Generating Artificial Corpora for Plan Recognition

Nate Blaylock[1] and James Allen[2]

[1] Saarland University, Saarbrücken, Germany,
blaylock@coli.uni-sb.de,
[2] University of Rochester, Rochester, New York, USA
james@cs.rochester.edu

Abstract. Corpora for training plan recognizers are scarce and difficult to gather from humans. However, corpora could be a boon to plan recognition research, providing a platform to train and test individual recognizers, as well as allow different recognizers to be compared. We present a novel method for generating artificial corpora for plan recognition. The method uses a modified AI planner and Monte-Carlo sampling to generate action sequences labeled with their goal and plan. This general method can be ported to allow the automatic generation of corpora for different domains.

1 Introduction

Over the past 10+ years, many fields in AI have started to employ corpus-based machine learning techniques. Plan recognition, however, seems to have lagged behind. For example, we are only aware of a few plan recognizers [1–4] (the last two are our own) that are trained on corpora. We believe a major reason for this is the lack of appropriate corpora for plan recognition (which we will term *plan corpora*).

It is not that the field could not make use of plan corpora. Besides the machine-learning based systems mentioned above, many plan recognizers ([5–8], inter alia) make use of probabilities, but only briefly mention (if at all) how such probabilities could be discovered.[3] Additionally, corpora could be used to evaluate the performance of a plan recognizer, or even compare performance across recognizers (something which, as far as we are aware, has never been done).

In this paper we present a general method for automatically generating labeled plan corpora. In Section 2, we present possible ways of getting plan corpora from human sources and discuss their disadvantages. Then in Section 3 we introduce our method for artificially generating corpora and show an example in Section 4. We then discuss some general issues in Section 5. Finally, in Section 6, we discuss related work and in Section 7, we conclude and mention future work.

[3] A notable exception is [9].

2 Human Sources of Plan Corpora

In this section, we mention several plausible ways of gathering plan corpora by observing humans. These can be divided into the kind of data that they make available: unlabeled, goal labeled, and plan labeled data. We discuss each in turn and then discuss the general difficulties of gathering human data.

2.1 Unlabeled Data

There are several techniques used in related fields for gathering unlabeled data, which could be useful for plan recognition.

Several projects in ubiquitous computing [10, 11] have gathered raw data of a user's state over time (location and speed from GPS data) which they use to predict user activity. Plan recognizers, however, typically take action streams as input.

Davison and Hirsh [12] collected a corpus of over 168,000 Unix commands by observing 77 users over a period of 2-6 months. The corpus consists of timestamped sequences of commands (stripped of arguments) as automatically recorded by the history mechanism of `tcsh`. It is unclear how useful such unlabeled data would be by itself for plan recognition (although Bauer [13] has done work on using such data to automatically construct recipe libraries).

2.2 Goal-labeled Data

Much more useful to plan recognition are goal-labeled plan corpora, although such corpora are even harder to come by.

Albrecht et al. [2] extract a plan corpus from the logs of a Multi-User Dungeon (MUD) game. A log includes a sequence of both player location (within the game) as well as each command executed. In addition, the MUD records each successful quest completion, which can be used to automatically tag plan sessions with a top-level goal (as well as partial state with the user's location). Albrecht et al. report that the corpus data is quite noisy: first because of player errors and typos, and also because players in MUDs often interleave social interaction and other activities. We should also note that the goals in the corpus are atomic, as opposed to being parameterized goal schemas.

More tightly-controlled goal-labeled corpora have been gathered through data collection efforts in the Unix [14] and Linux [4] domains. In these experiments, test subjects are given a specific goal, such as "find a file that ends in .tex", and their shell commands are recorded as they try to accomplish the goal. The subjects then report when they have successfully accomplished the goal (as there is no way to easily compute this automatically).

In these controlled experiments, goal labeling is much more reliable because it is assigned a priori. Of course, this work can still be noisy, as when the subject misunderstands the goal, or incorrectly believes he has accomplished it. Also, this kind of data collection is expensive as compared to those mentioned above. The above-mentioned data collections monitor the normal activity of subjects,

whereas these types of collections require subjects to work on tasks specifically for the collection.

