Learning User Plan Preferences Obfuscated by Feasibility Constraints

Nan Li, William Cushing, Subbarao Kambhampati*, and Sungwook Yoon
School of Computing and Informatics
Arizona State University
Tempe, Arizona 85281 USA
{nan.li.3,wcushing,rao,sungwook.yoon}@asu.edu

Abstract

It has long been recognized that users can have complex preferences on plans. Non-intrusive learning of such preferences by observing the plans executed by the user is an attractive idea. Unfortunately, the executed plans are often not a true representation of user preferences, as they result from the interaction between user preferences and feasibility constraints. In the travel planning scenario, a user whose true preference is to travel by a plane may well be frequently observed traveling by car because of feasibility constraints (perhaps the user is a poor graduate student). In this work, we describe a novel method for learning true user preferences obfuscated by such feasibility constraints. Our base learner induces probabilistic hierarchical task networks (pHTNs) from sets of training plans. Our approach is to rescale the input so that it represents the user's preference distribution on plans rather than the observed distribution on plans.

Introduction

It is well known that users may have complex preferences on the plans they would like to execute. While taking preferences into consideration during plan generation has been an active area of research (Baier and McIlraith 2008), learning a user's planning preferences is an emerging area of research. Since interactive elicitation of user preferences can be time-consuming and error-prone, the work in this area has focused on passive learning by observing the plans executed by the user. One challenge here is that executed plans are not an accurate reflection of the user's true preferences, but rather preferences modulated by feasibility constraints. For example, a (poor) graduate student that prefers to travel by plane may instead be frequently observed traveling by car. In other words, by observing the plans executed by the user we can relatively easily learn what the user *usually does*,[1] and so can predict their behavior as long as feasibility constraints remain the same. It is a much trickier matter to infer what the user truly *prefers to do*, and it is this piece of knowledge that would allow predicting what the user will do in a novel situation.

*Kambhampati's research is supported in part by the ONR grant N00014-09-1-0017 and the NSF grant IIS-0905672.

[1]A useful piece of knowledge in the plan recognition scenario (Geib and Steedman 2007).

The objective of this paper is to explore ways in which we can learn true user preferences despite obfuscation from feasibility constraints. In general, this problem is impossible to solve completely, for many reasons. The user's preferences could change over time, they may very well be inconsistent (i.e. contradictory), the user may not have realized all of the options available to them (and so we observe them choosing a less preferred plan), the user's preferences could depend on variables we cannot observe, our observations might be very noisy, and so forth. Nonetheless we pursue the goal: an automatic system, operating without elicitation, that could guess user's preferences approximately correctly much of the time would be quite useful.

Towards this end, we describe a novel, but intuitive, approach for rescaling/reweighting observed plans in order to undo the filtering caused by feasibility constraints. We build upon our recent work in learning preferences on plans (Li, Kambhampati, and Yoon 2009). That learner induces a "grammar", in the form of a probabilistic hierarchical task network (pHTN), in order to reproduce its input distribution on plans (represented as a (weighted) set). In this work, then, we build on that approach by altering the relative frequencies of observed plans so that the distribution approximates the user's preferred distribution on plans. The idea is to automatically generate alternatives (e.g. by an automated planner) to the user's observed behavior, and assume that the observed behavior is preferred to the alternatives. This gives a large set of pairwise preferences, which we then transitively close (roughly speaking). So, then, the system is capable of taking an informed guess at which of two plans, previously never possible together, would be preferred in a novel situation (where both are possible).

First we discuss our algorithm for rescaling training data. Then, as we have not (yet) connected an automated planner, we design a random distribution on solutions, modeling feasibility constraints, in order to evaluate the rescaling technique.

Learning Preferences

We assume that we can directly observe a user's behavior, for example by building upon the work in *plan recognition*, and that we have access to the set of alternative solution plans a user could have employed in that situation, for example by building upon the work in *automated planning*.

