Learning Probabilistic Hierarchical Task Networks as Probabilistic Context-Free Grammars to Capture User Preferences

NAN LI
Computer Science Department
Carnegie Mellon University, Pittsburgh, PA 15213 USA
and
WILLIAM CUSHING
SUBBARAO KAMBHAMPATI
SUNGWOOK YOON
Department of Computer Science
Arizona State University, Tempe, AZ 85281 USA

We introduce an algorithm to automatically learn probabilistic Hierarchical Task Networks (pHTNs) that capture a user's preferences on plans by observing only the user's behavior. HTNs are a common choice of representation for a variety of purposes in planning, including work on learning in planning. Our contributions are two-fold. First, in contrast with prior work, which employs HTNs to represent domain physics or search control knowledge, we use HTNs to model user preferences. Second, while most prior work on HTN learning requires additional information (e.g., annotated traces or tasks) to assist the learning process, our system only takes plan traces as input. Initially we will assume that users carry out preferred plans more frequently, and thus the observed distribution of plans is an accurate representation of user preference. We then generalize to the situation where feasibility constraints frequently prevent the execution of preferred plans. Taking the prevalent perspective of viewing HTNs as grammars over primitive actions, we adapt an Expectation-Maximization (EM) technique from the discipline of probabilistic grammar induction to acquire probabilistic context-free grammars (pCFG) that capture the distribution on plans. To account for the difference between the distributions of possible and preferred plans, we subsequently modify this core EM technique, by rescaling its input. We empirically demonstrate that the proposed approaches are able to learn HTNs representing user preferences.

Categories and Subject Descriptors: 2.11 [**AI Technology**]: Planning; 2.16 [**AI Technology**]: Hierarchical Task Networks learning

Additional Key Words and Phrases: Hierarchical Task Networks, learning user preferences, planning

1. INTRODUCTION

Application of learning techniques to planning is an area of long standing research interest [Zimmerman and Kambhampati 2003]. Most work (e.g., [Ilghami et al. 2002; Yang et al. 2005; Langley and Choi 2006; Yang et al. 2007; Hogg et al. 2008]) in this area to-date has, however, only considered learning domain physics or search control. Knowledge acquired by these algorithms helps planners to generate *feasible* plans, either with greater reliability or less computation. A relatively neglected alternative application of learning, and the focus of this work, is to produce higher *quality* plans: we apply automated learning to acquire users' *preferences* concerning plans.

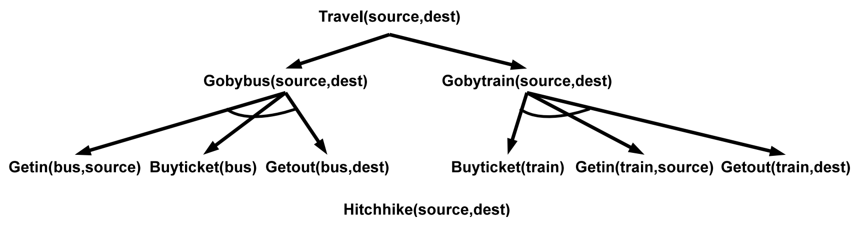


Fig. 1. Hierarchical task networks in a travel domain.

It has long been understood that, in practice, (1) the framework of classical planning [Fikes and Nilsson 1971] is not up to the task of representing complex constraints on whole plans [Nau et al. 2003], and (2) that user preferences can indeed be quite complex in this regard [Baier and McIlraith 2008]. Hierarchical Task Networks (HTNs) are, among other choices, an effective and popular means of representing such complex constraints on plans.

We view task networks as (context-free) *grammars* [Kambhampati et al. 1998; Geib and Steedman 2007], saying that a plan is *parseable* if it satisfies the constraints represented by an HTN (and *feasible* if it satisfies the remainder of the constraints, i.e., the classical planning constraints). So, to begin with, we can represent absolute preferences by considering a plan to be *valid* only if it is both feasible and parseable. That is, unparseable yet feasible plans are held to be so loathed by the user at hand that it would be better to fail than to offer the alternative [Kambhampati et al. 1998]. To represent *degree* of preference we immediately generalize our chosen representation language to Probabilistic Hierarchical Task Networks (pHTNs).

Elaborating, consider Figure 1. It depicts the preferences, modeled as an HTN, of a hypothetical user for the Travel domain. So, for this user, both *Gobytrain* and *Gobybus* are acceptable reductions of the *Travel* task. In contrast, the plan of hitch-hiking (modeled as a single action), while executable and goal achieving, is not considered valid — the user in question loathes that mode of travel to the point of giving up rather than hitch-hiking. For some other user we might very well have included an arrow directly from the *Travel* task to hitchhiking. Such 'boolean' preferences are themselves interesting, but clearly it is desirable to accommodate degrees of preference. To do so we attach probabilities to the methods that reduce tasks into subtasks, equating "probable" with "preferred", arriving at probabilistic Hierarchical Task Networks (pHTNs).

Note that one can employ HTNs to encode complex executability constraints as well. This is the view taken when learning domain physics in the form of HTNs. Indeed, one need not even attach any meaning to the primitives of a domain at all, instead representing all planning constraints by parseability of the given network. Likewise one can very well go about representing all constraints purely in the form of preconditions and effects. We do not seek a homogenous representation of all constraints. We envisage separate representations for simple physical constraints as preconditions and effects of primitives, for simple quality constraints as goals of achievement, for complex physical constraints as task reductions of a HTN (or set of logical assertions in LTL, Golog, . . .), and, the topic of this paper, for complex quality constraints as probabilistic task reductions of a pHTN.

To automatically learn pHTNs, then, we view them as mere Probabilistic Context-Free Grammars (pCFGs) and try to 'simply' apply the proven techniques for probabilistic grammar induction [Collins 1997; Charniak 2000; Lari and Young 1990]. Doing so requires significant compromises, since HTNs, by default, are a significantly richer formalism than CFGs. Moreover, the application areas of planning and natural language processing are quite distinct: the mapping between plans as sequences of actions and sentences as sequences of words (etc.) is only sound at a first pass. Then the work here undertaken is to adapt the techniques of (probabilistic) grammar induction to account for the differences between the appropriate assumptions for natural language processing and the assumptions appropriate for planning. We make two contributions:

(1) We eliminate the need for structurally annotated input.
(2) We learn the user's desired, rather than typical, behavior.

The following two subsections elaborate on the two issues considered and our approach to resolving them.

(Lack of) Known Structure: The first order of business is to overcome the assumption from natural language processing that significant information is available concerning the correct structure of the grammar. Since the point of language is to communicate mental constructs, it is indeed reasonable to suppose that significant information concerning the 'right' grammar nonprimitives could be obtained prior to learning. That is, the 'right' non-primitives (*verb*, *noun*, *noun phrase*, ...) are common to many users. So specifically algorithms for part-of-speech tagging (for example) are acceptable preprocessing steps in grammar induction. In contrast, preference is personal: the correct non-primitives are *not* common to an overwhelming majority of users.

We develop a Structure Hypothesizer (SH) to invent (out of thin air) the 'concepts' needed by the subsequent application of the specific Expectation-Maximization (EM) approach we borrow from the literature on probabilistic grammar induction [Lari and Young 1990]. We empirically demonstrate that, despite the lack of correctness guarantees of SH, the final performance is nonetheless promising. We further validate our approach by comparing it to the well known inside-outside algorithm for grammar induction, and demonstrating the superiority of our approach [Lari and Young 1990].

Feasibility Constraints: The second order of business is to correct for the assumption that the point of learning is to reproduce the input distribution [Li et al. 2009]. For, say, supporting expectation-based parsing (analogously: plan recognition [Geib and Steedman 2007]), one does indeed wish to learn the typical distribution of utterances. But, for preferences, the desired output of learning is the distribution over what the user *wants* to do (i.e., 'say'), rather than the distribution encoding the user's typical behavior (i.e., typical 'utterances'). As a first pass we use this insight merely to motivate the choice of performance metric in our empirical evaluations.

The second part of the paper considers in greater depth the issue that feasibility constraints can prevent users from engaging in preferred behavior, and, thus, also obscure our ability to infer preferences by simple observation of behavior. To resolve the issue, we permit ourselves to 'observe' the alternatives that the user might have hypothetically executed (in addition to the actual choice of behavior). For example, in (computer-aided) travel planning systems (e.g., Orbitz), we can indeed directly observe possible flight reservations shown to, but not chosen, by the user.[1] More generally we propose the use of *diverse (automated) planning* to fill in any gaps of knowledge concerning plausible alternatives

[1] This example is provided by one of the anonymous reviewers.

Table I. A probabilistic Hierarchical Task Network in Chomsky Normal Form.

Primitives: Buyticket, Getin, Getout, Hitchhike;		
Tasks: Travel, A_1, A_2, A_3, B_1, B_2;		
Travel $\rightarrow A_2\ B_1, 0.2$	Travel $\rightarrow A_1\ B_2, 0.8$	
$B_1 \rightarrow A_1\ A_3, 1.0$	$B_2 \rightarrow A_2\ A_3, 1.0$	
$A_1 \rightarrow$ Buyticket, 1.0	$A_2 \rightarrow$ Getin, 1.0	$A_3 \rightarrow$ Getout, 1.0

to observed behavior. In any case, we implement a simple *rescaling* approach that uses this additional information to rescale the input distribution to the learner so as to better reflect desire rather than behavior, and empirically demonstrate its effectiveness [Li et al. 2009]. More accurately, the approach consists of *clustering*, *transitive closure*, and, finally, *rescaling*.

In the following sections, we start by formally stating the problem of learning probabilistic Hierarchical Task Networks (pHTNs). Next, we present an algorithm that acquires pHTNs by observing only plan traces. The algorithm modifies the techniques of probabilistic grammar induction by performing structure-learning prior to the subsequent application of an expectation-maximization (EM) approach to parameter-learning. We compare the proposed approach with a straightforward application of pCFG learning method by evaluating the acquired pHTNs against idealized models of users. The experimental results show that the learner outperforms the inside-outside algorithm, and at least in the ideal case, can in fact capture its input distribution given a reasonable number of samples. Subsequently we consider possible obfuscation of user's preferences due to feasibility constraints. The approach is to rescale the input so that the initially described learner will behave as desired without further modification. We again conduct an empirical evaluation for idealized models of users, by considering 'worst-case' models of feasibility constraints, thereby giving the technique maximum room for improving performance. Finally we discuss related as well as future work, and summarize our contributions.

2. DEFINITIONS

In this section we formally define the problem of learning user preferences using *Probabilistic Hierarchical Task Networks* (pHTNs) as the model. These are exactly equivalent in form, but not in application, to Probabilistic Context Free Grammars.

Table I provides an example of a candidate pHTN potentially modeling the preferences of some hypothetical user in the Travel domain (see Figure 1). The example emphasizes the precise nature of the problem faced by the pHTN learner we develop. In particular, unlike the case of learning pCFGs for natural language, the internal non-primitives have no externally ascribed interpretation.

We begin with several supporting definitions.

