Apprenticeship Learning of Query Based Problem Solving Rules

A. H. Wadaa S. N. Shen
Computer Science Department
Old Dominion University
Norfolk, VA, 23529

Abstract

Attempting to synthesize general rules that embed an expert's problem solving expertise in a domain is a hard problem. Recently proposed Learning Apprentice systems promise a feasible solution to this problem. An apprentice would take problem solving episodes as training examples and apply machine learning methods to build a domain theory of problem solving expertise. In certain domains, solving a problem essentially entails multiple, possibly dependent, queries performed on an underlying database. We propose LASSP, an apprentice system for learning problem solving rules in such domains. Contrary to other learning apprentice systems, LASSP has a domain theory that is initially empty, yet ever improves in terms of its strength and correctness as successive problem solving episodes are experienced. LASSP has the desirable property of being able to learn and exploit its domain theory concurrently. Foremost, LASSP can asynchronously integrate expertise of multiple experts into a common domain theory, in a seamless manner. This parallel assimilation of knowledge potentially enables the apprentice to develop its domain theory in a fraction of the time normally needed in serial assimilation approaches.

1 Introduction

Learning Apprentice Systems (LAS) have been recently proposed as a new approach to bypass the data acquisition bottleneck [1, 5, 12, 13]. The main rationale of a LAS is to utilize past problem solving episodes as training examples of problem solving rules, Learning such rules enables the apprentice to support future problem solving episodes. The benefits of the LAS approach are several. Shifting the load of expertise elicitation in the large part to the machine, and dynamically enhancing learned expertise by continuous experience, gained from successive problem solving episodes, are the major advantages. Several learning apprentice systems have been proposed in the literature, some notable examples are LEAP [12] in the domain of VLSI design, GENESIS [5] in the domain of story understanding and DISCIPLE [13] in the domain of action plans. One common property of these LAS examples is that they are based on some strong (or weak) initial domain theory. A strong domain theory enables learning a general rule or schemata from a single training example of this rule or schemata. Another property common to these systems is that a solution to a problem in their domain is a purely inferential reasoning process in the domain theory, that is no function evaluation is necessary to perform any step towards the solution of a problem [13].

In this paper we propose LASSP, an apprentice system for learning problem solving rules in query based problem solving (QPS) domains. QPS domains are characterized by problems that are solved mainly by querying, an underlying database. LASSP initially has an empty domain theory. As a result of learning during problem solving episodes, both the strength and correctness of its theory are enhanced.

The paper is organized as follows. In Section 2 we present a general paradigm for apprenticeship learning, and establish a multicriteria classification of LASs pertaining to this paradigm. In Section 3, we propose LASSP and characterize it along the criteria defined earlier. A detailed discussion of LASSP's behaviour during a problem solving session is discussed in this section. Next, A formal model of the problem solving episode is given in Section 4. In Section 5, we describe the learning and expertise enhancement techniques in LASSP. Finally, a critic of the proposed system and some future research directions are discussed in Section 6.

2 A general paradigm for apprenticeship learning

Three fundamental properties of a Learning Apprentice System are the ability to learn and enhance expertise from examples, the ability to apply learned expertise, and the ability to explain or justify its behavior. The last property has great practical importance, since it facilitates the processes of debugging and performance evaluation. A general paradigm for apprenticeship learning incorporates two interacting subsystems (see Figure 1). The first, referred to as the Problem solving Platform (PSP), supports Problem solving episodes performed by experts and users. The former type of episodes are passed to the LAS as training examples. PSP acts as an interface between humans and the LAS. The second subsystem, being the Learning Apprentice System, learns exper-
tise from past episodes and supports experts and users by formulating episodes for solving future problems. The interaction between a human and the system until a problem has been solved is referred to as problem solving session. Each problem is solved in one problem solving session. Notice that experts have problem solving expertise that the apprentice can learn, while users are assumed to have little or no expertise, and hence will only use, rather than teach, the system.

An important issue that has to be resolved is that of characterizing our LAS, such that a meaningful comparison to other systems is possible. To approach this problem we propose a number of classification criteria considered to have dominant effect in so far as the properties and behaviour of apprentice systems are concerned. By carefully analyzing the proposed paradigm for Apprenticeship Learning, it follows that any LAS might be classified along three principal criteria, that form a classification space for all LASs. These criteria can be defined as follows.

**Domain of problems** This is the domain in which Problem solving expertise should be learned. The particular nature of problems and problem solving steps in that domain can greatly affect both the representation and manipulation of the domain theory. Some examples of such domains are classification [1], Planning [2], Automatic program development [3], Inductive generalization and specialization [12].

