POIROT – Integrated Learning of Web Service Procedures

Mark Burstein, Robert Laddaga, David McDonald, Michael Cox, Brett Benyo,
Paul Robertson, Talib Hussain, Marshall Brinn and Drew McDermott[1]
BBN Technologies and Yale University[1]
10 Moulton Street, Cambridge, MA 02138
burstein, rladdaga, dmcdonald, mcox, bbenyo, probertson, thussain and mbrinn @bbn.com
mcdermott@cs.yale.edu

Abstract

POIROT is an integration framework for combining machine learning mechanisms to learn hierarchical models of web services procedures from a single or very small set of demonstration examples. The system is organized around a shared representation language for communications with a central hypothesis blackboard. Component learning systems share semantic representations of their hypotheses (generalizations) and inferences about demonstration traces. To further the process, components may generate learning goals for other learning components. POIROT's learners or *hypothesis formers* develop workflows that include order dependencies, subgoals, and decision criteria for selecting or prioritizing subtasks and service parameters. *Hypothesis evaluators*, guided by POIROT's *meta-control* component, plan experiments to confirm or disconfirm hypotheses extracted from these learning products. Collectively, they create methods that POIROT can use to reproduce the demonstration and solve similar problems. After its first phase of development, POIROT has demonstrated it can learn some moderately complex hierarchical task models from semantic traces of user-generated service transaction sequences at a level that is approaching human performance on the same learning task.

Introduction

POIROT (Plan Order Induction by Reasoning from One Trial) is an architecture and set of learning components that collectively forms a multi-strategy learning system for Web Service procedures. As one of two large-scale collaborative efforts in the DARPA Integrated Learning Program, we are exploring techniques for applying a variety of learning and reasoning strategies to learn hierarchical procedures from one or at most several examples. Our approach is guided by a meta-controller that manages a set of active *learning goals* of different types that cause different components to engage in generalization, explanation, combination, experimentation and diagnosis of typically incomplete learned models. As the process proceeds, POIROT tests its own conclusions on new problems by calling the same web services it observed the demonstrator using.

POIROT learns hierarchical task models given a single demonstration of a task, a semantically annotated trace of a sequence of web service calls. The annotation is possible due to the representation of the web services using OWL-S (Martin et al. 2007), which provides semantic typing of inputs and outputs, and representations of each services' preconditions and effects in a form consistent with AI planning operator definitions. POIROT's learning is emergent from the work of a number of largely independent reasoning components accumulating shared questions, hypotheses and answers on a central blackboard. To be successful, POIROT must apply inductive, abductive and explanation-guided generalization techniques. It also must perform self-guided experiments to resolve conflicts and confirm that the learned procedure will work. In the future, POIROT will reason about the space of problems for which the procedure will apply in order to scope its applicability and improve its confidence in what was learned.

We believe that these distinct learning and reasoning processes must be strategically applied to different aspects of this learning task. Our approach utilizes a meta-control executive called the *learning moderator* that communicates with other components using a semantic blackboard built on a persistent RDF store (Sesame). The moderator guides POIROT's use of its multiple bottom-up, primarily inductive components and its top-down, explanation-based ones. It signals when to propose *learning hypotheses,* generalizations of the trace in the form of process models, and when to evaluate them. Another meta-control related component called the STITCHER combines learning hypotheses into a single more complete model which can be tested by the planning and execution subsystem on new problems.

Learning goals reflect the system's need to fill gaps in what components have learned or acquire additional information that would allow components to learn more. POIROT's blackboard and learning moderator maintain these open learning questions, and components declaring the capability to address them can respond by subscribing to classes of these goals. In addition to managing tasks addressing learning goals, the moderator manages the phasing of tasks to monitor for the completeness of the combined model, initiate testing of the learned model, plan how to experimentally resolve conflicts between hypotheses created by different learners, and the diagnose and repair problems with the learned task models.

In this paper, we describe POIROT's approach to learning, and its capability after our first phase of development. To date, only about half of the components have been integrated into the system, but it already does quite well on simple problems due to the capabilities of the learning mechanisms it is combining. We begin with a brief overview of POIROT's task and its architecture. Next we describe key motivations for and design aspects of the Learnable Task Modeling Language (LTML) that is the interlingua of POIROT. We then describe the modules that have been integrated

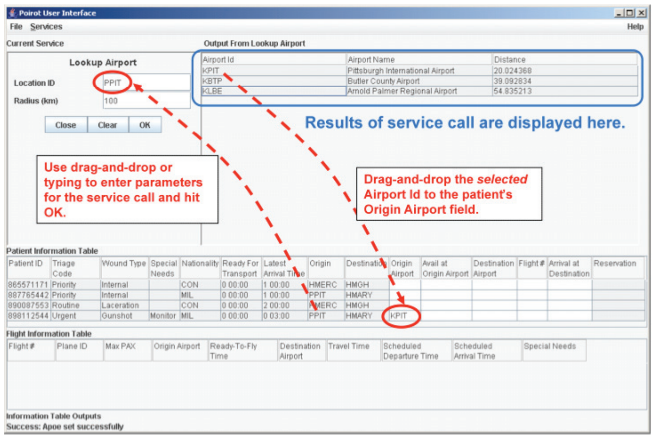
Copyright © 2008, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.
This project is supported by DARPA IPTO under contract FA8650-06-C-7606. Approved for Public Release, Distribution Unlimited.

thus far and the approach used to combine these results into an executable workflow model. Finally, we discuss the results of phase one testing and provide a brief summary of previous work in the area before concluding.