HTN: A *Hierarchical Task Network* (HTN) $\mathcal{H} = \langle \mathcal{A}, \mathcal{T}, \mathcal{M} \rangle$ consists of a set of *actions* $\mathcal{A}$ (the primitives), a set of *tasks* $\mathcal{T}$ (the non-primitives), and a set of *methods* $\mathcal{M}$ (the rules). We follow the, notably restricted, SHOP notion of HTNs [Nau et al. 2003]. So each *method* $m \in \mathcal{M}$ is an $(\ell + 1)$-tuple, $\langle Z, t_1, t_2, \ldots, t_\ell \rangle$, also written $Z \rightarrow t_1 \ldots t_\ell$, specifying how the task $Z \in \mathcal{T}$ may be *performed* by sequentially performing each t_i. (In terms of production rules, one rewrites Z into the right-hand side of the chosen method.) Denote all the available methods for performing a task Z as $\mathcal{M}(Z) = \{m \in \mathcal{M} \mid m = \langle Z, \ldots \rangle\}$.

Chomsky Normal Form: Without loss of generality,[2] we immediately restrict our attention to *Chomsky Normal Form*: each method decomposes a task into either two tasks or one primitive. So for any method m, either $m = \langle Z, X, Y \rangle$ (also written $Z \to XY$), with $X, Y \in \mathcal{T}$, or $m = \langle Z, a \rangle$ (also written $Z \to a$) with $a \in \mathcal{A}$.

Actions: Classical Planning: We suppose that actions are defined as in classical planning [Fikes and Nilsson 1971]. Abstractly, a *planning domain* delinates a set of possible *states* and ascribes partial *(state) transition functions* to each of the actions (the primitives $\mathcal{A}$). For simplicity, identify actions with their partial transition functions. Then an action a is *executable* from s if $a(s)$ is defined. A *plan* $\phi = a_1, a_2, \ldots, a_n$ (an action sequence) is just defined by left-to-right partial function composition ($\phi(s) = a_n(a_{n-1}(\ldots a_1(s)\ldots))$); so, as with actions, ϕ is executable from s if $\phi(s)$ is defined. A plan ϕ *achieves* G from s if ϕ is executable from s and $\phi(s) \in G$ is true, equivalently: $\phi(s) \in G$ is defined and true. A *planning problem* is given by its *initial state* and *goal* (semantically a set of states). A *feasible* plan achieves the goal from the initial state.

Tasks: HTN Planning: *HTN planning* is an extension of classical planning by some HTN $\mathcal{H} = \langle \mathcal{A}, \mathcal{T}, \mathcal{M} \rangle$ and top-level task $T \in \mathcal{T}$. Plans remain as action sequences. An action sequence is a *solution* if it is both feasible and *parseable* to the task T by the methods $\mathcal{M}$. In short, a parse just records generating a plan from the top-level task according to the given methods.

Formally: A *parse (tree)* $\mathcal{X} = (V, E)$ of an action sequence $\phi = a_1, \ldots, a_n$ by the methods $\mathcal{M}$ to the task T is an ordered labeled binary tree on n leaves with root r satisfying the following. Label each vertex $v \in V$ with task $T(v) \in \mathcal{T}$ and method $m(v) \in \mathcal{M}(T(v))$. Partition the vertices V into the ordered leaves $\mathcal{L} = \ell_1, \ell_2, \ldots, \ell_n$ and parents $\mathcal{P}$ (internal vertices): $V = \mathcal{L} \cup \mathcal{P}$. Then:

(1) $T(r) = T$,

(2) for each parent $p \in \mathcal{P}$, with ordered children x, y: $m(p) = T(p) \to T(x)T(y)$,

(3) for each leaf $\ell_i \in \mathcal{L}$: $m(\ell_i) = T(\ell_i) \to a_i$.

Now we come to the generalization of the formal model to probabilities.

pHTNs: A *probabilistic Hierarchical Task Network* (pHTN) $\mathcal{H} = \langle \mathcal{A}, \mathcal{T}, \mathcal{M}, \theta \rangle$ is an extension of the underlying HTN $\langle \mathcal{A}, \mathcal{T}, \mathcal{M} \rangle$ by an assignment of *conditional probabilities* θ, a.k.a. *parameters*, to the methods $\mathcal{M}$ such that, for each task $Z \in \mathcal{T}$:

$$P(m \in \mathcal{M}(Z) \mid Z) = \theta(m), \qquad \text{i.e.,} \tag{1}$$

$$\sum \theta(\mathcal{M}(Z)) = 1. \tag{2}$$

Recall that $\mathcal{M}(Z)$ denotes all methods by which one might perform the task Z: pHTNs extend HTNs by specifying the prior distribution over each such choice. The parameters θ then also induce a prior distribution on action sequences as follows.

Prior Probability: Fix the top-level task T and the pHTN $\mathcal{H} = \langle \mathcal{A}, \mathcal{T}, \mathcal{M}, \theta \rangle$. Let $\mathcal{X}$ be any parse (to T by $\mathcal{M}$). Say $\mathcal{X}\{v\}$ is the sub-tree of $\mathcal{X}$ with root $v \in V(\mathcal{X})$. With x and y the ordered children of any parent $p \in \mathcal{P}(\mathcal{X})$, the prior probability of the sub-tree $\mathcal{X}\{p\}$,

[2]This may no longer be true when generalizing to probabilities.

given its root task, is, recursively:

$$P(\mathcal{X}\{p\} \mid T(p)) = \theta(m(p)) \cdot P(\mathcal{X}\{x\} \mid T(x)) \cdot P(\mathcal{X}\{y\} \mid T(y)). \tag{3}$$

The base cases, i.e., the trivial sub-trees on each leaf $\ell \in \mathcal{L}(\mathcal{X})$, are just:

$$P(\mathcal{X}\{\ell\} \mid T(\ell)) = \theta(m(\ell)). \tag{4}$$

So the prior probability of an entire parse tree is:

$$P(\mathcal{X}) = \prod_{v \in V(\mathcal{X})} \theta(m(v)). \tag{5}$$

For convenience say $m(V)$ is the multiset: $m(V) = \bigcup_{v \in V}\{m(v)\}$. Similiarly let $\theta(M)$ be the multiset $\theta(M) = \bigcup_{m \in M}\{\theta(m)\}$. Finally permit the abbreviation $\theta(\mathcal{X}) = \theta(m(V(\mathcal{X})))$. So $P(\mathcal{X}) = \prod \theta(\mathcal{X})$. Should T and $\mathcal{H} = \langle \mathcal{A}, \mathcal{T}, \mathcal{M}, \theta \rangle$ be not fixed then say explicitly: $P(\mathcal{X} \mid T, \mathcal{H}) = \prod \theta(\mathcal{X})$.

Then the prior probability of an action sequence is the sum of prior probabilities of every parse of that sequence (every complete parse is a disjoint event). Let $\mathbf{X}(\phi) = \{\mathcal{X} \mid \mathcal{X}$ is a parse of ϕ to T by $\mathcal{M}(\mathcal{H})\}$ be those parses. So:

$$\begin{aligned} P(\phi \mid T, \mathcal{H}) &= \sum_{\mathcal{X} \in \mathbf{X}(\phi)} P(\mathcal{X} \mid T, \mathcal{H}), \qquad (6)\\ &= \sum_{\mathcal{X} \in \mathbf{X}(\phi)} \prod \theta(\mathcal{X}). \qquad (7)\end{aligned}$$

This is the probability of generating ϕ by simply evaluating $\mathcal{H}$ top-down from T. Note that plans with non-zero priors are parseable (and thus are considered valid according to non-probabilistic HTNs, c.f. [Kambhampati et al. 1998]).

Most Probable Parse: A nice feature of Chomsky Normal Form is that the prior probability of a plan is straightforward to calculate exactly: there can only be finitely many possible parses of a fixed sequence of primitives.[3] Nonetheless, there may very well be exponentially many possible parses, which is not much better, in practice, than infinitely many possible parses. So in the remainder we will end up considering just the *most probable parse* of ϕ. With respect to some fixed pHTN $\mathcal{H}$ and top-level task T, define:

$$\mathcal{X}^*(\phi) = \text{argmax}_{\mathcal{X} \in \mathbf{X}(\phi)} \prod \theta(\mathcal{X}). \tag{8}$$

The (same) nice properties of Chomsky Normal Form are, in this case, actually exploitable: the most probable parse can be computed reasonably efficiently by the cubic-time dynamic programming algorithm described in the following [Li et al. 2009].

Posterior Probability: Now suppose that not all plans are possible. Say F is the set of feasible plans in some specific situtation, and write $[\phi \in F]$ to mean converting true to 1 and false to 0. Then the *posterior* distribution on plans given F is:

$$P(\phi \mid T, \mathcal{H}, F) = [\phi \in F] \cdot \frac{P(\phi \mid T, \mathcal{H})}{\sum_{\phi' \in F} P(\phi' \mid T, \mathcal{H})}. \tag{9}$$

[3]Conversely, that general pCFGs lack this property may be a loss of generality.

Plans with non-zero posterior probability are then feasible and parseable, so, such plans are solutions.

Quality: As a compromise, we use "probable" as a surrogate for "preferable". There are very interesting ways in which the connections between probability and preference are stronger than they may initially appear [Koller and Friedman 2009, page 15]. So, given a top-level task $T \in \mathcal{T}$ and pHTN $\mathcal{H} = \langle \mathcal{A}, \mathcal{T}, \mathcal{M}, \theta \rangle$, define the *quality* of a plan ϕ as just its prior probability $P(\phi \mid T, \mathcal{H})$. For example, if $\frac{P(A|T,\mathcal{H})}{P(B|T,\mathcal{H})} = 2$ we say the option A is twice as valuable/good as the option B — in the sense that we have observed the choice of A twice as often as we have observed the choice of B.

Note that solutions can be ranked by either posterior or prior probability: the two are proportional over solutions. It is ever so slightly better, though, to define quality as independent of feasibility as much as possible. For example,

$$\max_{\phi} P(\phi \mid T, \mathcal{H}) - \max_{\phi' \in F} P(\phi' \mid T, \mathcal{H})$$

is a measure of the gap between (our limited understanding of) the reality F and the user's personal utopia.

Learning pHTNs: Abstractly, the problem is: automatically learn a user-specific quality metric on plans based on pure observation of behavior. The preceding sets up the specific concrete interpretation we pursue, most importantly: take pHTNs as defining the notions of plan and quality. Formally: Fix the total number of task symbols k,[4] and the top-level task T. Given i.i.d. samples $\Phi = \{\phi_1, \phi_2, \ldots, \phi_n\}$ of observed action sequences, find the most likely Chomsky Normal Form pHTN on k task symbols, $\mathcal{H}^*$:

$$\mathcal{H}^* = \mathrm{argmax}_{\mathcal{H}} P(\mathcal{H} \mid \Phi, T). \tag{10}$$

The likelihood of a model, though, has not been defined. We employ the standard exploit of Bayes Rule to transform to the problem of maximizing the, defined, likelihood of the observation:

$$\begin{aligned}
\mathcal{H}^* &= \mathrm{argmax}_{\mathcal{H}} P(\mathcal{H} \mid \Phi, T), \\
&= \mathrm{argmax}_{\mathcal{H}} P(\Phi \mid T, \mathcal{H}) \cdot \frac{P(\mathcal{H} \mid T)}{P(\Phi \mid T)}, && \text{(Bayes Rule)} \\
&= \mathrm{argmax}_{\mathcal{H}} P(\Phi \mid T, \mathcal{H}), && \text{(assume a uniform prior)}^5 \\
&= \mathrm{argmax}_{\mathcal{H}} \prod_{\phi \in \Phi} P(\phi \mid T, \mathcal{H}). && (11)
\end{aligned}$$

[4]Actually, the learner, in its first phase, picks its own k, but not in any especially principled fashion. While ad-hoc, any automated guess is surely no worse than demanding human input ("fix k"): one is always free to simply override automated guesses. In any case, as the learner does not deeply consider the merits of varying k, the definition is accurate enough.