2.3 Plan-labeled Data

Of course, the most useful type of plan corpus would include not only the top-level goal, but also the plan and situation.

Bauer [1] records user action sequences (and corresponding system state) in an email program and uses a plan recognizer post hoc to label them with the appropriate goal and plan. This post hoc recognition can potentially be much more accurate than online prediction, because it is able to look at the whole execution sequence. A potential problem we see with this approach is that if the original plan recognizer consistently makes mistakes in predicting plans, these mistakes will be propagated in the corpus. This includes cases where the plan library does not cover extra or erroneous user actions.

2.4 General Challenges for Human Plan Corpora

In addition to the individual disadvantages mentioned above, we see several shortcomings to this kind of human data collection for plan recognition.

First, this kind of data collection is most feasible in domains (like operating systems) where user actions can be directly observed and automatically recorded. This, unfortunately, excludes most non-software interaction domains. In fact, the only way we can envision to gather data for other domains would be to have it annotated by hand, which could be expensive and time-consuming (not to mention error-prone).

Finally, a major shortcoming of the above work is that it is at most labeled with a top-level goal.[4] In most domains where plan recognition is used (e.g., natural language understanding), the system can benefit not only from the prediction of a top-level goal, but also partial results where a subgoal is predicted. This is especially true of domains with plans composed of large action sequences, where the top-level goal may not become apparent until very far into the plan's execution. We imagine that manual annotation of plan labeling would be quite tedious and error prone.

3 Artificial Corpus Generation

In contrast to human data collection, we propose the use of an AI planner and Monte-Carlo simulation to stochastically generate *artificial* plan corpora. This method can be used for any domain and provides a corpus accurately labeled with goal and hierarchical plan structure. It also provides a cheap way to produce the kind of large corpora needed for machine learning. The method is as follows:

[4] Except for [1], although, as we mention above, the corpus can be skewed by the original recognizer's mistakes.

1. We modify an AI planner to search for valid plans non-deterministically.
2. We model the desired domain for the planner.
3. The algorithm does the following to generate each item in the corpus:
   (a) Stochastically generates a goal
   (b) Stochastically generates a start state
   (c) Uses the planner to find a valid plan for generated goal and start state

We first describe our modifications to an AI planner. Then we discuss issues of domain modeling. We then discuss stochastic generation of the goal and then of the start state. Finally, we discuss the characteristics of corpora generated by this process.

3.1 Planner Modification

For plan recognition, we want to create corpora which allow for all possible plans in the domain. Typical AI planners do not support this, as they usually deterministically return the same plan for a given goal and start state. Many planners also try to optimize some plan property (like length or cost) and therefore would never output longer, less optimal plans. We want to include all possible plans in our corpus to give us broad coverage.

We, therefore, modified the SHOP2 planner [15] to randomly generate one of the set of all possible plans for a given goal and start state.[5] We did this by identifying key decisions points in the planner and randomizing the order that they were searched.

SHOP2 [15] is a sound and complete hierarchical transition network (HTN) planner. SHOP2 is novel in that it generates plan steps in the order they will be executed, which allows it to handle complex reasoning capabilities like axiomatic inference and calls to external programs. It also allows partially ordered subtasks. The planning model in SHOP2 consists of *methods* (decomposable goals), *operators* (atomic actions), and *axioms* (facts about the state).

In searching the state space, there are three types of applicable decisions points, which represent branches in the search space:[6]

- Which (sub)goal to work on next
- Which method to use for a goal
- Which value to bind to a parameter

In order to provide for completeness, SHOP2 keeps lists of all possibilities for a decision point so that it may backtrack if necessary. We modified the planner so that these lists are randomized after they are populated but before they are

[5] In principle, the corpus generation technique described here is possible using any planner. The only caveat is that the planner must be randomized, which may or may not be a straightforward thing to do. One of the reasons we chose SHOP2 was its small code base and a modular design that was amenable to randomization.

[6] There is also a fourth which deals with `:immediate` tasks, but that is beyond the scope of this paper.

used. This one-time randomization guarantees that we search in a random order but also allows us to preserve the soundness and completeness of the algorithm. We believe the randomized version is equivalent to computing all valid plans and randomly choosing one.

