Learning Task Models for Collagen

Andrew Garland and **Neal Lesh** and **Charles Rich** and **Candace L. Sidner**
{garland,lesh,rich,sidner}@merl.com

A Mitsubishi Electric Research Laboratory (MERL)
201 Broadway, Cambridge MA, 02139, USA
Phone: 617-621-7500, Fax: 617-621-7550

Abstract

For an application-independent collaborative tool, a key step is to develop a detailed task model for a particular domain. This is a time consuming and difficult task, and seems to require a fairly advanced knowledge of AI representations for plans, goals, and recipes. This paper discusses some preliminary ideas for making it easier to construct and evolve task models, either through interaction with a human domain expert, through machine learning, or in a mixed-initiative system.

Introduction

An important trend in recent work on human-computer interaction and user modeling has been to view human-computer interaction as a kind of collaboration. In this approach, the human user and the computer, often personified as an "agent," coordinate their actions toward achieving shared goals. Collagen is an application-independent collaboration manager based on the SharedPlan theory of task-oriented collaborative discourse (Grosz & Sidner, 1990; Rich & Sidner, 1998; Lesh, Rich, & Sidner, 1999).

For an application-independent tool like Collagen, a key step in building a collaborative agent is to develop a detailed task model for a particular domain. This is a time consuming and difficult task that seems to require a fairly advanced knowledge of AI representations for plans, goals, and recipes. This paper discusses some preliminary ideas for making it easier to construct and evolve task models.

As with most learning problems, the intended usage of learned knowledge has as much impact on learning techniques as the characteristics of the underlying domain. In the case of Collagen, it is imperative that the collaborative agent be able to do more than complete tasks for the user or be able to autonomously generate sequences of primitive actions. To be an active participant in a *shared* problem-solving task, the agent must be able to make utterances (including suggestions) that the human finds both comprehensible and useful.

Collagen task models

Task models in Collagen are straightforward representations of actions, plans, and recipes. Actions are either primitive actions, which can be executed directly, or non-primitive (abstract) actions, which are achieved indirectly by achieving other actions. Non-primitive actions can be specified to be "top level", meaning that they might be done for no other purpose than themselves. Each action has associated parameters, results, and timestamp. Each action also has preconditions and effects, although currently there is no explicit representation of causal knowledge in our task model (the preconditions and effects are functions that indicate whether the action can be executed or has succeeded, respectively). Recipes are methods for decomposing non-primitive actions into subgoals. Each recipe describes a set of steps that can be performed to achieve a non-primitive action. There may be several different recipes for achieving a single action. The recipes contain constraints that may include temporal ordering between actions, as well as other logical relations among their parameters. Some of the steps of a recipe may be optional or repeatable.

The most substantial task models we have developed thus far is for the Lotus eSuite^{TM} email system. Based on empirical study of people working on email, Sidner and colleagues have formalized the task structure of this domain in terms of high-level goals, such as "working on email," lower-level goals, such as "filling in a message," and primitive actions corresponding to individual clicks on the eSuite^{TM} interface, such as "saving a message." Our task model for the email domain contains 31 recipes, 32 primitive actions, and 19 non-primitive actions. To control the complexity of the target library, we are using a simple cooking domain as a testbed. Figure 1 shows one non-primitive action and one recipe from that domain.

Currently, we produce our task models by typing them directly into a text editor. We could improve upon this approach by providing visualization tools that would allow us to see and directly edit a graphical representation of the information in the task model. Some of the information in the task model seems easier to visualize than others. For example, we could easily use

```
public act Boil {
   parameter Liquid liquid;
}

public recipe ARecipeName achieves PreparePasta {
   step Boil boilWater;
   optional step GetPasta getPastaIfNeeded;
   step MakePasta makeSomePasta;
   bindings {
      achieves.pastaWater == boilWater.liquid;
   }
   constraints {
      getPastaIfNeeded.item == makeSomePasta.pasta;
      boilWater.liquid == makeSomePasta.water;
      boilWater precedes makeSomePasta;
      getPastaIfNeeded precedes makeSomePasta;
   }
}
```

Figure 1: Sample Collagen representations for a simple cooking domain.

an "and/or" graph to represent subgoal relationships and directed arrows (of a particular color) to represent ordering relationships between actions in a recipe. However, it seems more difficult to visualize arbitrary constraints on parameters of an action. It seems that creating such tools for Collagen's task models would be a relatively straightforward, though not small, engineering task.