[5]Other choices than a uniform prior are possible. For example, one could generalize to varying k (in a principled fashion) by applying *information theory*, perhaps ending up with, say, $\mathcal{H}^* = \mathrm{argmax}_{\mathcal{H}} \log k \cdot P(\Phi \mid T, \mathcal{H})$.

3. LEARNING PHTNS FROM USER GENERATED PLANS

Our formalization of (probabilistic) Hierarchical Task Networks is isomorphic to formal definitions of (probabilistic) Context Free Grammars. This comes at a price: HTNs, even of just SHOP [Nau et al. 2003], are normally taken to be rather richer languages for representing constraints over plans. The advantage is that grammar induction techniques are more or less directly applicable. As far as the insight from pCFGs is concerned: For parameter-learning we adapt the Expectation-Maximization (EM) approach from [Lari and Young 1990].

However, despite formal equivalence, casting the problem as learning pHTNs (rather than pCFGs) does make a difference in what assumptions are appropriate. For example, we do not allow hints concerning non-primitives to be given in any form, and in particular we do not permit such hints as annotations on the primitives of observations. For language learning the non-primitives of interest are widely agreed upon: *noun*, *verb phrase*, and so forth. It is impossible to communicate without such agreement. In particular, information sources such as dictionaries and informal grammars can be mined relatively cheaply in order to provide useful annotations, as in part-of-speech tagging.

In contrast, in the case of preference learning for plans, the non-primitives of interest, preferences, are user-specific mental constructs. Then it is unreasonable to rely upon annotated observations: our system must invent non-primitives of its own accord. We develop a Structure Hypothesizer (SH) to, among other things, engage in such invention of non-primitives. Because of the manner in which it functions it could be easily modified to accept (certain kinds of) hints concerning non-primitives should such be available. By default though, it assumes only plain observations of behavior.

In the remainder of the section we describe the details of the full learner. It operates in two phases. First the Structure Hypothesizer (SH) considers the problem of inventing sufficiently rich structure (the tasks and methods) so as to allow the parsing of each observation to the top-level task. At its conclusion we have an HTN. In the second phase, the probabilities of the methods are set by a variant of the Expectation-Maximization (EM) approach from [Lari and Young 1990]. The final result is, naturally, a local optima in the space of pHTNs.

3.1 Structure Hypothesizer (SH)

We develop a Structure Hypothesizer (SH) in order to generate a set of methods that can, at least, parse all plan examples. More than that, it seeks to parse all the plan examples without resorting to various kinds of trivial grammars (for example, parsing each plan example with a disjoint set of methods). The basic idea is to iteratively factor out frequent common subsequences, in particular frequent common pairs since we work in Chomsky Normal Form. We describe the details in the following. Figure 2 summarizes the algorithm in pseudocode.

SH learns methods in a bottom-up fashion. It starts by initializing $\mathcal{H}$ with tasks, Z_a, each distinctly reducing to one of the primitives: $Z_a \rightarrow a$ for all $a \in \mathcal{A}$. Then all plan examples are rewritten, backwards (i.e., parsed), using this initial set of methods. All this amounts to is 'converting to upper case' — the initialization is minor notational fantasy to formally satisfy the requirements of Chomsky Normal Form.[6]

[6]Implementations can, instead, treat primitives as instances of tasks not permitted to appear on left-hand-sides.

Algorithm 1: SH(plan examples Φ, primitive actions $\mathcal{A}$) returns pHTN $\mathcal{H}$

```
1  $\mathcal{H} := \{Z_a \to a \mid a \in \mathcal{A}\}$ ;                    // primitive action schemas
2  $\Phi :=$ rewrite-plans($\Phi, \mathcal{H}$);
3  while not empty($\Phi$) do
4      case $|\phi :=$ shortest-plan($\Phi$)$| \leq 2$
5          if $|\phi| = 2$ then $\mathcal{H} := \mathcal{H} + (T \to \phi)$;
6          else $\mathcal{H} := \mathcal{H} \cup \{T \to \alpha \mid Z \to \alpha \in \mathcal{H}\}$          // $\phi = Z$ for some $Z$
7      case ($\langle Z, X, d\rangle :=$ best-simple-recursion($\Phi$)) is good enough
8          if $d =$ left then $\mathcal{H} := \mathcal{H} + (Z \to Z\ X)$;
9          if $d =$ right then $\mathcal{H} := \mathcal{H} + (Z \to X\ Z)$;
10     case otherwise
11         $(X, Y) :=$ most-frequent-pair($\Phi$);
12         $\mathcal{H} := \mathcal{H} + (Z_{XY} \to X\ Y)$;                   // $Z_{XY}$ is a new task
13     $\Phi :=$ rewrite-plans($\Phi, \mathcal{H}$);          // Plans rewritten to $T$ are removed
14 end
15 $\mathcal{H} :=$ initialize-probabilities($\mathcal{H}$);
16 return $\mathcal{H}$
```

Fig. 2. Pseudocode for the structure hypothesizer.

Next the algorithm enters its main loop: hypothesizing additional structure until all plan examples can be parsed to an instance of the top level task, T.[7] In short, SH hypothesizes a method, rewrites the plan examples using the new method as much as possible and repeats until done. At that point probabilities are initialized randomly, that is, by assigning uniformly distributed numbers to each method and normalizing by task (so that $\sum \theta(\mathcal{M}(Z)) = 1$) — the EM phase is responsible for fitting the probabilities to the observed distribution of plans.

In order to hypothesize a method, SH first searches for evidence of simple loops: subsequences of symbols in the form $\{SZ, SSZ, SSSZ\}$ or $\{ZS, ZSS, ZSSS\}$. Certainly patterns such as $ZABABAB$ also have looping structure; these are identified at a later stage. The frequency of such simple repetitions in the entire plan set is measured, as is their average length. If both meet minimum thresholds, then the appropriate method (e.g., $Z \to ZS$) is added to $\mathcal{H}$. Note that such loops already possess base cases due to the bottom-up strategy. This process corresponds to lines 7–9 in Figure 2. The thresholds themselves are functions of the average length and total number of plan examples in Φ. For example, if the average length of such repetitions is longer than 30% of the average length of plan examples, and the repetitions appear in more than 10% of the plan examples, the method is added.

If one or both thresholds are not met, then the frequency count of every pair of symbols is computed, and the maximum pair is added as a reduction from a distinct (i.e., new) task. This is done in lines 11–12 of Figure 2. In the prior example of a symbol sequence $ZABABAB$, eventually AB might win the frequency count, and be replaced with some new symbol, say S. After rewriting, the example sequence becomes $ZSSS$, lending evidence in future iterations, of the kind SH recognizes, to the existence of a simple loop (of the form $Z \to ZS$). If eventually that method is added then the example gets rewritten to just Z.

[7]The implementation though in fact allows the single rule $T \to Z$.

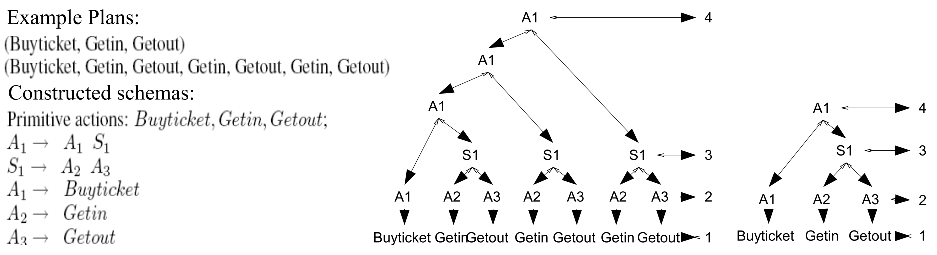

Example Plans:
(Buyticket, Getin, Getout)
(Buyticket, Getin, Getout, Getin, Getout, Getin, Getout)
Constructed schemas:
Primitive actions: *Buyticket*, *Getin*, *Getout*;
$A_1 \to A_1\ S_1$
$S_1 \to A_2\ A_3$
$A_1 \to$ *Buyticket*
$A_2 \to$ *Getin*
$A_3 \to$ *Getout*

Fig. 3. A trace of the Structure Hypothesizer on a variant of the Travel domain.

Example: Consider a variant of the Travel domain (Figure 1) allowing the traveler to purchase a day pass (instead of a single-trip ticket) for the train. Two training plans are shown in Figure 3. First SH rewrites the primitives (line 1): $A_1 \to$ Buyticket, $A_2 \to$ Getin, and $A_3 \to$ Getout. The updated plans are shown as level 2 in the figure. Next, since A_2A_3 is the most frequent pair in the plans (and there is insufficiently obvious evidence of looping), SH introduces the task and method $S_1 \to A_2A_3$ (line 11). After updating the plans (line 13), the plans become A_1S_1 and $A_1S_1S_1S_1$, depicted as level 3 in the figure. At this point SH perceives the loop (the simple repetition $A_1S_1S_1S_1$), and so adds the method $A_1 \to A_1S_1$ (line 8). After rewriting (line 13), all plans are parseable to the symbol A_1 (let $T = A_1$), so SH is done: the final structure is at the bottom left of Figure 3.

3.2 Setting Probabilities: Expectation-Maximization

In this section we describe the Expectation-Maximization (EM) approach we take to fitting appropriate parameters to the HTN returned by the preceding structure learning step (SH), thereby arriving at a pHTN.

In general, EM is a gradient-ascent method with two phases to each iteration: first the current model is used to compute 'Expected' values for the hidden variables (the E-step), and then the model parameters are updated to maximize the likelihood of those particular values for the hidden variables (the M-step). To show convergence it is more useful to characterize both steps as maximizing a single many-dimensional *loss function*, by holding disjoint subsets of dimensions fixed in each step. So 'Expected' actually means that one determines values for the hidden variables that maximize some loss function while holding model parameters constant. Then both steps are monotonically increasing the same function and convergence follows; of course global optimality does not follow.[8] The following gives the details of the E-steps and M-steps as applied to learning pHTNs.

Setup: Actually the structure learning step performs a trivial initialization of the probabilities (with no regard for the training data): say $\mathcal{H}_0 = \langle \mathcal{A}, \mathcal{T}, \mathcal{M}, \theta_0 \rangle$ is the initial pHTN. Later iterations update the pHTN, say $\mathcal{H}_t$ at iteration t, by updating the parameters θ_t.