3.2 Domain Modeling

Each new domain must be modeled for the planner, just as it would if the intent were to use the planner for its usual purpose. As opposed to modeling for plan generation, however, care should be taken to model the domain such that it can encompass all anticipated user plans.

Usually the planning model must be written by hand, although work has been done on (semi-)automating the process (e.g., [13]). Note that, in addition to the model of the plan library, which is also used in many plan recognizers, it is also necessary to model state information for the planner.

3.3 Goal Generation

We separate goal generation into two steps: generating the goal schema and generating parameter values for the schema.

Goal Schema Generation In addition to the domain model for the planner, the domain modeler needs to provide a list of possible top-level goals in the domain, together with their a priori probability. A priori probabilities of goals are usually not known, but they could be estimated by the domain modeler's intuitions (or perhaps by a small human corpus). The algorithm uses this list to stochastically picks one of the goal schemas.

Goal Parameter Value Generation In domains where goals are modeled with parameters, the values of the parameters must also be generated.

Goal parameter values can be generated by using one of two techniques. For goal schemas where the parameter values are more or less independent, the domain modeler can give a list of possible parameter values for each slot, along with their a priori probabilities. For schemas where parameter values are not independent, each possible set of parameter is given, along with their probabilities.

Once the goal schema has been chosen, the algorithm uses this lists to stochastically generate values for each parameter in the schema. At this point, a fully-instantiated goal has been generated.

3.4 Start State Generation

In addition to a top-level goal, planners also need to know the state of the world — the start state. In order to model agent behavior correctly, we need to

stochastically generate start states, as this can have a big effect on the plan an agent chooses.

Generating the start state is not as straightforward as goal generation for several reasons. First, in all but the simplest domains, it will not be feasible to enumerate all possible start states (let alone to assign them a priori probabilities). Second, in order to make the planning fast, we need to generate a start state from which the generated goal is achievable. Practically, most planners (including SHOP2) are **very slow** when given an impossible goal, as they must search through all of the search space before they notice that the goal is impossible.

For these reasons, only a start state which makes the generated goal achievable should be generated. Unfortunately, we know of no general way of doing this.[7] We do believe, however, that some general techniques can be used for start state generation. We discuss these here. The approach we have chosen is to separate the state model into two parts: fixed and variable. In the *fixed* part, we represent all facts about the state that should be constant across sessions. This includes such things as fixed properties of objects and fixed facts about the state (for example, the existence of certain objects, the location of cities, and so on).

The *variable* part of the state contains those facts which should be stochastically generated. Even with the fixed/variable separation, this part will probably not be a set of independent stochastically generated facts. Instead, the domain modeler must come up with code to do this, taking into account, among other things, domain objects, their attributes, and other states of the world. It is likely that values of sets of facts will need to be decided simultaneously, especially in cases where they are mutually exclusive, or one implies another, etc. This will also likely need to be closely linked to the actual goal which has been generated to ensure achievability.

3.5 The Resulting Corpus

A corpus generated by the process described above will contain a complex distribution of plan sessions. This distribution results from the interaction between (a) the a priori probabilities of top-level goals, (b) the probabilities of top-level goal parameter values, (c) the algorithm for generating start states, and (d) information encoded in the plan library itself. Thus, although it cannot be used to compute the a priori probabilities of top-level goals and parameter values (which are given as input to the generator), it can be used to e.g., model the probabilities of subgoals and atomic actions in the domain. This is information which cannot be learned directly from the plan library, since the recipes and variable fillers used are also dependent on e.g., the start state.

[7] One possibility might be backchaining from the goal state, although we have not explored this.

4 An Example: The Emergency Response Domain

We have created a domain model in an emergency response domain and used it to generate an artificial corpus. The domain includes such goals as setting up a temporary shelter and providing medical attention to victims. The coded domain consists of 10 top-level goal schemas, 46 methods and 30 operators. The plan library coded in a fairly common way and does not merit any further discussion here. For the rest of this section we discuss the generation of goals and start states in order to illustrate what may be needed in moving to a new domain (in addition to the creation of a plan library).

