ABL can generalize steps using any generalization grammar described in earlier PBD work.

Predicate Inference. The predicates of the edges leaving a node n (denoted by $\mathcal{P}(n)$) are induced using data $\tilde{\mathbf{X}}(n)$ selected as follows. Consider first the collection $\{(SAP_i^{(1)}, SAP_i^{(2)})\} = \mathbf{T}(n)$ of length-2 subtraces where $SAP_i^{(1)}$ is aligned with node n and $SAP_i^{(2)}$ with a node in $\mathcal{N}(n)$, the set of destinations of edges originating from n. Then $\tilde{\mathbf{X}}(n)$ is defined as the collection $\{SAP_i^{(2)} | (SAP_i^{(1)}, SAP_i^{(2)}) \in \mathbf{T}(n)\}$. Each state in $\tilde{\mathbf{X}}(s)$ is labeled with the corresponding aligned step. This labeled data is used to learn a decision tree classifier, which is then translated into a disjunction of *rules*. Here, a rule is a conjunction of terms having the form *attribute.value* $\in$ *valueSet*, where *attribute* is a specification of a property of a widget on the screen, and *valueSet* is a list of strings. Different terms in a rule use different attributes. The reason for the translation is that the disjunction of rules can be presented to the user in an easily readable format, while inspecting a decision tree classifier is typically difficult.

Solving the alignment-and-generalization problem.

From the description of the incremental model induction, we conclude that ABL solves the combined alignment-and-generalization problem by:

- Incrementally constructing a correspondence between each new SAP and a node in the procedure structure (this solves the alignment problem).

- Solving the generalization problem by constructing steps and inducing predicates as described above, using the data aligned with the procedure structure.

ABL Support of Edits

To effectively support manual editing, the learning algorithm must produce a human-readable, easily understandable representation of the procedure model, such as the script-like models yielded by ABL. A short ABL procedure example, which resets the properties of all the projects in an Eclipse workspace to their default values, is the following:

```
ForEach item in "Package Explorer"
    Right-click on item
    Select popup menu item "Properties"
    Click "Restore Defaults"
    Click "OK"
```

Here, control structures and action types are in boldface, constants are between quotes, and variables are in italics. The language of the script contains actions, "if-then-elseif-else" conditional statements, and loops, which can be appropriately nested and combined. As a consequence, the language can represent the structure of complex tasks.

ABL solves the precedence problem by incrementally learning using augmentations. When a new SAP is observed, ABL modifies the procedure structure using only this new SAP. When the user performs an editing operation, the resulting procedure structure becomes the starting point for the next augmentation. Since ABL does not use previous SAPs to modify the procedure structure, it satisfies the requirement that data observed before an edit cannot be used to "undo" the structural effects of the edit.

ABL solves the consistency problem by carefully selecting the data for induction. More specifically, since ABL uses the alignment correspondence to select the data for step induction and predicate inference, we have solved the consistency problem by redefining the concepts of alignment of SAPs with nodes and of consistency of subtraces in the presence of edits. The following conventions ensure that data observed before the edit and that could confuse the learning algorithm are not used for induction.

- SAPs aligned with a node before an editing operation are also aligned with the same node after the editing operation. When a copy operation on a node occurs, SAPs aligned with the original node are also aligned with its copy.
- The set $\mathbf{T}_i^\epsilon$ of subtraces that ABL considers consistent with the ith user edit ϵ_i is defined as follows: let $\mathbf{T}_i$ be the collection of (complete and partial) traces observed before ϵ_i; then $\mathbf{T}_i^\epsilon$ is defined as the collection of the longest subtraces in $\mathbf{T}_i$ that are consistent with the procedure model produced by ϵ_i according to Definition 2 (in other words, if $\tilde{\mathbf{t}} \in \mathbf{T}_i^\epsilon$, and $\tilde{\mathbf{t}} \subseteq \mathbf{t}$, any subset of $\mathbf{t}$ strictly containing $\tilde{\mathbf{t}}$ is not consistent with the model).

Some Remarks

ABL easily supports other user control mechanisms that substantially simplify PBD. The first is a special case of editing: the ability to discard the most recently observed action. We have observed that, in practice, the user occasionally interacts with the wrong control, quickly recovers from the mistake, removes the irrelevant scripts from the procedure model, and continues executing.

ABL also supports learning from partial demonstrations. This particular mechanism is very useful to capture the localized variability of a task due to different environment settings; it is also useful to maintain scripts in conjunction with editing: if a subtask within a model becomes obsolete, the user can remove it manually and demonstrate the current way of performing the subtask.

Since ABL maintains a collection of alternative procedure models ranked by an appropriate score, it automatically provides the user with the ability of visually inspecting the different alternatives, and, if desired, select a model other than the one with the highest score.

Experiments described in (Prabaker, Bergman, & Castelli 2006) have shown that even inexperienced users can effectively leverage ABL's features to construct models of nontrivial software installation and configuration tasks.

Conclusions

We have described Augmentation-Based Learning, a new learning algorithm for PBD that supports seamless alternation of procedure demonstrations and explicit user edits. ABL incrementally combines data from multiple demonstrations into complex procedure structures that are represented to the user as scripts in a simple programming language. This programming language supports blocks, conditionals, and loops, which can be induced using demonstrations alone. Editing provide the user with very precise and direct control on the products of the learning algorithm. To support edits, ABL solves the precedence problem by using only data observed after the edit to modify the procedure structure obtained via the edit. ABL solves the consistency problem by judiciously selecting data for induction, and disregarding data that is inconsistent with subsequent edits.

Laboratory experiments (Prabaker, Bergman, & Castelli 2006) have shown that ABL is effective even for inexperienced authors users, who easily leveraged editing to create models of complex installation and configuration tasks.

References

Bergman, L.; Castelli, V.; Lau, T.; and Oblinger, D. 2005. Docwizards: a system for authoring follow-me documentation wizards. In *Proc. 18th annual ACM Symp. on User Interface Software and Technology, UIST2005*, 191–200. ACM Press.

Oblinger, D.; Castelli, V.; and Bergman, L. 2006. Augmentation-based learning, combining observations and user edits for programming-by-demonstration. In *Proc. 2006 Int. Conf. on Intelligent User Interfaces*, 202–209, runner–up for best paper award.

Prabaker, M.; Bergman, L.; and Castelli, V. 2006. An evaluation of using programming by demonstration and guided walkthrough techniques for authoring and following documentation. In *Proc, of CHI,* Best of CHI Nominee.