E-step: In the E-step, the current parameters θ_t are used to compute the most probable parse tree $\mathcal{X}_t^*(\phi)$ of each example plan $\phi = a_1, \ldots, a_n \in \Phi$ (from the given start task T): $\mathcal{X}_t^*(\phi) = \operatorname{argmax}_{\mathcal{X} \in \mathbf{X}(\phi)} \prod_{v \in \mathcal{X}} \theta_t(v)$. The combined output is the current parse forest

[8]Perhaps surprisingly, theoretically speaking, it need not even be the case that convergence is to a local maxima. Even a local *minima* can be the final answer if it cannot be escaped by (a) changing only hidden variables, nor by (b) changing only model parameters. Such a point, if not also a maxima, requires a coordinated change in both parameters and variables in order to improve the loss function.

$\mathbf{X}_t = \sum_{\phi \in \Phi} \mathcal{X}_t^*(\phi)$ of all of the observed plans Φ. This computation can be implemented reasonably efficiently in a bottom-up fashion since any subtree of a most probable parse is also a most probable parse (of the subsequence it covers, given its root, etc.). That is, the following identity holds:[9]

$$\max_{\mathcal{X}} P(\mathcal{X} \mid a_i, \ldots, a_j, Z, \mathcal{H}_t) = \max_{\substack{\ell \\ m = Z \to XY \in \mathcal{M}(Z)}} (\theta_t(m) \cdot \max_{\mathcal{X}_{\text{left}}} P(\mathcal{X}_{\text{left}} \mid a_i, \ldots, a_\ell, X, \mathcal{H}_t) \cdot \max_{\mathcal{X}_{\text{right}}} P(\mathcal{X}_{\text{right}} \mid a_{\ell+1}, \ldots, a_j, Y, \mathcal{H}_t)). \quad (12)$$

The parsing then consists of computing all of its instantiations. More specifically, it suffices to bottom-up record the above products in a 4-dimensional table indexed by $i < \ell < j \in [n]$ and each of the methods $\mathcal{M}$, interleaving that with filling out the smaller 3-dimensional table (recording the left-hand maximizations) indexed by i, j and the tasks $\mathcal{T}$. By also recording the specific methods and midpoints witnessing the left-hand maximizations, the presently most probable parse of ϕ, $\mathcal{X}_t^*(\phi)$, can be easily extracted top-down, i.e., by beginning at the method and midpoint witnessing $\max_{\mathcal{X}} P(\mathcal{X} \mid a_1, \ldots, a_n, T, \mathcal{H}_t)$. So with r the number of methods, the parsing is bounded by $O(n^3 r)$.

Notes: (a) In theory the number of methods could be cubic in the number of tasks, but in practice, the point of structure learning is to prevent this. A careful implementation can exploit that assumption (that the grammar will not really permit anything close to allowing every symbol to reduce to every other pair of symbols) so as to not quite so literally fill out the tables described.

(b) This simple (dynamic programming) description of the parsing is convenient as it also outlines the framework for bottom-up generation of all possible parses: replace the maximizations with manipulation of weighted sets of parses.

(c) In other contexts, namely learning pCFGs, it is also convenient to note that the parsing easily accomodates direct, even noisy, observations of non-primitives (as generated by, say, a part-of-speech tagger). This is because the parsing computes most probable sub-trees conditioned on *every* conceivable sub-tree root; with minor modifications, additional observations can be permitted in the form of non-uniform priors over the sub-tree roots.

M-step: After getting the most probable parse trees (with respect to the current parameters) for all plan examples, the learner moves on to the M-step. In this step, the probabilities associated with each method are updated by maximizing the likelihood of generating those particular parse trees singled out by the E-step. This merely consists of setting each probability according to its relative frequency in the parse trees just computed.

To derive that update rule: Let $M[\text{event}]$ count events in the parse forest of most probable parse trees computed in the E-step: $\mathbf{X}_t = \sum_{\phi \in \Phi} \mathcal{X}_t^*(\phi)$. Let $\mathbf{V} = V(\mathbf{X})$ be all of the vertices of all the most probable parse trees. Then more specifically, let $M[Z]$ count those vertices $v \in \mathbf{V}$ satisfying $T(v) = Z$ and let $M[m]$ count those satisfying $m(v) = m$. So, with $m \in \mathcal{M}(Z)$, the answer will be $\theta_{t+1}(m) = \frac{M[m]}{M[Z]}$. Say $\mathcal{X} \in \mathbf{X}_t$ to mean that $\mathcal{X}$ is

[9]The base case is particularly simple since we associated, for convenience, every primitive a, with a distinct non-primitive Z_a and method $Z_a \to a$.

one of the components, a most probable parse tree, of $\mathbf{X}_t$, then:

$$\begin{aligned}
\theta_{t+1} &= \operatorname{argmax}_\theta P(\mathbf{X}_t \mid T, \mathcal{A}, \mathcal{T}, \mathcal{M}, \theta), \\
&= \operatorname{argmax}_\theta \prod_{\mathcal{X} \in \mathbf{X}_t} P(\mathcal{X} \mid T, \mathcal{A}, \mathcal{T}, \mathcal{M}, \theta), \\
&= \operatorname{argmax}_\theta \prod_{\mathcal{X} \in \mathbf{X}_t} \prod_{v \in V(\mathcal{X})} P(m(v) \mid T(v), \theta), \\
&= \operatorname{argmax}_\theta \prod_{v \in \mathbf{V}} \theta(m(v)), \\
&= \operatorname{argmax}_\theta \prod_{Z \in \mathcal{T}} \prod_{m \in \mathcal{M}(Z)} \theta(m)^{M[m]}. \qquad (13)
\end{aligned}$$

The last step is justifed because $M[m]$ will be 0 for any method not appearing in any of the parses, so, the extra terms being introduced into the product are all 1: take $0^0 = 1$. (Which will indeed be the form of the extra terms for the maximizing choice of θ.)

Each sub-product (i.e., for each $Z \in \mathcal{T}$) is a multinomial in the variables $\theta(\mathcal{M}(Z))$, subject to the constraint $\sum \theta(\mathcal{M}(Z)) = 1$. There are no other constraints, so, each such sub-product can be independently maximized. Abstracting, the form of the problem is:

$$\text{maximize} \prod_i x_i^{y_i}, \qquad (14)$$

$$\text{subject to} \sum_i x_i = 1. \qquad (15)$$

This is a classic problem [Koller and Friedman 2009, chapter 17]. As normal, one differentiates the logarithm in order to solve it. The answer is: $\forall i, x_i = \frac{y_i}{\alpha}$ (for $\alpha = \sum_i y_i$). So in the parameters θ_{t+1} we have, as claimed, for all $Z \in \mathcal{T}, m \in \mathcal{M}(Z)$:

$$\begin{aligned}
\theta_{t+1}(m) &= \frac{M[m]}{\sum_{m' \in \mathcal{M}(Z)} M[m']}, \qquad (16) \\
&= \frac{M[m]}{M[Z]}. \qquad (17)
\end{aligned}$$

Carrying out this simple update is linear in $\mathbf{X}_t$. So an efficient implementation need not even remember all the parse trees. Indeed, the $M[]$ counters can be computed during parsing itself, obviating the need to even remember a single complete parse tree. So the run-time per iteration, an E-step followed by an M-step, is linear in the training set: $O(|\Phi|)$. (Or, if parsing times are highly variable across plans, but stable across iterations, then estimate by $O(\mathbf{X})$ instead.) In any case, the per-iteration time is not the big question. How many iterations to run EM for is. While guaranteed to converge, convergence is only to a local optima. So, somewhat less obvious stopping rules such as "run for 1000 iterations" can be quite effective, if, say, EM is embedded in one of the many meta-learning strategies for dealing with locally optimizing learners given multimodal objectives. In our experiments plain EM sufficed, and was more than fast enough to just permit it to run for very large iteration counts.

Summary: The E-step completes the input data Φ by computing the parses of Φ 'expected' by $\mathcal{H}_\ell$. Subsequently the M-step treats those parses as ground truth, by setting the new method probabilities to the 'observed' frequency of their application in the completed data. This improves the likelihood of the model (under the assumptions of the above derivation, such as i.i.d. random variables), and the process is repeated until convergence to a local optima of the likelihood function.

3.3 Discussion

Hard-EM: As described, the method is the more intuitive, but nonstandard, version of EM known as hard-EM. In our case the choice of hard-EM has the specific effect of introducing bias in favor of *less ambiguous grammars*. That is, letting $\mathbf{X}$ denote the forest of parse trees (over the plans Φ), hard-EM is actually examining the question of maximizing $P(\mathcal{H}, \mathbf{X} \mid \Phi)$ rather than the true objective of maximizing $P(\mathcal{H} \mid \Phi)$. (Soft-EM examines the true objective.) This single design choice has quite a few advantages, or if you prefer, one fundamental effect that presents itself in different ways. Beyond (a) the computational advantage of considering only most probable parses, and (b) the well-known generalization advantage afforded by bias against complex models, there is a less-discussed advantage that we exploit in the following: (c) hard-EM seeks models which are easier to *explain*/justify in terms of a limited number of examples. The very same effect also carries the disadvantage that hard-EM can be expected to fail to produce the true solution to the learning problem even provided data well in excess of sample complexity. For much more analysis and discussion of the specific tradeoffs between hard- and soft-EM see [Kearns et al. 1997; Kandylas et al. 2007]; for a comprehensive treatment of theoretical and practical issues in probabilistic reasoning over graphs in general see [Koller and Friedman 2009].

Structure versus Parameter Learning: Although the EM phase of learning does not introduce new methods, it does (potentially) participate in structure learning in the sense that it can effectively delete methods by assigning zero probability. Accordingly the implementation, in post-processing, actually deletes such methods. Moreover it deletes any methods with probabilities too close to 0 (on the grounds of numerical instability and/or further regularization). For this reason SH does not attempt to find a completely minimal grammar before running EM. So specifically SH may introduce slightly more tasks and methods than strictly necessary. Then the EM phase has some freedom to play a (limited) role in the choice of the structure of the final grammar. As-is this increase in the number of tasks is quite small and there is little danger of ruining goodness-of-fit, sample complexity, and other such curse-of-dimensionality issues. The important issues and tradeoffs to be considered when examining the relationship between parameter and structure learning techniques are beyond the scope of this paper: see [Koller and Friedman 2009].

But to illustrate the issues: No simple loop ($Z \to ZS$) fits as closely as the disjunction over all finite prefixes of the recursion actually witnessed in the training data. Provided with such rules, and the right starting parameters, the EM phase will 'delete' the loop in preference to its unrollings, because it can achieve 'perfect' fit that way. Of course the resulting model will not fare so well on the test data. Conversely, roughly this very same behavior is indeed correct in the slightly elaborated situation that the majority of training data consists of examples of long repeating sequences with just a minority of short examples. The hypothesis that such (bimodal) data was produced by a loop with a stochastic exit event is weak. (It is much more plausible that a single event determines short/long, with any subsequent variation in length explained perhaps as loops with high exit probability.) An *ideal* parameter learning approach would, given the opportunity, have no difficulty in making the correct, structural, decision: 'delete' the explanation as a single simple loop. But treating all structure as parameters is hardly feasible, which is of course the motivation for considering structure learning as a separate problem in the first place. In short the interplay is a complex topic.