4.1 Goal and Start State Generation

As mentioned above, the domain includes 10 goal schemas which are specially marked as top-level goals (the difference is not specified in SHOP2 itself). In addition, we added a priori probabilities to each of the goal schemas.

The goal schema was chosen based on those probabilities as discussed above. The schema is then passed to a function which generates the parameter values and the start state simultaneously. In particular, we start with the fixed start state, then stochastically generate locations for movable objects, and then generate other domain facts based on goal schema specific code. We mention these in order here.

Fixed State The fixed state consists mostly of fixed locations (such as towns and hospitals), objects and their properties. It also includes inference rules supported in SHOP2 which represent things like object types and properties (e.g., adult(x) $\Rightarrow$ can-drive(x)).

Object Locations As part of the variable state, we define a set of *movable* objects. They are movable in the sense that we wanted to randomly choose where they were located (such as ambulances and workers). We define a list of *sets* of objects, for which it is not important *where* they are located, but only that all objects in the set are in the same location (such as a vehicle and its driver). We also define a list of possible locations, which is used to generate a random location for each object set. (Note, we ensure in the fixed state that locations are fully connected so we don't have to worry about goal impossibility at this step.)

Goal Schema Specific The rest of the state is created, together with parameter values, in goal schema specific functions. In the emergency domain these were typically very simple, usually just determining which object to use for parameter values.

An example of a more complicated example is that of the goal schema of clearing a road wreck, which takes a wrecked car as a parameter. As we do

not model the set of all possible cars in the world, we automatically generate a unique car object as well as its necessary properties (e.g., that it's wrecked, its location, etc.) Note that in cases where extra properties are generated, these are also stochastically generated from a priori probabilities (e.g., whether or not the roads are snowy).

5 Discussion

In this section, we raise several issues about the utility of artificial generation of plan corpora versus the collection of human plan corpora. As we have just begun to generate and use such corpora, we do not believe we are in a position to definitively answer these. Rather, we raise the questions and give some initial thoughts, which we hope can lead to a discussion in the plan recognition community. The questions treat three general areas: the effort needed to generate artificial corpora; the accuracy of such corpora; and the general power of the technique.

Effort Obviously, the technique we describe above requires a certain amount of work. Minimally, one needs to create a plan library as well as an algorithm for generating start states. Plan library creation is known to be difficult and is a problem for the planning community in general (cf. [13]). This may not be a unique problem to artificial corpora, however, as a plan library would likely be necessary anyway in hand-labeling human corpora. Start state generation is also not trivial, although in our experience, it was much less work than the building the plan library.

The main question which needs to be answered here is how the effort to create the machinery for generating an artificial plan corpus compares to the effort needed to gather and annotate a human corpus. Before we can answer this, we not only need more experience in generating artificial corpora, but also experience in producing human corpora - especially plan-labeled corpora.

Accuracy Another point is how accurately an artificial corpus can model human behavior. Ideally, to test this, one would want to gather a human corpus and independently generate an artificial corpus in the same domain and then make some sort of comparison. Of course, care must be taken here, as we suspect that the accuracy of an artificial corpus will be highly-dependent on the plan library as well as the algorithm for generating start states. Another, more practical, evaluation would be the comparison of the performance of a plan recognizer on human data when it has been trained on artificial data versus human data.

Power Another question is in which situations an artificial corpus could be successfully used to approximate human behavior. The technique presented here makes the simplifying assumption (which is also present in most plan recognizers) that the agent first creates an entire plan and then executes it, and that each action is successfully executed. This obviously will not work well in domains

where this is not the case. In future work, we would like to adapt this technique to use an artificial agent instead of a planner, to plan and simulate execution of the plan in creating a corpus. This would allow us to simulate such phenomena as action failure, replanning, and so forth. In general, we believe that the techniques reported here can build on existing work in agents in modeling human behavior and can be useful in most domains of interest in plan recognition.

6 Related Work

Conceptually, our work is based on work in NLP which uses grammars to stochastically generate artificial corpora for training language models for speech recognition [16]. Of course, there are many differences in methodology. Surface string generation from a stochastic grammar typically assumes no context (state), whereas state is very important in plan recognition. Also, in surface string generation, there is no "goal" which restricts acceptable output.