**Experience episodes** These are total or partial problem solving situations resulting from experts’ problem solving sessions and presented to the LAS as training examples. Such training examples form the basis for the process of domain theory development. The learned theory is largely dependent on the type of experience episodes available. Some examples of experience episode types are complete training examples [1], partial training examples [13], and failure of a domain theory rule while attempting to apply it [13].

**Learning engine** The set of processes cooperating towards developing the domain theory are collectively referred to as the learning engine. Several Learning Methods can be utilized in isolation, or integrated with others to form a learning engine. Both the constructs and contents of the learned domain theory are determined in part by the particular methods used. Some examples of such methods are Explanation based Learning [2, 11], Inductive generalization and specialization [13], Analogical and case based reasoning [13], and Learning by asking questions [12, 13].

In general terms, LASSP has a problem domain similar in some aspects to that of Planning and Automatic program generation, handles complete training examples, partial training examples, and failure of domain theory rules, and integrates rote learning and guided inductive generalization and specialization in its learning engine.

### 3 LASSP

In QPS domains a Problem solving platform supports experts and users in solving data intensive, query based problems. A problem solution process will typically incorporate several queries to a database. New queries might be formulated as a result of preceding queries, and might take as input their output data, and hence some queries are data dependent on others. In our opinion, the feasibility of having a LAS in these domains is well justified for three main reasons. First, QPS domains are common place, typically exemplified by a conventional relational database system. The DBMS represents our PSP, and a problem is solved by extracting and processing certain sets of data from the database. The subqueries performed to extract these data sets and their dependencies can be thought of as a problem solving episode. Second, problem solving episodes can be easily represented and stored in these systems by the platform itself, hence cheap training examples are available for the LAS. Consider again the example of a conventional relational database system. Since the query manager has to translate all subqueries to an equivalent representation in relational algebra, and resolve all their interdependencies as a standard part of processing an application program code, then a representation for the problem solving episode can be readily obtained from the query manager. Third, By definition, the semantic and structural relationships given by the database schema of the underlying database can potentially guide and bias the learning process, specially that of inductive generalization and specialization.

#### 3.1 Characterizing LASSP

The characteristics of our proposed learning apprentice system regarding each of the three classification criteria described above are now discussed in detail.

**LASSP problems domain** As mentioned above, the distinctive characteristics of QPS domains are that problems deal with massive amounts of data stored in databases and that the solution steps entail function evaluations, in the form of successive queries on the database, possibly bearing dependencies to one another. Consequently, problems in this domain differ from those in domains like planning [13], or VLSI circuit design [12] in at least two ways. First, problem solving steps here are not just logical deductions or inductions in the domain theory. Both logical Inference and function evaluation are essential for solving Problems. A Second distinction concerns the operators which are used at each solution step. In QPS domains arbitrary functions, or rather queries, can be used in solving a problem, as opposed to a limited number of well defined static operators as in actions planning [13] for example. These differences are of major effect on the design of a LAS for such domains.

**LASSP experience episodes** An experience episode is
a situation in which LASSP might learn, that is it might affect its domain theory. In our system, two types of episodes are assumed possible during an expert's problem solving session. Either an expert supplies the total steps for solving a particular domain problem, referred to as a "problem solving episode", or, an expert rejects a step(s) being proposed by the apprentice in an episode.

**LASSP learning engine** Several methods have been utilized as learning engines for different learning apprentice systems. Recently, some approaches for integrating several learning methods in the same learning engine have been proposed, for example [12, 13]. The appeal of integration is twofold. Under the realistic assumption of only a weak theory about the domain, no one learning method is able to handle the situation, since multiple types of inference might be needed. As an example, explanation based learning methods (EBL) must first deductively prove in their domain theories that an example belongs to the goal concept as a prerequisite for learning from that example. Thus, the basic EBL method fails in case of a weak domain theory. Another advantage is that integration proposes a solution to the problem of "falling off the knowledge cliff" problem. In simple terms this is the problem of rapid deterioration in the learning system's performance, in reaction to any slight shift beyond the reach of its domain theory [13]. This problem can be attributed in part to a weak domain theory. For these reasons, we have followed the integration approach in designing LASSP's learning engine.

### 3.2 The learning process in LASSP

As for the rationale behind the learning process in LASSP, it is to always assume a learned rule in the domain theory to be applicable by default under a particular condition of the world, until a failure of that rule under that condition is experienced, thus falsifying the assumption. Now let's consider the learning process in some detail. Let a hypothesis be an arbitrary condition defined on the state of the underlying database. Thus, any rule in the domain theory is either true (applicable), or false (not applicable) under a given hypothesis. Effectively, each rule defines a subspace of the hypothesis space, such that under all hypothesis contained in that subspace, the rule is true. This subspace is referred to as the version space of the rule.