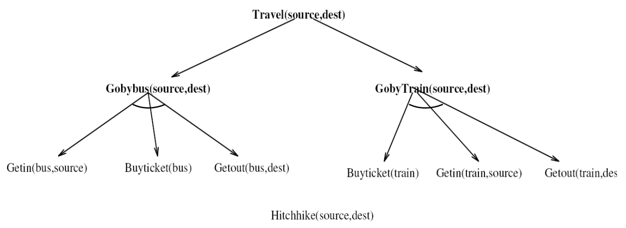

Figure 1: Preferences of a potential user.

Table 1: An example (learned) pHTN.

$Travel \rightarrow 0.2,\ A_2\ B_1$	$Travel \rightarrow 0.8,\ A_1\ B_2$
$B_1 \rightarrow 1.0,\ A_1\ A_3$	$B_2 \rightarrow 1.0,\ A_2\ A_3$
$A_1 \rightarrow 1.0,\ Buyticket$	$A_2 \rightarrow 1.0,\ Getin$
$A_3 \rightarrow 1.0,\ Getout$	

So each training episode consists of an observed solution along with a set of alternative solutions.[2] We will use an abstract domain modeling traveling for running examples ("the Travel domain"). Figure 1 depicts the qualitative preferences, as a hierarchical task network, of one possible user in this domain — a user that loathes hitchhiking.

Input. The input of the learning system is a list of records, $R = r_1, r_2, \ldots, r_m$. Each record, r, consists of a set of (solution) plans, $S(r) : \{s_1, s_2, \ldots, s_n\}$, and the observed plan, $O(r) \in S(r)$.

Since each input record is only capable of expressing preferences among feasible plans (the selected plan is preferredto the alternatives), user preferences among some plans may have never been revealed by the input. For example, consider a user that prefers traveling by car rather than by hitchhiking, and prefers traveling by plane rather than by train. Then there is no information about whether cars are preferred to planes, or trains to hitchhiking, and so forth. So there are three possible answers to "Is A preferred to B?": *yes*, *no*, and *unknown*.

We learn such partial orders on plans (partially known preferences) by learning a set of total orders on subsets of all plans.Suppose we have two knowledge sources H_1 and H_2 such that plans A and B belong to the domain of H_1 and plans C and D belong to the domain of H_2. Then we can answer a query on A and B, or on C and D, by consulting H_1 or H_2 respectively, but a query on A and C is *unknown*, because C is not in the domain of H_1 and A is not in the domain of H_2. Note this abstract example models the preceding concrete example. We learn partially known preferences by posing several learning problems to the base learner rather than a single learning problem.

Output of rescaling, Input of learning. The result of rescaling is a list of weighted plan clusters, $C = c_1, c_2, \ldots, c_k$. Each cluster, c, consists of a set of plans and associated weights; we write $p \in c$ for membership and $w_c(p)$ for the associated weight.

[2]Or if you prefer, each training episode consists of a planning problem and an observed solution, from which a set of (alternative) solutions could be inferred prior to (or during) rescaling.

pHTN. The base learner (Li, Kambhampati, and Yoon 2009) exploits the connection between context-free grammars (CFGs) and (simple) hierarchical task networks (SHTNs (Nau et al. 1999)), borrowing a probabilistic grammar induction technique (Lari and Young 1990) and applying it to planning in order to learn probabilistic (simple) hierarchical task networks (pHTNs). We will use parsing and task reduction terminology interchangeably. Table 1 demonstrates a learned pHTN for the Travel domain.[3]

Given a pHTN H and a plan p we can efficiently compute a) the most probable parse of p by H and b) the (a priori) likelihood of a given parse tree. We will write $l_H(p)$ for the likelihood of the most probable parse of p (and abbreviate with $l(p)$ when H is clear in context). We say p is preferred to q (by H) if $l(p) > l(q)$. Similarly we say p is (definitely) not preferred to q if $l(p) < l(q)$. Finally, if either p or q fails to be parsable (by H) then we say that the preference between p and q is unknown.

Output of learning. The result of learning is a list of pHTNs, $\mathcal{H} = H_1, H_2, \ldots, H_k$, each "grammar" approximating the input weighted plan cluster of the same index.

In the following we give the details of the rescaling step and refer the reader to our previous work for the details of the learning step (Li, Kambhampati, and Yoon 2009).