Contextual Dependencies: In general one might very well want to model/learn preferences such as "If in Europe, prefer traveling by trains to planes."; however, the setup as given insists on purely context-free statements. There is a general mapping between the context-sensitive and context-free settings best illustrated by example. One considers each term like "(buy ?customer ?vendor ?location ?product ?price ?unit)" and every specific instantiation such as "(buy mike joe walmart bat 3 dollars)", and instead represents these along the lines of "buy-mike-joe-walmart-bat-3-dollars". (For the sake of discussion, push all implicit dependencies, as on global variables, explicitly into parameters.) Then methods which appear to be restricted to context-free settings can be seen to be implicitly performing context-sensitive inference — but the mapping is an exponential translation. So, taken literally, such mappings are far from practical. Nonetheless one can get a surprising amount of mileage from the perspective so long as one does not literally write down the full ground representation ahead of time. The 'trick' is to only write down small pieces of the ground representation, i.e., on an as-needed basis, in some clever fashion.

Concretely, applying our techniques to learn context-sensitive preferences entails a feature-selection step to write down primitives along the lines of *BuyTicketInEurope* rather than just *BuyTicket*. Note that such a feature-selection step already 'exists' — we have already chosen to write *BuyTicket* rather than merely *Buy*. Future work could build on this work by *automating* the feature-selection step in order to better address contextual dependencies [Guyon and Elisseeff 2003; Liu and Yu 2005].

3.4 Evaluation

To evaluate our pHTN learning approach, we designed and carried out experiments in both synthetic and benchmark domains. All the experiments were run on a 3.06 GHz Mac machine with 4GB of RAM. Although we focus on accuracy (rather than CPU time), we should clarify up-front that the runtime for learning is quite reasonable — between less than a millisecond to 4ms per training plan. We take an oracle-based experimental strategy, that is, we generate an oracle pHTN $\mathcal{H}^*$ to represent a possible user and then subsequently use it to generate a set of preferred plans Φ. Our learner then induces a pHTN $\mathcal{H}$ from only Φ, allowing us to assess the effectiveness of the learning in terms of the differences between the original and learned models. In some settings (e.g., knowledge discovery) it is very interesting to directly compare the syntax of learned models against ground truth, but for our purposes such comparisons are much less interesting: we can be certain that, syntactically, $\mathcal{H}$ will look nothing like a real user's preferences (as expressed in pHTN form) for the trivial reason (among others) that $\mathcal{H}$ will be in Chomsky normal form. For our purposes, since user preferences are expressed as the distribution of observed plans, the correctness of $\mathcal{H}$ should be measured as whether it is able to generate an approximately correct distribution on plans. So the ideal evaluation is some measure of the distance between distributions (on plans), for example Kullback-Leibler (KL) divergence:

$$D_{KL}(\mathcal{P}_{\mathcal{H}^*} \mid\mid \mathcal{P}_{\mathcal{H}}) = \sum_{\phi} \mathcal{P}_{\mathcal{H}^*}(\phi) \cdot \log \frac{\mathcal{P}_{\mathcal{H}^*}(\phi)}{\mathcal{P}_{\mathcal{H}}(\phi)}, \tag{18}$$

where $\mathcal{P}_{\mathcal{H}}$ and $\mathcal{P}_{\mathcal{H}^*}$ are the distributions of plans generated by $\mathcal{H}$ and $\mathcal{H}^*$ respectively. This measure is lower-bounded by 0, achieved by equal distributions, and otherwise diverges to positive infinity. (There are both positive and negative terms, but, the positive terms always dominate.) It is not symmetric.

However, as given the summation is over the infinite set of all plans, so instead we approximate by sampling, but this exacerbates a deeper problem: the measure is trivially infinite if $\mathcal{P}_{\mathcal{H}}$ gives 0 probability to any plan (that $\mathcal{P}_{\mathcal{H}^*}$ does not). So in the following, for every oracle pHTN $\mathcal{H}^*$ with n tasks, we take measurements by sampling $100n$ plans from each of $\mathcal{H}^*$ and $\mathcal{H}$, obtaining sample distributions $\hat{\mathcal{P}}_{\mathcal{H}^*}$ and $\hat{\mathcal{P}}_{\mathcal{H}}$, then we prune any plans not in $\hat{\mathcal{P}}_{\mathcal{H}^*} \cap \hat{\mathcal{P}}_{\mathcal{H}}$, renormalize, obtaining $\hat{\mathcal{P}}'_{\mathcal{H}^*}$ (say P_1) and $\hat{\mathcal{P}}'_{\mathcal{H}}$ (say P_2), and finally compute:

$$\hat{D}(\mathcal{H}^* \mid\mid \mathcal{H}) = D_{KL}(P_1 \mid\mid P_2) = \sum_{\phi} P_1(\phi) \cdot \log \frac{P_1(\phi)}{P_2(\phi)}. \tag{19}$$

This is not a good approach if the intersection is small, but in our experiments $|\hat{\mathcal{P}}_{\mathcal{H}^*} \cap \hat{\mathcal{P}}_{\mathcal{H}}|/|\hat{\mathcal{P}}_{\mathcal{H}^*} \cup \hat{\mathcal{P}}_{\mathcal{H}}|$ is close to 1 (i.e. greater than 0.98 on average). This measure remains non-symmetric, non-negative, and divergent in a sense. But effectively the measure is upper-bounded by $O(\log n)$, because probabilities cannot be smaller than $\frac{1}{n}$ in distributions defined by n samples.

3.5 Experiments in Randomly Generated Domains

In these experiments, we randomly generate the oracle pHTN $\mathcal{H}^*$ by randomly generating a set of recursive and non-recursive methods on n tasks. In non-recursive domains, the randomly generated methods form a binary and-or tree with the goal as the root. The probabilities are also assigned randomly. Generating recursive domains is similar with the only difference being that 10% of the methods generated are recursive. For measuring overall performance we provide $10n$ training plans and take $100n$ samples for testing; for any given n we repeat the experiment 100 times and plot the mean. We compare the performance of the proposed learner with the inside-outside algorithm. The results are shown in Figure 4(a) and 4(b). We also discuss two additional, more specialized, evaluations.

Learning Curves: In order to test the learning speed, we first measured KL divergence values with 15 non-primitives given different numbers of training plans. The results are shown in Figure 4(a). We can see that even with a relatively small number of training examples, our learning mechanism can still construct pHTNs with divergence no more than 0.2. As expected, the performance further improves given many training examples. As briefly discussed in the setup, our measure is not interesting unless the learned pHTN can reproduce most testing plans with non-zero probability, since any 0 probability plans are ignored in the measurement — so we do not report results given only a very small number of training examples (the value would be artificially close to 0). Here ‘very small’ means too small to give at least one example of every/most reductions in the oracle pHTN; without at least one example the structure hypothesizer will (rightly) prevent the generation of plans with such structure.

Comparison with Inside-Outside: To better understand the effectiveness of the proposed learner, we also compared our schema learner with the famous inside-outside al-

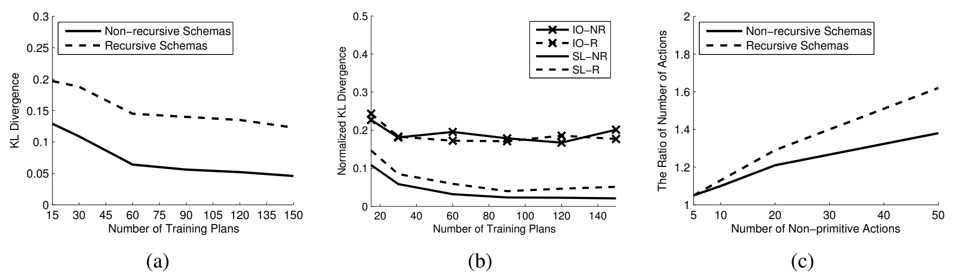


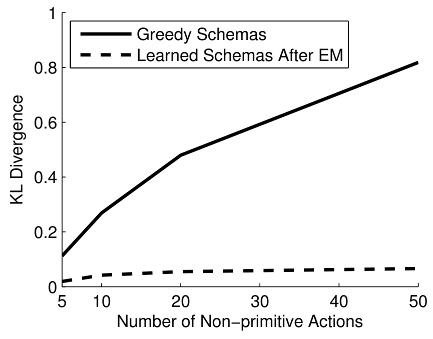
Fig. 4. Experimental results in synthetic domains (a) KL Divergence values with different number of training plans. (b) Normalized KL Divergence values of the proposed schema learner (SL) and the inside-outside algorithm (IO) with different number of training plans. (c) Measuring conciseness in terms of the ratio between the number of actions in the learned and original schemas.

gorithm [Lari and Young 1990]. [10] The inside-outside algorithm was also tested with domains of size 15 given an increasing number of training examples. Since the inside-outside algorithm requires a pre-specified number of non-primitives symbols given in addition to the training traces as input, we give it the **actual** number of non-primitives used by the oracle. Our modified measure of distribution divergence removes plans not in $\hat{\mathcal{P}}_{\mathcal{H}^*} \cap \hat{\mathcal{P}}_{\mathcal{H}}$; but in fact few plans were removed in the case of evaluating the proposed learner. The same manipulation removes about 15% of plans in the case of the inside-outside algorithm, significantly boosting its apparent performance. Despite the advantage, our approach still outperforms the inside-outside algorithm: 0.046 versus 0.268 in the non-recursive domains, and 0.120 versus 0.240 in the recursive domains, given 150 training plans, on the performance metric originally defined.

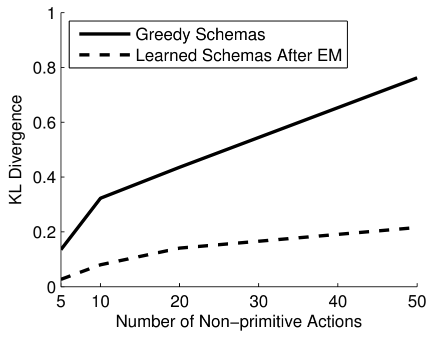
In an attempt to quantify the magnitude of the advantage given to the Inside-Outside algorithm under the original performance metric, we also evaluated a different measure of divergence between distributions, *normalized KL divergence*, as follows. First we compute $\hat{\mathcal{P}}_{\frac{1}{2}(\mathcal{H}+\mathcal{H}^*)} = \frac{1}{2}(\hat{\mathcal{P}}_{\mathcal{H}^*} + \hat{\mathcal{P}}_{\mathcal{H}})$, i.e., take the union of both sample sets, and then compute the KL divergence between $\hat{\mathcal{P}}_{\mathcal{H}}$ and $\hat{\mathcal{P}}_{\frac{1}{2}(\mathcal{H}+\mathcal{H}^*)}$. This is an extremely generous evaluation framework: the learner gets credit for any output whatsoever. In particular the worst possible learner, producing only plans loathed by a user, for this measure anyways scores $1 \log 2$. Taking the base of logarithms at 2 (which is natural for this sort of expression) the measure is bounded by 0 and 1. The measure is still non-symmetric.