There are two basic ways in which we envisage task models for Collagen being acquired:

1. Programming-By-Demonstration. The development of visualization tools will support the construction and maintenance of task models by domain experts who are not experienced in AI or task modeling. However, programming-by-demonstration techniques are an appealing alternative to asking a domain expert to use visualization tools alone. The appeal lies in the expectation that domain experts will find it easier to perform tasks (and then discuss the performance) than to construct an abstract task model from scratch.

   We imagine that the human expert will perform simple tasks on the application to be modeled. The performed tasks might be the same ones that one would use to demonstrate the application to another human. Afterwards, the expert will annotate a log of her actions so that the computer can extract a task model from the annotated logs. The actions in the log could be annotated, for example, to indicate whether an action was optional or repeatable.

   Furthermore, the expert would segment, or hierarchically cluster, the action log such that actions that contribute to a single purpose will be put in the same segment. Each segment would correspond, roughly, to a single recipe in the task model. (Although there may be several demonstrations of each recipe.)

   Figure 2 is an example of a possible segmentation and annotation for the actions that someone might use to demonstrate how to use a web site for finding movies playing in local theaters. The actions are labeled A1 through A6, the indentation indicates the segmentation, and the segment labels and the annotations are in square brackets.

2. Machine learning. As described in the next section, there are many pieces of information needed in order to develop a task model that takes full advantage of the expressiveness of the Collagen representation. While programming-by-demonstration techniques will hopefully make the knowledge elicitation task seem more natural to a domain expert, they will not reduce the number of annotations needed. Machine learning techniques provide a means of reducing the annotation effort of a domain expert by inferring pieces of information instead of requiring the expert to explicitly specify them.

   This paper is concerned with two possible automated approaches to learning pieces of information helpful in developing task models. One approach is to incrementally construct a more accurate task model from a series of examples. In essence, general knowledge about the relationships that exist between actions and recipes can be exploited to allow a learning engine to infer annotations a domain expert may have made. Another approach is statistical in nature. We imagine that the computer might have access to a (potentially very large) log of unannotated user activity of the application. These logs could be analyzed by statistical inference techniques to detect relationships that are important to constructing a task model, such as segments of related actions or optional recipe steps. The last section of this paper discusses how these machine learning techniques and programming-by-demonstration techniques can be combined in a mixed-initiative learning

```
[Segment: GET-MOVIE-SCHEDULE]
  (A1) Go to MovieFinder site
  [Segment: SELECT-MOVIE]
    (A2)  Fill in director slot with Woody Allen
    (A3)  Press GO button
    (A4)  Select "Sweet and Low Down"
  [Segment: SELECT-THEATER]
    (A5) Select "Kendall Theater"
  (A6) Print page  [optional]
```

Figure 2: Sample segmentation and annotation.

system.

Learning issues

The previous section presented an overview of the knowledge contained in a Collagen task model. This section discusses in detail the pieces of information needed to construct a task model that takes full advantage of the expressiveness of the Collagen representation. For each piece of information, the trade-offs between explicitly eliciting that information from a domain expert (through annotations) and using machine learning techniques are discussed.