Clustering. First we collapse all of the input records from the same or similar situations into single weighted clusters, with one count going towards each instance of an observed plan participating in the collapse. For example, suppose we observe 3 instances of *Gobyplane* chosen in preference to *Gobytrain* and 1 instance of the reverse. Then we will end up with a cluster with weights 3 and 1 for *Gobyplane* and *Gobytrain* respectively. In other words $w_c(p)$ is the number of times p was chosen by the user in the set of situations collapsing to c (or ϵ if p was never chosen). This happens in lines 2–20 of Algorithm 1.

Transitive Closure. Next we make indirect inferences between clusters; this happens by iteratively merging clusters with non-empty intersections. Consider two clusters, c and d, in the Travel domain. Say d contains *Gobyplane* and *Gobytrain* with counts 3 and 1 respectively, and c contains *Gobytrain* and *Gobybike* with counts 5 and 1 respectively. From this we infer that *Gobyplane* would be executed 15 times more frequently than *Gobybike* in a situation where all 3 plans (*Gobyplane*, *Gobytrain*, and *Gobybike*) are possible, since it is executed 3 times more frequently than *Gobytrain* which is in turn executed 5 times more frequently than *Gobybike*. We represent this inference by scaling one of the clusters so that the shared plan has the same weight, and then take the union. In the example, supposing we merge d into c, then we scale d so that $c \cap d = \{\text{Gobytrain}\}$ has the same weight in both c and d, i.e. we scale d by

[3]The nonprimitives, except the top level, lack descriptive names because the grammar is *automatically* learned from only (weighted) primitive action sequences; there are so many because it is easier to process grammars in Chomsky normal form. Note that we optimize pHTNs for computer, not human, readability.

Algorithm 1: Rescaling

```
   Input: Training records $R$.
   Output: Clusters $C$.
1  initialize $C$ to empty
2  forall $r \in R$ do
3      if $\exists c \in C$ such that $S(r) \subseteq c$ or $S(r) \supseteq c$ then
4          forall $p \in S(r) \setminus c$ do
5              add $p$ to $c$ with $w_c(p) := \epsilon$
6          end
7          if $w_c(O(r)) \geq 1$ then
8              increment $w_c(O(r))$
9          else
10             $w_c(O(r)) := 1$
11         end
12     else
13         initialize $c$ to empty
14         add $c$ to $C$
15         forall $p \in S(r)$ do
16             add $p$ to $c$ with $w_c(p) := \epsilon$
17         end
18         $w_c(O(r)) := 1$
19     end
20 end
21 while $\exists c, d \in C$ such that $c \cap d \neq \emptyset$ do
22     sum_ratios := $0$
23     forall $p \in c \cap d$ do
24         sum_ratios += $w_c(p)/w_d(p)$
25     end
26     scale := sum_ratios$/|c \cap d|$
27     forall $p \in d \setminus c$ do
28         add $p$ to $c$ with $w_c(p) := w_d(p) *$ scale
29     end
30     remove $d$ from $C$
31 end
32 return $C$
```

$5 = w_c$(Gobytrain)$/w_d$(Gobytrain). For pairs of clusters with more than one shared plan we scale $d \setminus c$ by the average of $w_c(\cdot)/w_d(\cdot)$ for each plan in the intersection, and leave the weights of $c \cap d$ as in c. Computing the scaling factor happens in lines 21–26 and the entire merging process happens in lines 21–31 of Algorithm 1.

Learning. Finally we learn a grammar for each of the remaining clusters in C. This is done with the method in (Li, Kambhampati, and Yoon 2009). Summarizing, there are 2 major steps. First a greedy analysis proposes a sufficient variety of possible method reductions so that each of the input action sequences can be parsed. Then an expectation-maximization method is run to set the probabilities across each potential reduction of a method, aiming at finding the probabilistic task network that best fits the input distribution (but it is a local search).