As shown in Figure 4(b), the proposed learner consistently outperforms the inside-outside algorithm with different numbers of the training plans. Sign tests show that the differences are significant ($p < 0.0001$) across various numbers of training plans. A reasonable explanation begins by noting that the inside-outside algorithm begins with non-zero probability of every possible method, and inspection reveals that in our experiments many methods are retained (probability not close to 0) even after learning. The design choice of soft-EM does not help in this regard. In contrast our learner begins with a structure learning step to effectively limit the scope of the subsequent parameter learning. (Furthermore it uses hard-EM, thereby encouraging grammar unambiguity.) Having a plethora

[10]The implementation is by Mark Johnson: `http://www.cog.brown.edu/~mj/Software.htm`



(a)



(b)

Fig. 5. Experimental results in synthetic domains (a) KL Divergence between plans generated by original and learned pHTNs in *non-recursive* domains. (b) KL Divergence between plans generated by original and learned pHTNs in *recursive* domains.

of parameters brings at least two dangers into play: *overfitting* and *sample complexity*. As the test distributions were in fact identical to the training distributions (up to sampling error), the greatest obstacle facing the inside-outside algorithm is likely the lack of sufficiently many samples to set its many parameters reliably. Albeit, discussed above, about 15% of plans failed to appear in the intersection during testing, so overfitting may also be playing a role. This catastrophic ambiguity of the pHTNs considered by the inside-outside algorithm has one final negative feature that we will note (once more exacerbated by soft-EM): its per plan training times range from 100–303 milliseconds. (Our learner uses less than 4 milliseconds per plan.)

Conciseness: The conciseness of the learned model is also an important factor measuring the quality of the approach (despite being a syntactic rather than semantic notion), since allowing huge pHTNs will overfit (with enough available symbols the learner could, in theory, just memorize the training data). A simple measure of conciseness, the one we employ, is the ratio of non-primitives in the learned model to non-primitives in the oracle (n) — the learner is not told how many symbols were used to generate the training data. Figure 4(c) plots results. For small domains (around $n = 10$) the learner uses between 10% and 20% more non-primitives, a fairly positive result. However, for larger domains this result degrades to 60% more non-primitives, a somewhat negative result. Albeit the divergence measure improves on the test set, so while there is some evidence of possible overfitting, the result is not alarming. Future work in structure learning should nonetheless examine this issue (conciseness and overfitting) in greater depth.

Effectiveness of the EM Phase: To examine the effect of the EM phase, we carried out experiments comparing the divergence (to the oracle) before and after running the EM phase. Figures 5(a) and 5(b) plot results in the non-recursive and recursive cases respectively. Overall the EM phase is quite effective, for example, with 50 non-primitives in the non-recursive setting the EM phase is able to improve the divergence from 0.818 (the divergence of the model produced by SH) to the much smaller divergence of 0.066. The result is statistically significant under sign-testing, $p < 0.001$, except for domains of size 5 (where the difference in performance is not statistically significant). Our best explana-

Table II. Learned schemas in Logistics

Primitives: load, fly, drive, unload;	
Tasks: movePackage, $S_0, S_1, S_2, S_3, S_4, S_5$;	
movePackage $\rightarrow$ movePackage movePackage, 0.17	
movePackage $\rightarrow S_0\ S_5, 0.25$	$S_5 \rightarrow S_3\ S_2, 1.0$
movePackage $\rightarrow S_0\ S_4, 0.58$	$S_4 \rightarrow S_1\ S_2, 1.0$
$S_0 \rightarrow$ load, 1.0	$S_1 \rightarrow$ fly, 1.0
$S_2 \rightarrow$ unload, 1.0	$S_3 \rightarrow$ drive, 1.0

tion based on careful inspection of the experimental results is that random domains on 5 non-primitives are just too simple to sufficiently penalize the default assigment SH makes.

Note: Divergence in the recursive case is consistently larger than in the non-recursive case across all experiments: this is expected. In the recursive case the plan space is actually infinite; in the non-recursive case there are only finitely many plans that can be generated. So, for example, in the non-recursive case, it is actually possible for a finite sample set to perfectly represent the true distribution: simply memorizing the training data will produce 0 divergence eventually. In infinite plan spaces no finite set of samples can perfectly represent the true distribution.

3.6 Benchmark Domains

In addition to the experiments with synthetic domains, we also evaluated on two domains inspired by benchmarks of the International Planning Competition [Malte Helmert and Refanidis 2008]. In each, we hand-craft pHTNs encoding our own preferences and from there continue to employ the same oracle-based evaluation approach. The two domains are Logistics and GoldMiner; but in both we simplify by taking our primitives as just the operator names (rather than a full ground representation). For Logistics the preference is for planes over trucks, and fewer vehicles over more vehicles. In Gold Miner the preference encompasses the entire solution strategy (rather than the more localized preferences of logistics). Both domains feature simple looping behavior, which in the preceding synthetic experiments had a notable negative impact on performance, despite the fact that SH is specifically built to recognize simple looping behavior.

Logistics Planning: The domain we used in the first experiment is a variant of the Logistics planning domain, inside which both planes and trucks are available to move packages, and every location is reachable from every other. There are 4 primitives in the domain: *load*, *fly*, *drive* and *unload*. We use 11 tasks to express, in the form of an oracle pHTN $\mathcal{H}^*$ (in Chomsky Normal Form, hence 11 tasks), our preferences concerning logistics planning. We presented 100 training plans of lengths ranging from 3 to 15 to the learning system; by inspection we verified that these demonstrate our preference for moving packages by planes rather than trucks and for using overall fewer vehicles. The divergence of the learned model is 0.04 agains a test set of size 1000 on a single run.

It is interesting to note that the learned structure is smaller than the oracle structure. As the oracle structure associated meaningful names with non-terminals it is indeed plausible that there is a degree of approximate redundancy to the manner of its encoding — it could very well be that legitimate encoding of our preferences can be had more compactly. So while we are generally unconcerned with syntax of the learned model, it is interesting to consider in this case: Table II shows the learned model. With some effort one can ver-

Table III. Learned schemas in Gold Miner

Primitives: move, getLaserGun, shoot, getBomb, getGold;	
Tasks: goal, $S_0, S_1, S_2, S_3, S_4, S_5, S_6$;	
goal $\rightarrow S_0$ goal, 0.78	goal $\rightarrow S_1\ S_6, 0.22$
$S_0 \rightarrow$ move, 1.0	$S_5 \rightarrow S_2\ S_0, 1.0$
$S_1 \rightarrow$ getLaserGun, 0.22	$S_1 \rightarrow S_1\ S_5, 0.78$
$S_2 \rightarrow$ shoot, 1.0	$S_6 \rightarrow S_3\ S_4, 1.0$
$S_3 \rightarrow$ getBomb, 0.29	$S_3 \rightarrow S_3\ S_0, 0.71$
$S_4 \rightarrow$ getGold, 1.0	

ify that the learned schemas do capture our preferences: the second and third methods for 'movePackage' encode delivering a package by truck and by plane respectively (and delivering by plane has significantly higher probability), and the first method permits repeatedly moving packages between vehicles, but with relatively low probability. That is, it is possible to recursively expand 'movePackage' so that one package ends up transferring between vehicles, but, the plan that uses only one instance of the first method per package is significantly more probable (by 0.17^{-k}, according to the learner, for k transfers between vehicles).

Gold Miner: The second domain we used is inspired by Gold Miner, introduced in the learning track of the 2008 International Planning Competition. The setup is a (futuristic) robot tasked with retrieving gold (blocked by rocks) within a mine; the robot can employ bombs and/or a laser cannon. The laser cannon can destroy both hard and soft rocks, while bombs only destroy soft rocks. However, the laser cannon will also destroy any gold immediately behind its target. The desired strategy, which we encode in pHTN form using 12 tasks ($\mathcal{H}^*$), for this domain is roughly: *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.* As in Logistics, though, the training data consists merely of the names of the operators (so there is no way to express "hard rock", "soft rock" and "next to the gold").

We gave the system 100 training plans of various lengths ranging from 6 to 49 (generated by $\mathcal{H}^*$). The learner achieved a divergence of 0.52. This is a much larger divergence than in the case of Logistics above, which can perhaps be explained by the significantly longer applications of looping behavior (using the laser cannon repeatedly). As noted in the random experiments, performance is negatively impacted by the use of recursion/loops.

Nonetheless the learner did succeed in qualitatively capturing our preferences, which can be seen by inspection of the learned model in Table III. Specifically, the learned model only permits plans in the order given above: get the laser cannon, shoot, get and then use the bomb, and finally get the gold. Observe that the learner made such a leap of faith on just 100 training examples failing to demonstrate any other possible order: the use of a separate structure learning step is presumably to thank. One does not imagine a robust implementation of EM, or any other form of parameter-learning, capable of driving so many parameters to 0 on just 100 training examples. At least it tends to be the case, for parameter learners, that once logical certainty is reached by a parameter, it becomes stuck there forevermore. For robustness then, one takes steps to prevent assigning 0 or 1.

4. PREFERENCES CONSTRAINED BY FEASIBILITY

In general, users will not be so all-powerful that behavior and desire coincide. Instead a user must settle for one (presumably the most desirable) of the feasible possibilities. Supposing those possibilities remain constant then there is little point in distinguishing desire and behavior; indeed, the philosophy of behaviorism *defines* preference by considering such controlled experiments. Supposing instead that feasible possibilities vary over time, then the distinction becomes very important. Consider for example the fact that the observed travel behaviors of a poor grad student might all consist of her driving. Only in the rare situation that such a student's travel is funded, so we get to observe her true preference for planes over cars (or for that matter, cars over walking). In addition to that example, consider the requirement to go to work on weekdays (so the constraint does not hold on weekends). Clearly the weekend activities are the preferred activities. However, the learning approach developed so far would be biased — by a factor of $\frac{5}{2}$ — in favor of the weekday activities. In the following we consider how to account for this effect: the effect of feasibility constraints upon learning preferences.

Recall that we assume that we can directly observe a user's behavior, for example by building upon the work in *plan recognition*. In this section we additionally assume that we have access to the set of feasible alternatives to the observed behavior — for example by assuming access to the planning problem the user faced and building upon the work in *automated planning* [Nau et al. 2004]. In cases where we only have access to the planning problem description (i.e., the initial state and the goal), we could use planners capable of generating diverse plans [Srivastava et al. 2007; Nguyen et al. 2009]. Note that there might be an enormous number of feasible alternatives, but only a subset of them may have been chosen by the user at least once. Never chosen plans are considered as undesired plans, and thus do not need to be modeled by the acquired pHTNs. Therefore, when considering feasible alternatives, we can restrict our attention to a subset on the order of the number of observed plans. So, in this section, the input to the learning problem becomes:

Input. The i^{th} observation, $(\phi_i, F_i) \in \Phi$, consists of a set of feasible possibilities, F_i, along with the chosen solution: $\phi_i \in F_i$.