Overview of Learning Task

POIROT's objective is to form a generalized hierarchical task model from 'observed' semantic traces of sequences of web service transactions. Our test domain is medical evacuation airlift planning, which is analogous to the use of web services for trip planning when different web sites are used to book airline tickets, reserves local transportation (e.g., taxis, trains buses) and hotel rooms. Some differences from these commercial services are that planning medical evacuations can include requesting new flights be published, the use of helicopters and ambulances rather than trains and taxis, and the planning for and reservation of support medical personnel and equipment to accompany patients on their journeys. A typical demonstration used as the basis for POIROT's learning shows how a user builds a schedule for a number of patients with specific requirements. Each patient is available at a specified time and location to be moved, with a given destination hospital, latest arrival time and identified special needs (for medical care and equipment en route).

POIROT's observations are of a logged sequence of calls to web services collected when a user performs tasks such as looking up airports by geographic location, finding available flights to and from those airports, reserving seats on flights and reserving hospital beds at the destinations. Figure 1 shows a screenshot of the GUI used to call these services when developing these demonstrations. A video demonstration of this interface was also presented to our human subjects as training before we asked them to try the task. The results of these experiments were used to compare human and POIROT performance on the learning task.



Current Service

Lookup Airport

Location ID: PPIT

Radius (km): 100

Output From Lookup Airport

Airport Id	Airport Name	Distance
KPIT	Pittsburgh International Airport	20.024368
KBTP	Butler County Airport	39.092834
KLBE	Arnold Palmer Regional Airport	54.835213

Results of service call are displayed here.

Use drag-and-drop or typing to enter parameters for the service call and hit OK.

Drag-and-drop the *selected* Airport Id to the patient's Origin Airport field.

Patient Information Table

Patient ID	Triage Code	Wound Type	Special Needs	Nationality	Ready For Transport	Latest Arrival Time	Origin	Destination	Origin Airport	Avail at Origin Airport	Destination Airport	Flight #	Arrival at Destination	Reservation
865571171	Priority	Internal		CON	0 00:00	1 00:00	HMERC	HMGH						
887765442	Priority	Internal		MIL	0 00:00	1 00:00	PPIT	HMARY						
890087553	Routine	Laceration		CON	0 00:00	2 00:00	HMERC	HMGH						
898112544	Urgent	Gunshot	Monitor	MIL	0 00:00	0 03:00	PPIT	HMARY	KPIT					

Flight Information Table

Flight #	Plane ID	Max PAX	Origin Airport	Ready-To-Fly Time	Destination Airport	Travel Time	Scheduled Departure Time	Scheduled Arrival Time	Special Needs

Information Table Outputs
Success: Apoe set successfully

Figure 1: Demonstration UI

Figure 2 shows the procedure that was demonstrated in these experiments and indicates places where people or POIROT or both failed to capture the full complexity of the process. Sometimes this was because the demonstration did not fully show what was involved, and sometimes it was because not all of aspects of the model are being learned yet. We will cover some of these errors in the evaluation section later.