The first three pieces of information outlined — segmentations, action names, and recipe names — can be considered the bare minimum amount of information needed to develop a Collagen task model. They are essential in order to support an agent capable of making utterances (including suggestions) that a human finds useful for two reasons. First, the names given to non-primitives (and the names of the recipes that achieve them) will be an essential piece of the utterance, be it text or voice, presented to the user by the collaborative agent in many conversations. Second, the agent should make utterances at a time that seems "natural" to the user; in many cases, this entails top-down problem-solving: hierarchically decomposing problems into subproblems. Therefore, a linear representation for a task model, even one containing generalized actions (Bauer, 1998, 1999), is insufficient.

Segmentations The decomposition of an example trace into segments of related actions is essential in order to learn a task model containing hierarchical problem-solving representations. Each segment corresponds to a non-primitive action. Also, the primitive actions and sub-segments of a segment comprise the steps in a recipe to achieve that non-primitive action.

Segmenting an execution trace would not necessarily be difficult for the human domain expert who generated a trace given the proper visualization tools. At a conceptual level, identifying segments can be viewed as a grammar induction problem (if one focuses on just the primitive action types, i.e., ignores particular parameter values). Therefore, statistical inference techniques will be able to generate compact representations that we believe will be reasonable estimates for segmentations.

Non-primitive action names Each segment identified, generally consisting of several actions and/or segments, accomplishes a non-primitive action. It is important that the purpose of each segment be labeled consistently so that the learning engine can determine whether two segments represent two different ways to achieve the same non-primitive action or whether they represent the achievement of two different non-primitive actions. Techniques to automatically generate segmentations could presumably also generate consistent labelings to be used as surrogates for action names (albeit semantically meaningless ones).

Recipe names The particular "steps" (i.e. set of primitive actions and sub-segments) that make up a segment can be considered a recipe for achieving the corresponding non-primitive action. If the segment is annotated with a recipe name, the learning engine can determine whether the seen segment requires generalizing the current model of that recipe developed from previous examples. (This could entail, for example, marking a step that was believed to be required as optional or by dropping previously believed constraints that are violated by the current example.)

A good heuristic for allowing a learning engine to consistently assign the same name to a given recipe is to presume that recipes that accomplish a given purpose are composed of unique, disjoint sets of required step types. In that case, recipe names can be generated such as

```
PreparePastaRecipeVia_Boil_MakePasta
```

where step types are separated by an underscore and types for optional steps (such as GetPasta) are omitted. Step types are sorted in the recipe name to remove dependence on the order of seen steps. When the optionality of steps is learned, rather than presented as an annotation, the presumption allows for easily tracking and updating recipe names to reflect changing beliefs about the optionality of steps. An arbitrary number of steps of the same step type are accommodated by this heuristic.

Recipe step names Each recipe logically decomposes into steps. However, in general, there is no guarantee that a seen step of a given type (such as clicking GO) in the given example instance refers to a step of the same type in the developing model for the recipe. For example, it could be the case that the "true" recipe contains two steps of the given type, both of which are optional. In order for the learning engine to be able to make logically correct deductions about a step in a recipe, it needs a way to identify the seen step with the step in the developing model.

Human domain experts are unlikely to find assigning step names to individual steps palatable. A good heuristic for allowing a learning engine to "guess" at a step name is to presume that each recipe contains only a single step of a given type and that if multiple steps have the same type, then the step must be repeatable. In this case, a string representation of the step type can be used as the step name.[1]

Recipe step optionality or repeatability Marking steps as optional or repeatable on a consistent basis simplifies the reasoning needed by a learning engine. Further, such boolean characteristics could be clearly presented and easily changed (e.g., with mouse-event driven pull-down menu) in a good visualization tool.

Identifying optional steps can also be easily inferred by a learning engine, given the heuristics already mentioned for inferring recipe and step names. When comparing the current example of a recipe to the model of the recipe developed from previous examples, any steps in the model that are missing in the example can be presumed to be optional as can any steps in the example that are not part of the model.