Ultimately we arrive at a list of pHTNs ($\mathcal{H} = H_1, \ldots, H_k$), each capturing the user's preferences in one cluster of similar situations. Plans that can be generated by the same pHTN (with non-zero probability) are comparable in that pHTN, based on the likelihood of the most probable parse for each, and are otherwise incomparable (by that pHTN). If some pair of plans is comparable in several pHTNs (due to the generalization caused by learning a "grammar", as the input clusters were disjoint) then we take a simple majority vote (without tie-breaking).

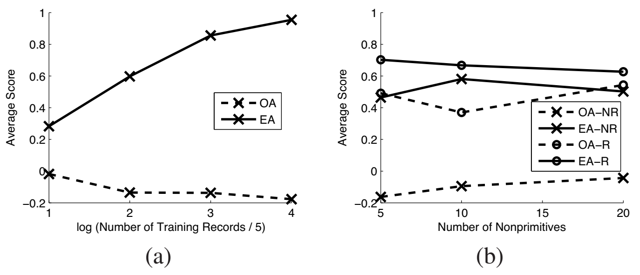

Figure 2: Experimental results for random H^*. "EA" is learning with rescaling and "OA" is learning without rescaling. (a) Learning rate. (b) Size dependence: "R" for recursive H^* and "NR" for non-recursive H^*.

Evaluation

We are primarily interested in evaluating the rescaling extension of the learning technique, i.e., the ability to learn preferences despite constraints. We design a simple experiment to demonstrate that learning purely from observations is easily confounded by constraints placed in the way of user preferences, and that our rescaling technique is able to recover preference knowledge despite obfuscation.

Setup

Performance. We take an oracle-based experimental strategy, that is, we imagine a user with a particular ideal pHTN, H^*, representing that user's preferences, and then test the efficacy of the learner at recovering knowledge of preferences based on observations of the imaginary user. After training the learner produces $\mathcal{H}$; to evaluate the effectiveness of $\mathcal{H}$ we pick random plan pairs and ask both H^* and $\mathcal{H}$ to pick the preferred plan. There are three possibilities: $\mathcal{H}$ agrees with H^* (+1 point), $\mathcal{H}$ disagrees with H^* (-1 point), and $\mathcal{H}$ declines to choose (0 points)[4].

The distribution on testing plans is not uniform and will be described below. The number of plan pairs used for testing is scaled by the size of H^*; $100t$ pairs are generated, where t is the number of nonprimitives. The final performance for one instance of the "game" is the average number of points earned per testing pair. Pure guessing, then, would get (in the long-term) 0 performance.

User. We carry out two kinds of experiments, distinguished by the distribution on imaginary users. In the first kind we randomly generate pHTNs. These experiments are further divided by generating either *non-recursive* or *recursive* pHTNs. In the second kind we design a particular pHTN by hand, modeling our own preferences in a few benchmark domains.

Training Data. For both randomly generated and handcrafted users we use the same framework for generating

[4]This gives rescaling a potentially significant advantage, as learning alone always chooses. We also tested scoring "no choice" at -1 point; the results did not (significantly) differ.

training data. To generate random problems we instead generate random solution sets (as there is no domain theory or planner in the system (yet)). That is, we *model* feasibility constraints using a particular random distribution on solution sets. We begin by constructing a list of plans, $\mathcal{P}$, from $100t$ samples of H^*, removing duplicates (so $|\mathcal{P}| \leq 100t$). Due to duplicate removal, less preferred plans occur later than more preferred plans (on average). We reverse that order, and associate $\mathcal{P}$ with (a discrete approximation to) a power-law distribution. Both training and test plans are drawn from this distribution. Then, for each training record r, we take a random number[5] of samples from $\mathcal{P}$ as $S(r)$. We pick the observed plan, $O(r)$, from $S(r)$ based on H^*.

Note that the random solution sets model the "worst case" of feasibility constraints, in the sense that it is the least preferred plans that are most often feasible — much of the time the imaginary user is forced to pick the least bad of a set of all bad options (relatively speaking).

Baseline. The baseline for our experiments will be our prior work, that is, the learning component without rescaling. We generate a single cluster with $w(p) = |\{i : p = O(r_i)\}|$ (the weight of a plan is the number of times it is observed), and run the original learning algorithm on that, producing a single pHTN: $\mathcal{H} = H_1$.