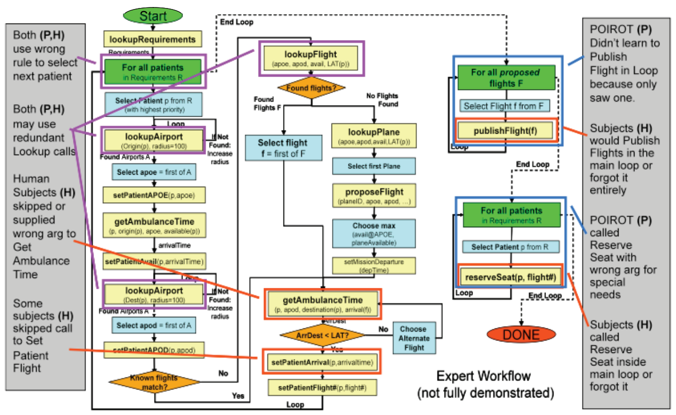

Both (P,H) use wrong rule to select next patient
Both (P,H) may use redundant Lookup calls
Human Subjects (H) skipped or supplied wrong arg to Get Ambulance Time
Some subjects (H) skipped call to Set Patient Flight
Start
lookupRequirements
Requirements
For all patients in Requirements R
Select Patient p from R (with highest priority)
Loop
lookupAirport (Origin(p), radius=100)
If Not Found: Increase radius
Found Airports A
Select apoe = first of A
setPatientAPOE(p,apoe)
getAmbulanceTime (p, origin(p), apoe, available(p))
arrivalTime
setPatientAvail(p,arrivalTime)
Loop
lookupAirport (Dest(p), radius=100)
If Not Found: Increase radius
Found Airports A
Select apod = first of A
setPatientAPOD(p,apod)
Known flights match?
No
Yes
End Loop
lookupFlight (apoe, apod, avail, LAT(p))
Found flights?
Found Flights F
No Flights Found
Select flight f = first of F
lookupPlane (apoe,apod,avail,LAT(p))
Select first Plane
proposeFlight (planeID, apoe, apod, ...)
Choose max (avail@APOE, planeAvailable)
setMissionDeparture (depTime)
getAmbulanceTime (p, apod, destination(p), arrival(f))
ArrDest
ArrDest < LAT?
No
Choose Alternate Flight
Yes
setPatientArrival(p,arrivaltime)
setPatientFlight#(p,flight#)
Loop
For all proposed flights F
Select Flight f from F
publishFlight(f)
Loop
End Loop
For all patients in Requirements R
Select Patient p from R
reserveSeat(p, flight#)
Loop
End Loop
DONE
Expert Workflow (not fully demonstrated)
POIROT (P) Didn't learn to Publish Flight in Loop because only saw one.
Subjects (H) would Publish Flights in the main loop or forgot it entirely
POIROT (P) called Reserve Seat with wrong arg for special needs
Subjects (H) called Reserve Seat inside main loop or forgot it

Figure 2: Demonstrated Procedure and Observed Learning Errors

POIROT's Integration Architecture

POIROT's software framework uses a combination of agent and semantic web service architectural elements to integrate multiple learning and execution components. Figure 3 shows the current architecture with existing Phase I and planned future components. Components with black lettering were not fully integrated for Phase I. The central hypothesis blackboard where learning hypotheses and commentary are exchanged is built on a persistent RDF store (Sesame), extended to support context-bounded queries and a publish-subscribe interface that is used to invoke POIROT components. Although modules in POIROT use very different approaches to generate learning hypotheses, all hypotheses are ultimately represented using the Learnable Task Modeling Language (LTML) for placement on the blackboard. This language combines aspects of the Web Ontology Language OWL, the OWL ontologies for semantic web services OWL-S, and the planning language PDDL. As a result of translating all of these different models into LTML, that they can be directly compared, commented upon, recombined, and subjected to evaluation via experiment.

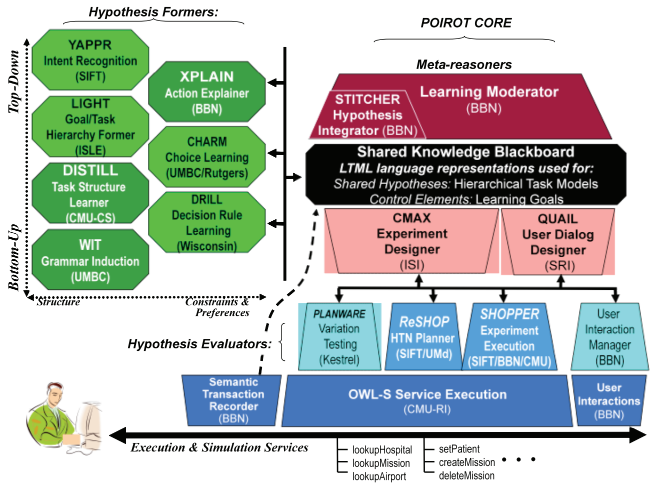

Hypothesis Formers:
POIROT CORE
Top-Down
Bottom-Up
YAPPR Intent Recognition (SIFT)
LIGHT Goal/Task Hierarchy Former (ISLE)
DISTILL Task Structure Learner (CMU-CS)
WIT Grammar Induction (UMBC)
XPLAIN Action Explainer (BBN)
CHARM Choice Learning (UMBC/Rutgers)
DRILL Decision Rule Learning (Wisconsin)
Structure
Constraints & Preferences
Meta-reasoners
STITCHER Hypothesis Integrator (BBN)
Learning Moderator (BBN)
Shared Knowledge Blackboard
LTML language representations used for:
Shared Hypotheses: Hierarchical Task Models
Control Elements: Learning Goals
CMAX Experiment Designer (ISI)
QUAIL User Dialog Designer (SRI)
Hypothesis Evaluators:
PLANWARE Variation Testing (Kestrel)
ReSHOP HTN Planner (SIFT/UMd)
SHOPPER Experiment Execution (SIFT/BBN/CMU)
User Interaction Manager (BBN)
Semantic Transaction Recorder (BBN)
OWL-S Service Execution (CMU-RI)
User Interactions (BBN)
Execution & Simulation Services
lookupHospital
lookupMission
lookupAirport
setPatient
createMission
deleteMission
. . .

Figure 3: POIROT Architecture

Hypothesis formers use bottom up (inductive) and top down (abductive or explanation-based) generalization techniques to form workflow hypotheses that are posted on the blackboard. The learning moderator communicates about internal system learning goals and tasks with the other modules using a publish-subscribe protocol, and receives task status feedback through the same mechanism.