For the inferences to be provably correct in the worst case would require seeing $O(n)$ examples of a recipe, where n is the number of instantiations of the recipe that are possible (to assure that n is finite, we can assume that repeatable steps are never repeated more than a fixed constant N number of times). However, the best case is $O(1)$ and we can presume that since the domain expert is interested in minimizing their own work, we are more likely to be faced with the best case rather than the worst.

Recipe constraints Although Collagen supports arbitrary structural relationships between parameters and results, recipes typically exploit two kinds of constraints. The first is partial ordering information between steps and the other is equality of action parameters and results. Annotating examples to denote these relationships will vary greatly in difficulty depending on the visualization tool, the amount of relationships that exist in a domain, and the level of comfort the domain expert has with thinking about such relationships.

Ordering and equality constraints can be inferred by tracking potential relationships from individual examples and removing potential relationships when they are contradicted by a new example. As with optionality, the worst case to prove correctness is $O(n)$ in theory, but is likely to be close to $O(1)$ in practice.

Non-primitive action parameters Non-primitive actions need parameters for two reasons. One is semantic: if the agent is to suggest that (the non-primitive) action Boil be achieved, it is sensible for the suggestion to include the particular liquid under consideration. The second reason is technical: non-primitive parameters allow recipes to specify relationships between steps regardless of which recipe is used to accomplish the step.

Given the somewhat abstract nature of non-primitive parameters, it is likely that domain experts will find it difficult to determine the number and type of them. However, it is also a difficult inference problem. For any given recipe tree, it is not difficult to postulate the existence and type of parameters for the non-primitive actions. However, parameters should be common to all recipe trees that achieve the action. And, like identifying recipe steps, matching postulated parameters between different recipe trees can only be guaranteed to be correct if done by the user.

There is no heuristic to infer non-primitive action parameters that will be reasonable in all domains. One possible heuristic requires making several assumptions about the task model. The first is to assume that the type of a non-primitive action's parameter will be exactly the same as the parameter of the primitive action whose value is being propagated in the recipe tree. Secondly, each non-primitive action must be assumed to have at most one parameter (and one result) of a given type. Finally, for a non-primitive action parameter of a given type to be inferred to exist, *all* recipes that achieve the non-primitive must postulate the existence of exactly one parameter of that type. Even in domains where such strong assumptions are not warranted, it may be practical to adopt this heuristic nonetheless, since non-primitive parameters are likely to be the least intuitive concept for a domain expert.

Mixed-initiative Learning

The previous sections discussed the types of annotations that might be explicitly specified by a domain expert or estimated by machine learning techniques. However, we expect that neither method alone will be fruitful; rather, a mixed initiative manner of learning a task model will be needed.

The minimum amount of participation for a domain expert would be in the case when machine learning techniques are capable of determining a "correct" recipe library for a given domain from unannotated usage logs. In that situation, the domain expert must only pro-

[1] If the temporal ordering of the steps can be guaranteed to be known, then any number of steps of the same type can be handled by appending a counter to the string representation of the step type.

Additional annotations provided by the oracle	Number of examples needed			
	Avg.	Dev.	Min.	Max.
All	2.77	1.406	1	7
None	6.95	3.063	3	20
Ordering constraints	5.42	2.301	2	14

Table 1: The kind of annotations provided influences the number of examples needed to learn a simple task model.

vide semantically meaningful names (or give meaningful glosses to meaningless names) for non-primitives and recipes. However, such a minimal role for the domain expert is unlikely. In general, we imagine that the human will be additionally required to supervise the relationships identified by the learning engine.

There are three principle ways in which the knowledge of the domain expert and the output of a learning engine could be interleaved for learning task models. One of these is for the learning engine to make statistical inferences from the unannotated usage logs before the domain expert begins the learning session. The results of this *a priori* analysis will be presented to the domain expert as a first-pass suggested annotation. A possible downside to such an approach might be that the first-pass suggestions differ from the domain experts beliefs, thus requiring the human to do more work (parsing the suggestion and then restoring the interface to a *tabula rasa* from which to work).