In the rest of the section we consider how to exploit this additional training information (and how to appropriately define the new learning task). The main idea is to rescale the input (i.e., attach weights to the observed plans ϕ_i) so that rare situations are not penalized with respect to common situations. One way of viewing the rescaling mechanism is that it makes appropriate numbers (i.e., weights) of plan duplicates based on different feasibility situations. Hence, in the above example, even if we have only observed that the poor graduate student travels by plane once, since this is a rare but informative situation, we could attach a large weight to the travel-by-plane plan. This is like we pretend that we have seen this plan carried out much more often in the "ideal" world situation.

We approach the learning problem from the perspective that our evidence for preference consists just of ϕ_i over the remainder of F_i. The question then becomes how to merge such evidence across differing feasibility situations: $F_i \neq F_j$. The approach is to consider plans in the intersection of both situations, using these to mediate an indirect comparison. That is, we attempt to take our original evidence and *transitively close* it. This may still leave us with disconnected components of situations. Here we essentially give up, and permit the system to answer 'unknown' concerning pairs of plans from distinct components. This additional capability somewhat complicates evaluation (as the base system can only answer 'yes' or 'no' to such queries). To answer queries about comparable plans: we apply the base learner to each rescaled component, arriving at a set of pHTNs capturing user preferences.

4.1 Analysis

Previously we assumed the training data (observed plans) Φ was sampled (i.i.d.) directly from the user's true preference distribution (say $\mathcal{U}$):

$$P(\Phi \mid \mathcal{U}) = \prod_{\phi \in \Phi} P(\phi \mid \mathcal{U}). \tag{20}$$

But now we assume that varying feasibility constraints intervene. For the sake of notation, imagine that such variation is in the form of a distribution, say $\mathcal{F}$, over planning problems (but all that is actually required is that the variation is independent of preferences, as assumed below). Note that a planning problem is logically equivalent to its solution set. Then we can write $P(F \mid \mathcal{F})$ to denote the prior probability of any particular set of solutions F. Since the user chooses among such solutions, we have that chosen plans are sampled from the posterior of the preference distribution:

$$P(\Phi \mid \mathcal{U}, \mathcal{F}) = \prod_{(\phi,F) \in \Phi} P(\phi \mid \mathcal{U}, F) \cdot P(F \mid \mathcal{F}). \tag{21}$$

Again since what is possible should not depend upon desire, and desire should not depend upon what is possible, we assume that preferences and feasibility constraints are mutually independent. One can certainly imagine either dependence — respectively Murphy's Law (or its complement) and the fox in Aesop's fable of Sour Grapes (or envy) — but it seems to us more reasonable to assume independence. Then we can rewrite the posterior of the preference distribution:

$$P(\Phi \mid \mathcal{U}, \mathcal{F}) = \prod_{(\phi,F) \in \Phi} \frac{P(\phi \mid \mathcal{U})}{\sum_{\phi' \in F} P(\phi' \mid \mathcal{U})} \cdot P(F \mid \mathcal{F}) \qquad \text{(by assumption).} \tag{22}$$

Assuming independence is important, because it makes the preference learning problem attackable. In particular, the posteriors preserve relative preferences — for all $\phi, \phi' \in F$, the *odds* of selecting ϕ over ϕ' are:

$$\begin{aligned} O(\phi, \phi') &:= \frac{P(\phi \mid \mathcal{U}, F)}{P(\phi' \mid \mathcal{U}, F)}, & (23)\\ &= \frac{P(\phi \mid \mathcal{U})}{\sum_{\phi'' \in F} P(\phi'' \mid \mathcal{U})} \div \frac{P(\phi' \mid \mathcal{U})}{\sum_{\phi'' \in F} P(\phi'' \mid \mathcal{U})}, & (24)\\ &= \frac{P(\phi \mid \mathcal{U})}{P(\phi' \mid \mathcal{U})}. & (25) \end{aligned}$$

Therefore we can, given sufficiently many of the posteriors, reconstruct the prior by transitive closure; consider ϕ, ϕ', ϕ'' with $\phi, \phi' \in F$ and $\phi', \phi'' \in F'$:

$$\begin{aligned} O(\phi, \phi'') &= O(\phi, \phi') \cdot O(\phi', \phi''), \\ &= \frac{P(\phi \mid \mathcal{U}, F)}{P(\phi' \mid \mathcal{U}, F)} \cdot \frac{P(\phi' \mid \mathcal{U}, F')}{P(\phi'' \mid \mathcal{U}, F')}. & (26) \end{aligned}$$

So then the prior can be had by normalization:

$$P(\phi \mid \mathcal{U}) = \frac{1}{1 + \sum_{\phi' \neq \phi} O(\phi', \phi)}. \tag{27}$$

Of course none of the above distributions are accessible; the learning problem is only given Φ. Let $M_F[\phi] = |\{i \mid (\phi, F) = (\phi_i, F_i) \in \Phi\}|$, $M_F = \sum_\phi M_F[\phi]$, and $M = \sum_F M_F = |\Phi|$. Then Φ defines a sampling distribution, for any F:

$$\begin{aligned} \hat{P}(\phi, F \mid \Phi) &= \frac{M_F[\phi]}{M}, && \text{so,} \\ \hat{P}(\phi \mid F, \Phi) &= \frac{M_F[\phi]}{M_F}, \\ &\approx P(\phi \mid \mathcal{U}, F) && \text{(in the limit),} \end{aligned}$$

in particular:

$$\hat{O}_F(\phi, \phi') := \frac{M_F[\phi]}{M_F[\phi']} \approx \frac{P(\phi \mid \mathcal{U})}{P(\phi' \mid \mathcal{U})} \qquad \text{(in the limit).} \tag{28}$$

However, for anything less than an enormous amount of data one expects $\hat{O}_F$ and $\hat{O}_{F'}$ to differ considerably for $F \neq F'$, therein lying one of the subtle aspects of the following rescaling algorithm. The intuition is, however, simple enough: pick some base plan ϕ and set its weight to an appropriately large value w, and then set every other weight, for example that of ϕ', to $w \cdot \hat{O}(\phi', \phi)$ (where $\hat{O}$ is some kind of aggregation of the differing estimates $\hat{O}_F$ which we will describe in more detail later); finally give the weighted set of observed plans to the base learner. From the preceding analysis, in the limit of infinite data, this setup will learn the (closest approximation, within the base learner's hypothesis space, to the) prior distribution on preferences.

To address the issue that different situations will give different estimates (due to sampling error) for the relative preference of one plan to another ($\hat{O}_F$ and $\hat{O}_{F'}$ will differ) we employ a merging process on such overlapping situations. Consider two weighted sets of plans, c and d, and interpret the weight of a plan as the number of times it 'occurs',[11] i.e., say $w_c(\phi) = M_c[\phi]$. In the simple case that there is only a single plan in the intersection, $\{\alpha\} = c \cap d$, then there is only one way to take a transitive closure — for all ϕ in c and $\phi' \in d \setminus c$:

$$\begin{aligned} \hat{O}_{cd}(\phi, \phi') &= \hat{O}_c(\phi, \alpha) \cdot \hat{O}_d(\alpha, \phi'), && (29) \\ &= \frac{M_c[\phi]}{M_c[\alpha]} \cdot \frac{M_d[\alpha]}{M_d[\phi']}, && (30) \\ &= \frac{M_c[\phi]}{s \cdot M_d[\phi']}, \qquad \text{with } s = \frac{M_c[\alpha]}{M_d[\alpha]}, && (31) \end{aligned}$$

so in particular we can merge d into c by first rescaling d:

$$M_{cd}[\phi] = \begin{cases} M_c[\phi] & \text{if } \phi \in c, \\ s \cdot M_d[\phi] & \text{otherwise.} \end{cases} \tag{32}$$

[11] The scaling calculations will likely produce non-integer weights though.

Table IV. An illustration of merging two clusters process.

	Gobyplane	Gobytrain	Gobybike
Cluster c	5		1
Cluster d	3	1	
Merged cluster {c, d}	15	5	3

In general let $s^{cd}_\alpha = \frac{M_c[\alpha]}{M_d[\alpha]}$ for any $\alpha \in c \cap d$ be the *scale factor* of c and d w.r.t. α. Then in the case that there are multiple plans in the intersection we are faced with multiple ways of performing a transitive closure, i.e., a set of scale factors. These will normally be different from one another, but, in the limit of data, assuming preferences and feasibility constraints are actually independent of one another, every scale factor between two clusters will be equal. So, then, we take the average:

$$s^{cd} := \frac{1}{|c \cap d|} \cdot \sum_{\alpha \in c \cap d} s^{cd}_\alpha, \qquad \text{and,} \tag{33}$$

$$M_{cd}[\phi] := \begin{cases} M_c[\phi] & \text{if } \phi \in c, \\ s^{cd} \cdot M_d[\phi] & \text{otherwise.} \end{cases} \tag{34}$$

In short, if all the assumptions are met, and enough data is given, the described process will reproduce the correct prior distribution on preferences. Figure 6 provides the remaining details in pseudocode, and in the following we discuss these details and the result of the rescaling/merging process operating in the Travel domain.

4.2 Rescaling

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

Clustering: First, we consider input records associated with the same set of feasible plans are produced under the same or similar situations. 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 in similar or identical situations. 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 in Figure 6, which also defines 'similar' (as set inclusion). Future work should consider more sophisticated clustering methods.

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. As shown in Table IV, 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 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

Algorithm 2: Rescaling

```
   Input: Training records $\Phi$, $\epsilon$.
   Output: Clusters $C$.
 1 initialize $C$ to empty
                                  // Cluster plans from similar situations
 2 forall $(\phi, F) \in \Phi$ do
 3     if $\exists c \in C$ such that $F \subseteq c$ or $F \supseteq c$ then
 4         forall $p \in F \setminus c$ do
 5             add $p$ to $c$ with $w_c(p) := \epsilon$
 6         end
 7         if $w_c(\phi) \geq 1$ then
 8             increment $w_c(\phi)$
 9         else
10             $w_c(\phi) := 1$
11         end
12     else
13         initialize $c$ to empty
14         add $c$ to $C$
15         forall $p \in F$ do
16             add $p$ to $c$ with $w_c(p) := \epsilon$
17         end
18         $w_c(\phi) := 1$
19     end
20 end
                              // Merge clusters with non-empty intersections
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 += $\frac{w_c(p)}{w_d(p)}$
25     end
26     scale := $\frac{\text{sum\_ratios}}{|c \cap d|}$
27     forall $p \in d \setminus c$ do
28         add $p$ to $c$ with $w_c(p) := w_d(p) \cdot$ scale
29     end
30     remove $d$ from $C$
31 end
32 return $C$
```

Fig. 6. Pseudocode for the rescaling algorithm

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 $5 = \frac{w_c(\text{Gobytrain})}{w_d(\text{Gobytrain})}$. For pairs of clusters with more than one shared plan we scale $d \setminus c$ by the average of $\frac{w_c(\cdot)}{w_d(\cdot)}$ for each plan in the intersection, but we leave the weights of $c \cap d$ as in c (one could consider several alternative strategies for plans in the intersection). Computing the scaling factor happens in lines 21–26 and the entire merging process happens in lines 21–31 shown in Figure 6.