When fully populated the list of hypothesis formers will include:

WIT – a system for learning the temporal order of steps using grammar induction techniques (Oates and Yaman 2007)

DISTILL – a model for learning universal plans from causal ordering relations (Winner and Veloso 2003).

XPLAIN – A system for generating explanations of user actions using a library of explanation patterns, based on the MetaAqua story understanding system (Cox and Ram 1995, 1999).

DRILL – An inductive logic programming approach to learning decision rules, focusing on learning multi-attribute ranking rules for ordering object consideration for loops. (Torrey et al. 2007)

CHARM – Another technique for learning how to make choices that explicitly uses background knowledge about the target task to filter the ranking criteria, based in part on (desJardins 1995)

LIGHT – A technique for learning hierarchical, goal-indexed subprocedures for use by an HTN planning and execution system (Choi and Langley 2005)

YAPPR – A probabilistic plan recognition system that assumes a previously learned HTN model, but can identify unknown new elements. (Geib and Goldman 2003)

Using an HTN planner for experimentation

POIROT tests what it has learned by selecting sets of task methods that together should be sufficient to enable an HTN Planner to reenact the demonstration or to perform similar tasks. ReSHOP, based on the SHOP2 Planner (Nau et al. 1999) has been incorporated into POIROT by Robert Goldman at SIFT Technologies and Dana Nau at the University of Maryland. ReSHOP interprets LTML descriptions of HTN planning domains and problems, develops executable task plans, and executes them plan interpreter called SHOPPER. SHOPPER, in turn, invokes domain web services through calls to the OWL-S virtual machine (OVM), developed by Katia Sycara's group at CMU Robotics Institute (Paolucci et al. 2003).

POIROT evaluates its learned task models by executing them and checking for errors and invalid results. The use of semantic web services, described using parts of LTML based directly on OWL-S, provides the learners with declarative models of the primitive actions of the domain being observed and enables the system (through the OVM) to perform experiments by actually calling the demonstrated services. It allows learned procedural models to be executed even though the services used were not known to the system beforehand.

In Phase II, we will introduce another module, CMAX (Cohen and Morrison 07) which will allow POIROT to design its own experiment by creating workflows for ReSHOP and SHOPPER.

The Learnable Task Modeling Language

The modularity of our architecture depends on an ability to share hypotheses and conclusions from very different learning systems. This required descriptions of what was learned be represented in a common language. Drew McDermott, Mark Burstein and Robert Goldman with support from a number of others on the team developed a language for representing the accumulated hypotheses about generalizations of the observed task called the Learnable Task Modeling Language (LTML). LTML is based in part on the PDDL planning language, the Web Ontology Language OWL, and its semantic web services extension OWL-S (Martin et al 2007). It uses a more compact and manageable surface syntax than that of OWL and RDF. LTML is used to represent planning methods, plans (observational traces), and domain concepts. It is also used to capture hypothesis annotation information, including such as things as provenance of hypotheses, conflicts between hypotheses, explanations of aspects of traces, and internal learning goals and tasks having to do with all of these.

LTML has to satisfy a number of sometimes competing requirements. It has to be capable of describing hierarchical task models (methods) in a way that is acceptable to state of the art HTN planners such as SHOP2, but in a way that allows essentially arbitrary fragments of such descriptions to be independently generated and annotated as hypotheses. It has to support descriptions of relationships among hypotheses, such as conflicts or congruencies, and support the capture of explanations or justifications for hypotheses. As it is the language of the central blackboard, it is also used for describing internal learning goals that make reference to hypotheses as objects of discourse.

Controlling learning

Our overall learning framework is based in part on the concept of goal-driven learning, described previously (Hunter 1990, Cox & Ram 1995, desJardins 1995) as 'knowledge acquisition planning' or 'planning to learn'. Goal-driven learning leads to targeted searches for explanations when observations don't fit into the developing model, and to the creation of new knowledge goals about the availability of choices and decision processes associated with task elaboration. It also allows us to orchestrate and manage a set of learning components by posting learning goals that these components can respond to in an order that respects their knowledge needs and products.

Of the components listed above, WIT, DISTILL and DRILL are primarily bottom-up learners, using information provided in the demonstration trace and only a small amount of additional background information. But DRILL, which is learning loop-ordering criteria, needs to be told where the loops are and what objects are being iterated over. The meta-controller therefore delays the construction of this learning goal until there is a hypothesis about loops that requires it.

The top-down learners are those that formulate hypotheses by explanation-based techniques. These include XPLAIN, LIGHT, CHARM and YAPPR. These systems are used to hierarchically organize and provide rationale for parts of the learned model, and, as a result to fill in additional details about how to make particular decisions or preserve key associations so that the overall model is successful at achieving the stated goal.