A second way in which the learning system and the domain expert may interact is that the learning system can seek verification from the user for various inductive hypothesis (Angros, 2000). For example, when a demonstrated segment with a novel set of steps subsumes the set of steps for a known recipe, the heuristic in the previous section will conclude that the current demonstration is an instance of the known recipe. In mixed-initiative mode, this conclusion can be presented to the domain expert for confirmation. Along the same lines, the computer can search through the unannotated logs to find interesting cases to ask the human expert about.

The third, and most elegant, way for the learning engine to support the domain expert is to present the expert with a first-pass analysis of a new example trace based on the current state of the task model. Since the task model will contain information elicited from the expert while annotating previous examples, this should be a superior first pass to one based solely on statistical inference.

Preliminary Results

We have partially implemented the machine learning techniques described in this work in a proof-of-concept system. For the results presented here, only heuristics for incrementally constructing a more accurate task model from a series of examples have been implemented; no statistical techniques have been implemented.

The system learns from a collection of example traces (each trace contains only primitive actions) generated beforehand from a known task model. The known task model is then used by an "oracle" procedure, thereby eliminating the need for a domain expert to make any annotations. The learning engine does not have access to the known task model; it is presented with a series of annotated examples just as is if a human were present. The oracle also halts the learning algorithm when the developing task model is equivalent to the known model.[2] The advantage of this approach is that by removing the human from the learning process, many more experiments can be conducted.

The task model under consideration is for a simple cooking domain and contains only four non-primitive actions and four recipes. Because of optional steps and partial ordering, there are 24 different sequences of primitive action types that can be generated from this library. Including the possible parameter values of the primitive actions, there are 288 possible example traces that can be generated. A subset of 100 unique traces was randomly selected to be the test collection.

In order to test the efficacy of the inference techniques, the system was run four times. Each run of the system corresponds to a different amount of annotations provided by a domain expert (in our case, by the oracle). The annotations provided for a given example are restricted to the relationships that exist among the primitive actions in that example. For example, if an optional step is not present in an example, no information whatsoever about that step (including its absence) is part of the annotation. The values for each run were averaged over 100 sequences of examples drawn (in random order) from the test collection.

Segmentations and non-primitive action names are always provided by the oracle, but the additional pieces of information described earlier in this paper may or may not be provided. The measurements in Table 1 show that this task model requires 2.77 examples (on average) to re-learn when the examples are fully annotated (labeled "All"). In contrast, 6.95 examples are needed when no additional annotations are given ("None"). The last line in the table shows that additional providing only ordering constraints means that 5.42 examples are needed.

[2]Without such guidance, the learning engine continues processing examples until either no more examples are present or the developing model has not changed for many (defined by a preset threshold) examples in a row.

References

Angros, Jr., R. 2000. *Learning What to Instruct: Acquiring Knowledge from Demonstrations and and Focussed Experimentation.* Ph.D. Dissertation, Department of Computer Science, University of Southern California, Los Angeles, CA.

Bauer, M. 1998. Towards the Automatic Acquisition of Plan Libraries. In *Proc. of the 13th European Conf. on Artificial Intelligence*, 484–488.

Bauer, M. 1999. From Interaction Data to Plan Libraries: A Clustering Approach. In *Proc. of the 16th Intl. Joint Conf. on Artificial Intelligence.*

Grosz, B., and Sidner, C. 1990. Plans for discourse. In Cohen, P. R.; Morgan, J.; and Pollack, M. E., eds., *Intentions in Communication.* Cambridge, MA: MIT Press. 417–444.

Lesh, N.; Rich, C.; and Sidner, C. 1999. Using plan recognition in human-computer collaboration. In *Proc. of the 7th Intl. Conf. on User Modeling*, 23–32.

Rich, C., and Sidner, C. 1998. COLLAGEN: A collaboration manager for software interface agents. *User Modeling and User-Adapted Interaction* 8(3/4):315–350.