The rationale behind learning in LASSP can hence be stated as follows. For a particular rule in the domain theory, the apprentice always assumes that a hypothesis in the hypothesis space belongs also to the version space of that rule, until a failure of the rule under that hypothesis is experienced, thereby enabling a correction of the boundaries of the rule's version space. The goal of the apprentice is to learn sufficient rules for solving all types of problems in its domain. Before presenting a formal model for the problem solving episode and the domain theory let's first consider the roles of the PSP, LASSP, and the Domain Expert in a problem solving session.

Initially the domain theory of the apprentice is empty. The first Problem solving episode is passed by the PSP in a suitable format to the apprentice. The apprentice then transforms the episode to a set of rules, as will be described later. The rules are then added to the domain theory. For each subsequent problem in the domain, the apprentice attempts to formulate a problem solving episode based on his domain theory. If successful, the apprentice proposes the episode, through the PSP to the expert for consideration.

An expert might reject a step(s) in the episode and in return supplies alternatives to these steps. Such alternatives are then transformed by the apprentice to new rules in the domain theory. The expert must also supply a reason for rejecting a step pertaining to the particular problem being solved. The expert's reason is an arbitrary boolean condition and the truth of this condition at any point in time, should be calculable by the PSP. Rejection conditions impose either a necessary or a sufficient condition for the applicability of a certain rule in the domain theory. Such conditions are then incorporated into the premises of the relevant rules each as a conjunction if a necessary condition, or as a disjunction if a sufficient condition. Note that adding a conjunction has the effect of eliminating some hypothesis from the rule's version space, while adding a disjunction effectively adds a set of hypothesis to this space.

If however the apprentice fails to formulate an episode (for lack of expertise), or is only successful in formulating a partial problem solving episode, the expert must then supply the missing steps in form of queries, and the corresponding domain theory rules are then generated. For each new attempt to solve a problem the cycle repeats.

Notice that the same behaviour applies to a session for a normal user, however a normal user can only ask the system to supply alternative episodes for solving a particular problem, but can not change or add to the domain theory.

### 4 Formalizing the Problem Solving Episode

A formal model for the problem solving episode is now presented. We use this model as a framework for describing the domain theory manipulation techniques in LASSP.

Any problem solved by an expert or a user in our domain is assumed to be of a particular type. Problem types represent a semantic classification developed by domain experts. A type identifier for the problem being solved is given by the expert at the beginning of each session, and is passed to the LAS. This identifier serves as the entry point to the apprentice's domain theory. Problem type identifiers come from a domain-wide standard dictionary. As a convention, the symbol $P^u$ is used to refer to a problem of type $u$. Henceforth, a relational database system, being a typical representative of QPS domains, is assumed in our dis-
A query $q$ in the query set $Q$ can be formally represented by an ordered triple $(I, X, D)$, where $I$ is a domainwide identifier for $q$, $X$ is an expression in relational algebra [4] representing the semantics of the query, $D$ is the set of all data objects referenced by $X$. A member $d$ in $D$ is an ordered pair $(s,t)$, representing one data object, such that $(s)$ is a sequence number serving as an identifier for that object in set $D$, and $(t)$ is the data type of the object. Examples of data types are relations, integers, reals, etc. Members of $D$ referenced for input, are collectively called the input data set of $q$. The other hand members of $D$ generated or altered by operations in $X$ are called the output data set. It is always true that,

$$I \equiv (I_1, X_1, D_1), q_2 = (I_2, X_2, D_2), \text{then} \quad I_1 = I_2 \iff X_1 \equiv X_2$$

where $(\equiv)$ denotes equivalence of expressions in relational algebra. Let $db$ be the set of all namable and processable data objects in the database. Thus a member of $db$ might be an entity relation, a snapshot relation, a view relation, a simple attribute in a relation, etc... [4] Members of $db$ have the same formal representation as members in $D$.