Probably the closest work to our own in the plan recognition field was done by Lesh [14], who uses the Toast reactive planer [17] to generate action sequences given a goal. However, none of the generation process was stochastic. It appears that goals were hand-generated, the state was constant, and the planner was not modified to make decisions non-deterministically, meaning that it would always produce the same action sequence given the same set of goals.

7 Conclusions and Future Work

We have presented a novel technique for generating corpora for plan recognizers. We combine the rich representation of an AI planner with Monte-Carlo sampling to generate corpora of action sequences tagged with goal and plan. Also, as it is artificially generated, it is easy to produce a very large corpus.

In future work, we want to move beyond just plans, and model an actual agent. We believe this would allow us to more closely model agents that we would want to perform plan recognition on, and would include phenomena such as plan failure and replanning. This corpus generation method would allow us to have access to this additional information (when an action failed, when replanning occurs, etc.), which would not be readily available from hand-annotated human data.

Acknowledgments

We would like to thank the anonymous reviewers for their comments and especially bringing up some of the issues discussed in Section 5.

This material is based upon work supported by a grant from DARPA under grant number F30602-98-2-0133; two grants from the National Science Foundation under grant number IIS-0328811 and grant number E1A-0080124; and the TALK project. Any opinions, findings, and conclusions or recommendations

expressed in this material are those of the authors and do not necessarily reflect the views of the above-mentioned organizations.

References

1. Bauer, M.: Acquisition of user preferences for plan recognition. In: Proceedings of the Fifth International Conference on User Modeling, Kailua-Kona, Hawaii (1996)
2. Albrecht, D.W., Zukerman, I., Nicholson, A.E.: Bayesian models for keyhole plan recognition in an adventure game. User Modeling and User-Adapted Interaction **8** (1998) 5–47
3. Blaylock, N., Allen, J.: Corpus-based, statistical goal recognition. In: IJCAI, Acapulco, Mexico (2003)
4. Blaylock, N., Allen, J.: Statistical goal parameter recognition. In: ICAPS, Whistler, British Columbia (2004)
5. Charniak, E., Goldman, R.P.: A Bayesian model of plan recognition. Artificial Intelligence **64** (1993) 53–79
6. Huber, M.J., Durfee, E.H., Wellman, M.P.: The automated mapping of plans for plan recognition. In de Mantaras, R.L., Poole, D., eds.: UAI94 - Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence, Seattle, Washington, Morgan Kaufmann (1994) 344–351
7. Pynadath, D.V., Wellman, M.P.: Accounting for context in plan recognition, with application to traffic monitoring. In: Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence, Montreal, Canada, Morgan Kaufmann (1995)
8. Bui, H.H., Venkatesh, S., West, G.: Policy recognition in the Abstract Hidden Markov Model. Journal of Artificial Intelligence Research **17** (2002) 451–499
9. Bauer, M.: Quantitative modeling of user preferences for plan recognition. In: UM, Hyannis, Massachusetts (1994)
10. Ashbrook, D., Starner, T.: Using GPS to learn significant locations and predict movement across multiple users. Personal and Ubiquitous Computing **7** (2003)
11. Patterson, D.J., Liao, L., Fox, D., Kautz, H.: Inferring high-level behavior from low-level sensors. In: UBICOMP. (2003)
12. Davison, B.D., Hirsh, H.: Predicting sequences of user actions. In: Notes of the AAAI/ICML 1998 Workshop on Predicting the Future: AI Approaches to Time-Series Analysis, Madison, Wisconsin (1998)
13. Bauer, M.: Acquisition of abstract plan descriptions for plan recognition. In: Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98), Madison, WI (1998) 936–941
14. Lesh, N.: Scalable and Adaptive Goal Recognition. PhD thesis, University of Washington (1998)
15. Nau, D., Au, T.C., Ilghami, O., Kuter, U., Murdock, J.W., Wu, D., Yaman, F.: SHOP2: An HTN planning system. Journal of Artificial Intelligence Research **20** (2003) 379–404
16. Kellner, A.: Initial language models for spoken dialogue systems. In: Proceedings of ICASSP'98, Seattle, Washington (1998)
17. Agre, P., Horswill, I.: Cultural support for improvisation. In: Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI). (1992)