Phase I System Elements

During the first phase of the project, recently concluded, WIT, DISTILL and XPLAIN were the primary sources of procedural hypotheses. The STITCHER was used to combine the hypotheses produced by these three learners, producing a set of hierarchical task methods that could be combined by ReSHOP to form an executable plan. POIROT tested what it had learned by trying to recapitulate the demonstration, starting with the given requirements. Another component, DRILL, was able to discover reasonable rules for ordering the handling of patients, but was not fully integrated before the evaluation was completed. In this section, we describe the first three components in more detail, in order to motivate a discussion of how POIROT approached hypothesis synthesis in the STITCHER, and learned an executable model of the demonstrated procedure.

Task structure determination

In Phase I, POIROT relied primarily on two learning systems for task structure determination, each generalizing directly on the observed trace. Both techniques utilized the dependencies between information gathering actions (data querying services) and information consuming ones (when service parameters are filled in using acquired information) as a means of inferring dependencies between actions (service calls). WIT, however, used this information only to model explicit data flow, and used observed action order as its primary means of inducing structure.

The WIT system from UMBC is based on prior work on context-free grammar induction (Oates et al. 2002). The demonstration POIROT observes has embedded within it somewhat regular repetitions of steps for each patient. WIT develops an alphabet of generalized steps and then induces a transition model over this alphabet where the loops in the demonstration appear as cycles, and alternate paths within a cycle become branches in the loop.

WIT then decomposes this state graph into methods using heuristics based on identified decision points. While the model it produces can serve as a recognizer for acceptable workflows that closely mirror the demonstration, it cannot by itself be used to execute these workflows because WIT does not determine the conditions that dictate which branches to follow, or recognize when loops terminate.

DISTILL extends the work of Winner and Veloso (2002, 2003). It automatically learns 'domain-specific planners', or universal procedures, from demonstrated examples. DISTILL produces workflow methods where the steps are loosely ordered but strongly conditionalized to run only when their context-specific pre-conditions are satisfied. These run-time conditions have the additional role of identifying, from the current state, the information that needs to be supplied as parameters of the service calls being executed. The overall product is a dependency directed execution model for the class of problems demonstrated.

Both WIT and DISTILL build and post LTML representations of sets of HTN methods as hypotheses about how to generalize the trace. Each is, however only a partial solution to the learning task. WIT leaves branch conditions unspecified, and DISTILL ignores steps that it cannot relate directly to the goal.

Using explanations for unobserved dependencies

POIROT can be viewed in part as processing a stream of observations to explain how they form a coherent whole. While bottom-up hypotheses formers can identify regularities in a demonstrated workflow, aspects of the workflow that are not repeated or do not have observable causal support are more difficult to generalize in a useful way without applying additional background knowledge to explain their relevance. Explanations are produced by inferring relationships between steps using explanatory schemas that tie the observed steps to the domain goals of the demonstrator, and to hidden processes and constraints. POIROT's XPLAIN component is a hypothesis former based on Meta-AQUA (Cox & Ram 1999), a goal-driven explanation and learning system that seeks to understand each observation in its input stream as part of a story. It relies on domain knowledge, a case library of prior plan schemas and a set of general explanation patterns that are used to characterize useful explanations involving that background knowledge.

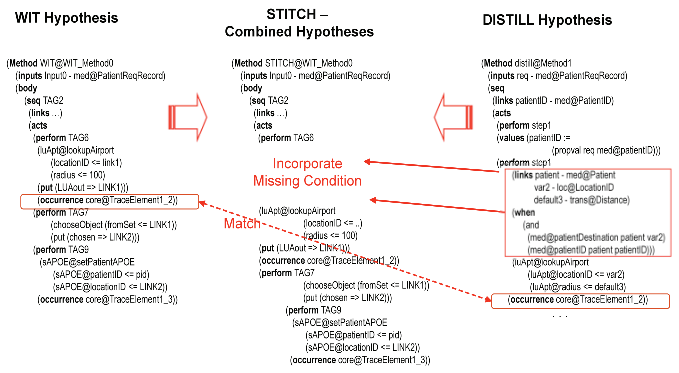
POIROT uses XPLAIN to provide rationales for steps that do not have strong observable connections to other steps in the demonstration. For example, steps that gather information used to confirm a precondition for another step appear unmotivated because the later step does not use the result as an input (the condition being satisfied is enough). In workflows from phase one, XPLAIN critically concludes that making flight reservations for each patient supports the user's goal of moving the patients and should follow selection of the flight even though there is no producer-consumer relationship between the steps.

Stitching hypotheses together

When all of the hypothesis formers have generated their individual hypotheses, POIROT tries to combine them in order to form a complete model that it can test. In the process it may discover conflicts or remaining gaps in the collective model. A component called the STITCHER has this responsibility.

As stated earlier, all of the models individually produced by the learners were inadequate as execution models for the demonstrated workflow. The WIT model lacked conditions on branches and loops, the DISTILL model lacked some necessary steps. XPLAIN only models steps that it has explanations for. None had a complete model of the process.