**Binding relation** Let $d_{jm}$ denote the data object having a number $n$ and belongs to the query that has an identifier $I$. If in episode $E$ the data object $d_{jm}$ takes on the value of $d_{jn}$, then it is said that $d_{jm}$ is bound to $d_{jn}$ in episode $E$. This relation is symbolically represented by

$$d_{jm} \bowtie d_{jn}$$

where $(\bowtie)$ denotes the bound to relation. $d_{jm}$ is called bound to object and $d_{jn}$ is called the binding object. Let the set $D^o$ be the union of set $db$ and all object sets $D$, belonging to members of the query set $Q$ in the problem solving episode $E$. The binding relation $BR$ in episode $E$ is a relation defined on the set $D^o$ as follows

$$BR = \{ (d_{jm}, d_{jn}) | d_{jm} \bowtie d_{jn} \text{ in } E \}$$

As mentioned before, $BR$ defines a partial order relation say $R$ on the query set $Q$, that can be defined as follows

$$R = \{ (J, I) \exists (d_{jm}, d_{jn}) \in BR \text{ in } E \} , \text{ and}$$

$$I, X, D), (J, Y, C) \in Q$$

If $(J, I) \in R$ then $J$ precedes $I$ in the order relation. Notice that in the special case where $d_{jm}$ is bound to a database object, $J$ should be replaced by the special symbol $db$. For each problem solving episode $E$ the PSP represents and stores both the query set and the binding relation of $E$. Different episodes can have non disjoint query sets. A stored query can be used in any subsequent episode for solving a problem of any type in the domain. Two special types of queries have particular importance.

**Initial queries** These are queries in $Q$ having all members in $D$ bound only to members in $db$, that is

$$I \text{ is an Initial query } \iff (\forall (J, I) \in R \Rightarrow (J = db))$$

**Terminal queries** A query $I$ is a terminal query in $E$ if no other query $J$ in $Q$ has a member in $D_T$ that is bound to a member in $D_T$ in episode $E$, that is

$$I \text{ is a terminal query } \iff \not \exists (I, J) \in R, \text{ for all } J \in Q.$$ It is assumed that the union of output data sets of all terminal queries in an episode represents the solution of the problem. The apprentice always augments each episode with a logical step called the goal query, that has an identifier $I_{goal}$ being the problem type $(w)$ itself, and a representation in relational algebra $X_{goal}$ being nil. The goal query has a data object set $D_{goal}$ satisfying the following condition.

Assume $q_n = (I_n, X_n, D_n)$ is a terminal query in $E$, therefore, $(d_n \in D_n) \Rightarrow \exists d_{goal} \in D_{goal}, \text{ such that}$$

$$(d_n, d_{goal}) \in BR \text{ in } E$$

The goal query notion simplifies searching in the domain theory, since each goal query represents in effect a pointer to a chain of rules corresponding to the relevant steps for solving problems of type $I_{goal}$.

## 5 LASSP’s domain theory

We propose a learning engine for LASSP that integrates two different learning methods. The first is rote learning, applied if the input to the apprentice is a total problem solving episode. Thus a total episode represents a positive example of an algorithm for solving a particular domain problem. The second is guided inductive generalization and specialization, applied if the apprentice experiences a rejection of a particular step in a problem solving episode. Thus a step rejection condition effectively represents a negative example for the applicability condition of the rule in the domain theory corresponding to the rejected step.
5.1 Domain theory rules

Assume a new problem solving episode \( E \) for a particular problem \( P^w \) is given to the apprentice. Taking \( E \) as a positive example of a solution to problems of type \( w \), the apprentice formulates first order predicate logic rules to represent episode \( E \). The rules thus formulated are referred to as step rules, and are totally dependent on the binding relation \( BR \) that is implied by \( E \). For each query \( q_n \) belonging to \( Q \), such that \( q_n = (I_n, X_n, D_n) \), \( c = \{D_n\} \), a single step rule \( r_n \) is formulated as follows

\[
If(Z_1 \land Z_2 \land Z_3 \land \ldots \land Z_c) \Rightarrow I_n,
\]

where \( Z_1 = (a \oplus b), (a, b) \in BR, (a = d_n) \in D_n \)

The premise of a step rule is the set of all bindings for data objects referenced by the query corresponding to the relevant step. A simple interpretation of the step rule is that step \( q_n \) is reachable and thus \( X_n \) can be performed only if all bindings of data objects referenced by \( q_n \) are true, in the sense that values of all objects \( (b) \) are available. If episode \( E \) is the first to be encountered for problems of type \( w \), then all step rules are directly added to the domain theory. All other subsequent episodes will have the effect of adding new rules and/or changing the premises of the step rules existing in the domain theory. A special type of step rule is formulated for the goal node. For each goal node \((I_{goal}, X_{goal}, D_{goal})\), \( c = \{D_{goal}\} \), a single step rule is formulated as follows

\[
If(Z_1 \land Z_2 \land Z_3 \land \ldots \land Z_c) \Rightarrow I_{goal},
\]

where \( Z_1 = (a \oplus b), (a, b) \in BR, (a = d_{goal}) \in D_{goal} \)

The premise of a step rule newly added to the domain theory is referred to as the initial condition of the rule. By definition the premise of a step rule at any time represents the upper bound hypothesis of the rule's version space, at that time. It should be noted that multiple rules having \( I_n \) as a consequence, but different premises can exist in the domain theory, thus indicating that query \( q_n \) is reachable in several ways.