All of the experiments were run on a 2.13GHz Windows PC with 1.98GB of RAM. The cpu time spent was less than 4 milliseconds per record; i.e. overall learning time was always small. So the main point of interest is the quality of the learned knowledge, that is, the performance of $\mathcal{H}$ in the described "game".

Random H^*

Rate of Learning. In order to test the learning rate, we first measured performance for H^* with a fixed number (5) of nonprimitives, varying the number of training records, averaging over 100 samples of H^*. The results are shown in Figure 2(a). We can see that with a large number of training records, rescaling before learning is able to capture nearly full user preferences, whereas learning alone performs slightly worse than random chance. This is expected since without rescaling the learning is attempting to reproduce its input distribution, which was the distribution on observed plans — and "feasibility" is inversely related to preference by construction. That is, given the question "Is A preferred to B?" the learning alone approach instead answers the question "Is A executed more often than B?".

Size Dependence. We also tested the performance of the two approaches under varying number of nonprimitives (using $50t$ training records); the results are shown in Figure 2(b). For technical reasons, explored in (Li, Kambhampati, and Yoon 2009), the base learner is much more effective at recovering user preferences when these take the form of recursive schemas, so there is less room for improvement. Nonetheless the rescaling approach outperforms learning alone in both experiments.

[5]The number of samples taken is selected from $|\mathcal{P}| \cdot |N(0,1)|/2$, subject to minimum 2 and maximum $|\mathcal{P}|$.

Hand-crafted H^*

In the second set of experiments, we designed pHTNs modeling our preferred solutions in the Logistics and Gold Miner domains. At the level of abstraction used the experimental setup described generates sets of "feasible" plans that are, in fact, sets of feasible solutions to *some* problem.

Logistics Planning. The domain we used in the first experiment is a variant of the Logistics domain. There are 4 primitive actions, *load*, *fly*, *drive* and *unload* in this domain, and 11 nonprimitives in our pHTN (H^*) for this domain. We presented 550 training records to both learning systems. Learning alone only scored 0.342, whereas rescaling before learning performed significantly better with a score of 0.847.

Gold Miner: The second domain we used is Gold Miner. There are 5 primitive actions *move*, *getLaserCannon*, *shoot*, *getBomb* and *getGold*. Our strategy for this domain is: *1) get the laser cannon, 2) shoot the rock until reaching the cell next to the gold, 3) get a bomb, 4) use the bomb to get gold.* Our encoding of this strategy in a pHTN uses 12 nonprimitives. We trained both approaches with 600 examples. Learning alone performed reasonably well with a score 0.605, still, rescaling before learning performed even better with a score of 0.706.

The preference in Logistics for nonrecursive reductions over recursive reductions of various nonprimitives is much sharper than in Gold Miner, so the qualitative difference in performance between the two domains could be understood as the same qualitative difference in the experiments with non-recursive and recursive random schemas (Figure 2(b)).

Conclusion

In this work we explored acquiring hierarchical user preferences from user behavior obfuscated by feasibility constraints. We take the approach of adjusting the observed frequencies of plans to fit the user's (inferred) preferred distribution rather than the observed, executed, distribution on plans. We evaluated this approach with random and hand-crafted preferences against a "worst-case" model of feasibility constraints, and showed that rescaling the observed distribution before learning was significantly more effective than learning only off of observed behavior (Li, Kambhampati, and Yoon 2009).

References

Baier, J. A., and McIlraith, S. A. 2008. Planning with preferences. In *AI Magazine*.

Geib, C. W., and Steedman, M. 2007. On natural language processing and plan recognition. In *IJCAI*.

Lari, K., and Young, S. J. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*.

Li, N.; Kambhampati, S.; and Yoon, S. 2009. Learning probabilistic hierarchical task networks to capture user preferences. In *IJCAI*.

Nau, D. S.; Cao, Y.; Lotem, A.; and Hector Mu n.-A. 1999. Shop: Simple hierarchical ordered planner. In *IJCAI*.