The process of combining these models is essentially one of repeated, incremental analogical matching and mapping, related to that in (Burstein 1986). As shown in Figure 4, steps in the various models (which look like programs written in LTML) are aligned by considering which steps in the trace they are generalizations of.



```
(Method WIT@WIT_Method0
  (inputs Input0 - med@PatientReqRecord)
  (body
    (seq TAG2
      (links ...)
      (acts
        (perform TAG6
          (luApt@lookupAirport
              (locationID <= link1)
              (radius <= 100)
          (put (LUAout => LINK1)))
          (occurrence core@TraceElement1_2))
        (perform TAG7
              (chooseObject (fromSet <= LINK1))
              (put (chosen => LINK2)))
        (perform TAG9
          (sAPOE@setPatientAPOE
              (sAPOE@patientID <= pid)
              (sAPOE@locationID <= LINK2))
          (occurrence core@TraceElement1_3))
```

```
(Method STITCH@WIT_Method0
  (inputs Input0 - med@PatientReqRecord)
  (body
    (seq TAG2
      (links ...)
      (acts
        (perform TAG6

          (luApt@lookupAirport
                    (locationID <= ..)
                    (radius <= 100)
          (put (LUAout => LINK1)))
          (occurrence core@TraceElement1_2))
          (perform TAG7
                    (chooseObject (fromSet <= LINK1))
                    (put (chosen => LINK2)))
              (perform TAG9
                (sAPOE@setPatientAPOE
                  (sAPOE@patientID <= pid)
                  (sAPOE@locationID <= LINK2))
              (occurrence core@TraceElement1_3))
```

```
(Method distill@Method1
  (inputs req - med@PatientReqRecord)
  (seq
    (links patientID - med@PatientID)
    (acts
      (perform step1
      (values (patientID :=
                   (propval req med@patientID)))
      (perform step1
        (links patient - med@Patient
               var2 - loc@LocationID
               default3 - trans@Distance)
        (when
           (and
             (med@patientDestination patient var2)
             (med@patientID patient patientID)))
        (luApt@lookupAirport
             (luApt@locationID <= var2)
             (luApt@radius <= default3)
        (occurrence core@TraceElement1_2))
                  . . .
```

Figure 4: Stitching LTML Hypotheses

This information is provided by the learners in their LTML output as 'occurrence' annotations on each step. Once the generalized steps in the models are aligned, their parameter bindings can be unified, producing variable correspondences between the various hypotheses. Pre-conditions (when clauses) on steps are then unified. In phase I, the preconditions were based on DISTILL step conditions, after variable renaming. Conversely, the most conservative model of step order, produced by WIT, ended up being used. DISTILL's weaker model of step order was consistent with but less complete than WIT's model.

After the initial phase of step merging, conditions on branches are inferred by state regression from the preconditions on steps within those branches. The state regression eliminated terms that were introduced by intermediate steps after branch points.

The final product is a model that has the ordering structure of the WIT model, the conditional structure of the DISTILL model, and the parameter binding information contributed by all three models. The model had the following performance, relative to human subjects tested on the same material.

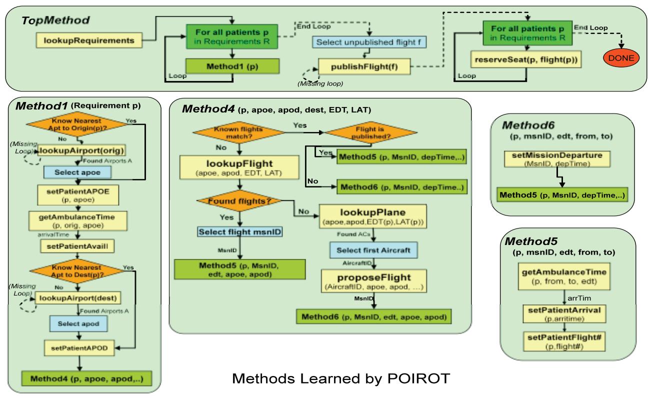

TopMethod
lookupRequirements
For all patients p in Requirements R
Method1 (p)
Loop
End Loop
Select unpublished flight f
publishFlight(f)
(Missing loop)
For all patients p in Requirements R
End Loop
reserveSeat(p, flight(p))
Loop
DONE
Method1 (Requirement p)
Know Nearest Apt to Origin(p)?
Yes
No
(Missing Loop)
lookupAirport(orig)
Found Airports A
Select apoe
setPatientAPOE (p, apoe)
getAmbulanceTime (p, orig, apoe)
arrivalTime
setPatientAvail
Know Nearest Apt to Dest(p)?
Yes
No
(Missing Loop)
lookupAirport(dest)
Found Airports A
Select apod
setPatientAPOD
Method4 (p, apoe, apod,..)
Method4 (p, apoe, apod, dest, EDT, LAT)
Known flights match?
Yes
No
Flight is published?
Yes
Method5 (p, MsnID, depTime,..)
No
Method6 (p, MsnID, depTime..)
lookupFlight (apoe, apod, EDT, LAT)
Found flights?
Yes
Select flight msnID
MsnID
Method5 (p, MsnID, edt, apoe, apod)
No
lookupPlane (apoe,apod,EDT(p),LAT(p))
Found ACs
Select first Aircraft
AircraftID
proposeFlight (AircraftID, apoe, apod, ...)
MsnID
Method6 (p, MsnID, edt, apoe, apod)
Method6 (p, msnID, edt, from, to)
setMissionDeparture (MsnID, depTime)
Method5 (p, MsnID, depTime,..)
Method5 (p, msnID, edt, from, to)
getAmbulanceTime (p, from, to, edt)
arrTim
setPatientArrival (p,arritime)
setPatientFlight# (p,flight#)
Methods Learned by POIROT