5.2 Manipulating the domain theory

Learning in LASSP is a continuous process of domain theory enhancement and augmentation. Enhancement essentially is a result of successive corrective specializations and generalization of step rules premises.

Specializing an overGeneralization
If a step \( q_n \) belonging to \( Q \) is rejected in the current problem solving episode as being not applicable, then the expert is asked to give a reason, for his action. That is, a condition \((c)\), such that \((c)\) is a sufficient condition for failure of the rule \( r_n \), where \( r_n \) is the rule applied by the apprentice in this case to reach \( q_n \). Notice that \((c)\) defines a set of negative examples for the applicability condition of \( r_n \). Thus, the apprentice attempts to minimally specialize the current upper bound hypothesis for \( r_n \)'s version space, by eliminating all negative examples covered by \((c)\). The apprentice adds a conjunction \( not(c) \), assumed to be an unnecessary condition for the truth of \( r_n \) to the premise of that rule. Thus, the rule

\[
If((condition) \Rightarrow I_n, \text{ becomes} \quad If((condition) \land \neg(c)) \Rightarrow I_n
\]

In principle \((c)\) is an arbitrary boolean expression that might need evaluating functions and/or querying the database for assessing its truth value.

Generalizing an overspecialization
If the expert determines that a certain step rule \( r_n \) is applicable in a particular episode, but the premise of \( r_n \) is false under the state of the database at the time, then the expert must again give a reason for his assertion. That is, a condition \((c)\), such that \((c)\) is a sufficient condition for the applicability of \( r_n \). Again condition \((c)\) defines a set of positive examples for the applicability condition of \( r_n \). Thus, the apprentice attempts to minimally generalize the current upper bound hypothesis for \( r_n \)'s version space, by adding all positive examples covered by \((c)\). The apprentice adds a disjunction \((c)\), assumed to be a sufficient condition for the truth of \( r_n \) to the premise of that rule. Thus the failing rule

\[
If((condition) \Rightarrow I_n, \text{ becomes} \quad If((condition) \lor (c)) \Rightarrow I_n
\]

Condition \((c)\) is an arbitrary boolean expression having the same properties as condition \((c)\) described above.

6 Conclusions and future research

An apprentice system for learning problem solving rules, that are based on queries, has been proposed. Its abilities and limitations can now be discussed. A LAS for Query Based Problem Solving systems is justifiable mainly because such domains are common place and normal problem solving sessions supply enough training examples. Since, these examples are not synthetic they should pose negligible overhead. LASSP’s domain theory is dynamic in the sense that learning new rules, correction of existing rules, as well as their exploitation in problem solving sessions proceed concurrently. The domain theory is a function of all problem solving episodes experienced in the past. This continuous education property enables the apprentice to correct its theory on basis of failures encountered, as well as integrate any new algorithms appearing in subsequent problem solving episodes. Of foremost importance, is LASSP’s ability to assimilate expertise from multiple experts in parallel. Thus, the time necessary for developing a domain theory is sharply reduced. Besides, the indirect interaction of the input of different experts through the common domain theory, potentially enables rapid trapping of errors in premises of domain theory rules.
The apprentice's domain theory is potentially affected by every expert's problem solving session. This property has great practical importance, since it enables the apprentice to correct its learned theory by learning from failures in episodes it formulates and proposes to experts. Second it enables him to seamlessly and incrementally integrate new techniques as they appear in problem solving episodes.

Several directions can be followed to study further the learning problem in QPS domains. A Performance study of LASSP regarding learning and convergence speeds towards a credible domain theory in the context of different QPS domains is essential to gain insight into its behaviour. An important point of interest is to study the effect of parallel assimilation on learning and convergence speeds of LASSP in practical learning problems. Another interesting problem is that of shadow learning. The problem is simply that of generalizing or specializing the premise of a given rule in the domain theory, guided by the database schema and empirical induction rules, rather than expert guidance. Hence shadow learning is practically costless. These generalizations and specializations might eliminate some negative examples or add some positive examples, hence resulting in a correction to the domain theory. Normal and shadow learning can be performed concurrently, moreover any errors resulting from shadow learning, for example if an overgeneralization adds negative examples to a rule's version space, should ultimately be trapped during subsequent normal learning situations.

References