Figure 5: Learned Methods

Experimental results

POIROT's performance was compared to human subjects working similar problems with similar background information presented to all subjects. Twenty human subjects (Pittsburgh area college students) all saw the same demonstration and the same test problems. POIROT saw the same demonstration as the human subjects, but was given nine different problems, all modest random variations on the human test problem. Table 1 shows three ways of aggregating the human subject test results (mean, median, mean of top 50%), and an average of the POIROT scores on five metrics. Steps and Choices are percent correct of parameter assignments and service calls respectively. Accuracy is a comparison with an expert workflow model. Coverage is a comparison with the steps explicitly in the demonstration. Goal Achievement is a measure of whether the right reservations were made for each patient, and several other details.

A one tailed T-test was used to determine that the POIROT average performance was at least 85% of human subject performance with greater than 99% confidence, even compared to the top 50% of human subjects. Note that POIROT's accuracy scores were quite similar to and coverage scores were better than the college students'. More interesting is that POIROT did significantly worse than the college students on goal achievement. So, without being noticeably more accurate, or better at imitating the demonstration, the college students were nevertheless better at building an evacuation plan.

Table 1. Human Subject and POIROT Statistics

	Choice	**Step**	**Choice**	**Step**	**Goal**
	Accuracy		**Coverage**		
mean	0.80	0.81	0.85	0.88	0.96
median	0.83	0.85	0.87	0.91	1.00
^50%	0.90	0.90	0.97	0.98	1.00
Poirot	0.86	0.83	0.98	0.98	0.86

In Figure 2, we presented the expert workflow for the kind of medical evacuation problems subjects were asked to solve. We annotated the steps where mistakes were made, orange for human subject mistakes (H), blue for POIROT mistakes (P), and purple for mistakes that both made. For example, selecting the wrong patient to act on next was a mistake made by both human subjects and POIROT. Some of the other mistakes were due to conditions of the workflow not shown in the demonstration.

Related Work

Investigation of multi-strategy reasoning and learning systems can be traced as far back as to the Pandemonium system (Selfridge, 1959) that combined multiple competing pattern recognition methods. Early papers on multi-strategy learning (Michalski, 1993) taxonomized learning methods to develop procedures combining multiple learning algorithms.

A large body of explanation-based learning work has focused on acquiring and using domain knowledge to reduce planning search. Some of the best known examples of the application of explanation-based learning for planning include learning control rules, (Minton, 1988); case-based and analogical reasoning, (Kambhampati & Hendler, 1992; Veloso, 1992, Burstein, 1985) and hierarchical and skeletal planning (Knoblock, 1994). Many of these methods suffer from the *utility* problem; learning more information can be counterproductive because of costs for storage and management of the information to determine which information is useful for solving a particular problem.

Other work has focused on analyzing example plans to reveal a strategy for planning in a particular domain. One example of this approach is Shavlik's algorithm BAGGER2 (Shavlik, 1990), which uses example solutions and domains and background knowledge to learn an algorithm for problem-solving also in the form of recurrences. BAGGER2 was able to learn such algorithms from very few examples, but relied on background knowledge and was unable to identify parallel repetition steps in a transport-domain problem. DISTILL can be seen as an extension of this work but with no reliance on background knowledge and addressing richly structured parallel repetition.

Our learning of task hierarchies is related to methods for learning macro-operators (e.g., Iba, 1988; Mooney, 1989), in that they explicitly specify the order in which to apply operators, but they do not typically support recursive references.

Conclusions

The POIROT Project is in the second of four year-long phases of development. The overall objective is the development of an architecture that draws on and combines the strengths of a collection of machine learning approaches to solve the difficult problem of learning complex procedural models from a single demonstration. We have already demonstrated near human level performance on some aspects of simple medical evacuation problems, and our goal for this phase is address a number of issues with regard to uncertain choices and robustness to local failures, utilizing to a greater degree abstract knowledge of resources, choice mechanisms, and physical constraints.

Acknowledgements

We gratefully acknowledge other members of the POIROT team: Paul Cohen, Marie desJardins, Chris Geib, Yolanda Gil, Robert Goldman, Karen Haigh, Pat Langley, Mike Lewis, Michael Littman, Steve Minton, Tom Mitchell, Karen Myers, Dana Nau, Tim Oates, Jude Shavlik, Katia Sycara, Manuela Veloso and their colleagues for numerous discussions and contributions to this effort. We also thank the rest of the BBN team and especially: Michael Atigetchi, Ray Tomlinson, Yvette Tenney, and John Ostwald for all of their work.

References

Burstein, M. (1986) "Concept Formation by Incremental Analogical Reasoning and Debugging", *Machine Learning: An Artificial Intelligence Approach*, Volume II, Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., eds., Morgan Kaufmann Publishers, Inc., Los Altos, Ca., 1986, pp. 351-370.

Choi, D., & Langley, P. (2005). “Learning teleoreactive logic programs from problem solving,” *Proceedings of the Fifteenth International Conference on Inductive Logic Programming* (pp. 51-68). Bonn, Germany: Springer.

Cox, M. T., & Ram, A. (1999). “Introspective multistrategy learning: On the construction of learning strategies,” *Artificial Intelligence*, *112*, 1-55

Cox, M. T., & Ram, A. (1995). “Interacting learning-goals: Treating learning as a planning task,” in J.-P. Haton, M. Keane & M. Manago (Eds.), *Advances in case-based reasoning* (pp. 60-74). Berlin: Springer-Verlag.

desJardins, M. (1995). “Goal-directed learning: A decision-theoretic model for deciding what next to learn,” in A. Ram & D. Leake (eds.), *Goal-Driven Learning*. Cambridge, MA: MIT Press/Bradford Books, pp. 241-249.

Geib, C. W. and Goldman, R. P. Recognizing plan/goal abandonment. In Proceedings of IJCAI 2003, 2003.

Hunter, L. E. (1990). “Planning to learn,” *Proceedings of Twelfth Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Lawrence Erlbaum Associates, pp. 261-276.

Iba, G., A Heuristic Approach to the Discovery of Macro-Operators. *Machine Learning*, 3(4):285-317, 1989.

Knoblock, C. Automatically Generating Abstractions for Planning. *Artificial Intelligence*, 68 (2):243-302, 1994.

Kambhampati, S., & Hendler, J. A. (1992). A validation structure based theory of plan modification and reuse. Artificial Intelligence, 55, 193-258.

Martin, D., Burstein, M., McDermott, D., McIlRaith, S., Paolucci, M., Sycara, K., McGuiness, D., Sirin, E. and Srinivasan, N. "Bringing Semantics to Web Services with OWL-S" *WWW Journal* (2007)

McDermott, D., Burstein, M., Goldman, R. and McDonald, D. LTML Manual. http://poirot.bbn.com/ltml/

Michalski, R. S. (Ed.). (1993). Multistrategy learning [Special issue]. Machine Learning, 11 (2/3).

Minton, S. 1988 Learning Effective Search Control Knowledge: an Explanation-Based Approach. Doctoral Thesis., CMU.

Morrison , C. & Cohen, P. (2007). Designing Experiments to Test and Improve Hypothesized Planning Knowledge Derived From Demonstration. AAAI Press, Technical Report WS-07-02.

Mooney, R.: The Effect of Rule Use on the Utility of Explanation-Based Learning. IJCAI 1989: 725-730

Nau, D., Cao, Y., Lotem, A., & Munoz-Avila, H. (1999). “SHOP: Simple hierarchical ordered planner,” *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence* (pp. 968-973). Stockholm: Morgan Kaufmann.

Oates, T., Desai, D. and Bhat, V. Learning k-Reversible Context-Free Grammars from Postive Structural Examples**.** In *Proceedings of the 19th Annual International Conference on Machine Learning*, 2002.

Paolucci, M, Ankolekar, A., Srinivasan, N., Sycara, K, (2003) “The DAML-S Virtual Machine,“ Proceedings of the Second International Semantic Web conference (ISWC 2003). Sanibel Island, FL.

Ram, A., and Leake, D. (Eds.), *Goal-Driven Learning* (pp. 421-437). Cambridge, MA: MIT Press/Bradford Books.

Selfridge, O. G. (1959). “Pandemonium: A paradigm for learning,” in D. V. Blake and A. M. Uttley (Eds.), Proceedings of the Symposium on Mechanisation of Thought Processes (pp. 511-529), London: H. M. Stationary Office.

Shavlik, J. (1990) “Acquiring recursive and iterative concepts with explanation based learning,” *Machine Learning*, 5, p. 39–70.

Torrey, L., Shavlik, J., Walker, T. & Maclin, R. (2007). **“**Relational Macros for Transfer in Reinforcement Learning” Proceedings of the Seventeenth Conference on Inductive Logic Programming, Corvallis, Oregon.

Winner, E. and Veloso, M. Analyzing Plans with Conditional Effects. In AIPS-2002. Toulouse, April 2002.

Winner, E. and Veloso, M. DISTILL: Learning Domain-Specic Planners by Example. In Proceedings of ICML-2003. Washington, D.C., August 2003.

Veloso, M. (1992). Learning by analogical reasoning in general problem solving. Doctoral dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Yaman, F. and Oates, T. (2007) “Workflow Inference: What To Do with One Example and No Semantics” In (Burstein and Hendler, eds.) Proceedings of the AAAI-07 Workshop “Acquiring Planning Knowledge via Demonstration.” 46-51 *AAAI Press*.