4.3 Learning

We learn a set of pHTNs for C by applying the base learner (with the obvious generalization to weighted input) to each $c \in C$:

$$\mathbf{H} = \{H_c = \text{EM}(\text{SH}(c)) \mid c \in C\}. \tag{35}$$

While the input clusters will be disjoint, the base learner may very well generalize its input such that various pairs of plans become comparable in multiple pHTNs within $\mathbf{H}$. Any disagreement is resolved by voting; recall that, given a pHTN $\mathcal{H}$ and a plan p, we can efficiently compute the most probable parse of p by $\mathcal{H}$. Let $\ell_{\mathcal{H}}(p)$ denote "the (a priori) likelihood of p", but, actually set it to the probability of the most probable parse of p. Given two plans p and q we let $\prec_{\mathcal{H}}$ order p and q by $\ell_{\mathcal{H}}(\cdot)$, i.e., $p \prec_{\mathcal{H}} q \iff \ell_{\mathcal{H}}(p) < \ell_{\mathcal{H}}(q)$; if either is not parseable (or tied) then they are incomparable by $\mathcal{H}$. Given a set of pHTNs $\mathbf{H} = \{\mathcal{H}_1, \mathcal{H}_2, \ldots\}$, we take a simple majority vote to decide $p \prec_{\mathbf{H}} q$ (ties are incomparable):

$$p \prec_{\mathbf{H}} q \iff |\{\mathcal{H} \in \mathbf{H} \mid q \prec_{\mathcal{H}} p\}| < |\{\mathcal{H} \in \mathbf{H} \mid p \prec_{\mathcal{H}} q\}|. \tag{36}$$

So, each pHTN votes, based on likelihood, for $p \prec q$ (meaning p is preferred to q), or $q \prec p$ (q is preferred to p), or abstains (the preference is unknown). Summarizing, the input Φ is 1) clustered, 2) transitively closed producing a smaller set of clusters, and 3) each is given to the base learner resulting in a set of pHTNs $\mathbf{H}$. Finally, the learned pHTNs $\mathbf{H}$ model the user's preferences via the relation $\prec_{\mathbf{H}}$.

4.4 Evaluation

In this part we are primarily interested in evaluating the rescaling extension of the learning technique, i.e., the ability to learn preferences despite feasibility 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.

4.4.1 *Setup.*

Performance: We again take an oracle-based experimental strategy, that is, we imagine a user with a particular ideal pHTN, $\mathcal{H}^*$, 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. More specifically we test the learner's performance in the following game. After training, the learner produces $\mathbf{H}_r$; to evaluate the effectiveness of $\mathbf{H}_r$ we pick random plan pairs and ask both $\mathcal{H}^*$ and $\mathbf{H}_r$ to pick the preferred plan. There are three possibilities: $\mathbf{H}_r$ agrees with $\mathcal{H}^*$ (+1 point), $\mathbf{H}_r$ disagrees with $\mathcal{H}^*$ (-1 point), and $\mathbf{H}_r$ declines to choose (0 points)[12].

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 $\mathcal{H}^*$; $100t$ pairs are generated, where t is the number of non-primitives. 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.

[12]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.

Domains/Users: As in the prior evaluation we evaluate on 1) randomly generated pHTNs modeling possible users, and on 2) hand-crafted pHTNs modeling our preferences in Logistics and Gold Miner. Both are extended with the same randomized model of feasibility constraints.

Training Data: We generate random problems by generating random solution sets in a particular fashion, that is, we model feasibility constraints using a particular random distribution on solution sets. To evaluate whether the proposed approach is able to recover the user's true preference when the observed plan distribution is obfuscated by feasibility constraints, 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 hypothetical user will be forced to pick the least evil rather than the greatest good. We describe this process in detail below, but note that the learning algorithm is general and not restricted to the type of feasibility obfuscation tested here.

We begin by constructing a list of plans, $\mathcal{P}$, from $100t$ samples of $\mathcal{H}^*$, removing duplicates by maintaining only the first appearance of the same plans (so $|\mathcal{P}| \leq 100t$). Since more preferred/probable plans are more likely to be generated first, in general, the order will be roughly from most to least preferred. We reverse that order, still denoted $\mathcal{P}$, and associate it with (a discrete approximation to) a power-law distribution. The result is that least preferred plans are, under $\mathcal{P}$, most likely. Both training and test plans are drawn from this distribution. Then, for each training record (ϕ_i, F_i), we take a random number[13] of samples from $\mathcal{P}$ as F_i. We sample the nominally observed plan, ϕ_i, from F_i by ℓ, that is, the probability of a particular choice $\phi_i = p$ is $\frac{\ell(p)}{\sum_{q \in F} \ell(q)}$.

Baseline: The baseline for our experiments will be the original approach: the base learner without rescaling. That is, we take a single cluster, where the weight of each plan is the number of times it is observed $w(\phi) = |\{i \mid \phi = \phi_i\}|$, and apply the base learner, obtaining a single pHTN, $\mathbf{H}_b = \{\mathcal{H}\}$, and score it in the same manner that the extended approach is scored by.

4.4.2 *Results: Random $\mathcal{H}^*$.*

Learning Curves: Figure 7(a) presents the results of a learning-rate experiment against randomly selected $\mathcal{H}^*$. For these experiments the number of non-primitives is fixed at 5 while the amount of training data is varied; we plot the average performance, over 100 samples of $\mathcal{H}^*$, at each training set size.

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 non-primitives (using $50t$ training records); the results are shown in Figure

[13]The number of samples taken is $|\mathcal{P}| \cdot n/2$, subject to minimum 2 and maximum $|\mathcal{P}|$, where n is drawn from the standard distribution $\mathcal{N}(0, 1)$. Larger solution sets model "easier" planning problems.

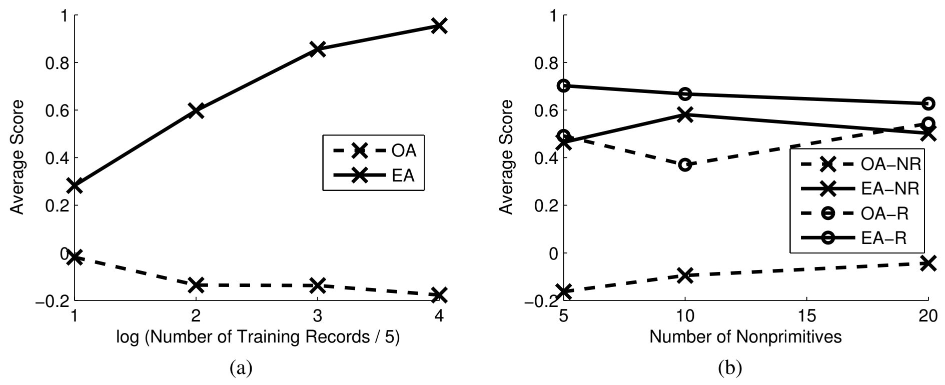


Fig. 7. Experimental results for random $\mathcal{H}^*$. "EA" is learning with rescaling and "OA" is learning without rescaling. Higher scores are better: 1 is perfect. Negative scores are *worse than randomly guessing*. (a) Learning curves for non-recursive $\mathcal{H}^*$. (b) Size dependence: "R" for recursive $\mathcal{H}^*$ and "NR" for non-recursive $\mathcal{H}^*$.

7(b). For technical reasons, discussed at the end of Section 4.4.3, the base learner is much more effective at recovering user preferences despite feasibility obfuscation when these take the form of recursive schemas, so there is less room for improvement. Nonetheless the rescaling approach improves upon learning alone in both experiments.

Significance: As shown in Figure 7(b), the learners with rescaling (*EA-NR*, *EA-R*) outperform the base learners (*OA-NR*, *OA-R*). To test whether the difference is significant or not, we carried out a sign test. For each domain size (i.e., 5, 10, 20), we compare the scores of the extended learner with the scores of the base learner over 100 schemas. The result shows that in both recursive and non-recursive domains, the score of the learner with rescaling is significantly higher ($p < 0.02$) for all domain sizes.

4.4.3 *Results: Hand-crafted $\mathcal{H}^*$.*

We re-use the same pHTNs encoding our preferences in Logistics and Gold Miner from the first set of evaluations. As mentioned we use the same setup as in the random experiments, so it continues to be the case that the distribution on random 'solutions' is biased specifically against the encoded preferences. Moreover, due to the level of abstraction used (truncating to action names), as well as the nature of the pHTNs and domains in question, the randomly generated sets of alternatives, F_i, are in fact sets of solutions to *some* problem expressed in the normal fashion (i.e., as an initial state and goal).

Logistics: After training with 550 training records ($50t$, for 11 non-primitives) the baseline system scored only 0.342 (0 is the performance of random guessing) whereas rescaling before learning performed much better with a score of 0.847 (0.153 away from perfect performance).

Gold Miner: After training with 600 examples ($50t$ for 12 non-primitives) learning alone scored a respectable 0.605, still, rescaling before learning performed better with a score of 0.706. The greater recursion in Gold Miner, as compared to Logistics is both hurting and helping. On the one hand the full approach scores worse (0.706 vs. 0.847), while on the other hand, the baseline's performance is hugely improved (0.605 vs. 0.342). As discussed previously, the presence of recursion in the preference model makes the learning problem much harder (since the space of acceptable plans is then actually infinite), which continues to be a reasonable explanation of the first effect (degrading performance).

The latter effect is more subtle. The experimental setup, roughly speaking, inverts the probability of selecting a plan, so that using a recursive method many times in an observed plan is more likely than using the same method only a few times. Then the baseline approach is attempting to learn a distribution skewed towards more recursion to less recursion, all else being equal. However, there is no pHTN that prefers more recursion to less recursion all else being equal: fewer uses of a recursive method always increases the probability of a plan. The closest the baseline can come is simply to fit an inappropriately large probability to any recursive method. So it will assess the likelihood of plans incorrectly. But, queried about the relative ordering on two plans, differing only in their depth of recursion over some method, the baseline will nonetheless produce the correct answer (prefer less recursion) despite assigning the wrong likelihoods. No other result is possible given the definition of pHTNs: always fewer uses of a method is (monotonically) more probable. Naive Bayes Classifiers exhibit an analogous effect [Koller and Friedman 2009, Box 17.A].

5. DISCUSSION AND RELATED WORK

In the planning community, HTN planning has for a long time been given two distinct and sometimes conflicting interpretations (c.f., [Kambhampati et al. 1998]): it can be interpreted either in terms of domain abstraction[14] or in terms of expressing complex plan constraints on plans[15]. The original HTN planners were motivated by the former view (improving efficiency via abstraction). In this view, only top-down HTN planning makes sense as the HTN is supposed to express effective search control. Paradoxically, w.r.t. that motivation, the complexity of HTN planning is substantially worse than planning with just primitive actions [Erol et al. 1996]. The latter view explains the seeming paradox easily — finding *a* solution should be easier, in general, than finding one that also satisfies additional complex constraints. From this perspective both top-down *and bottom-up* approaches to HTN planning are appropriate (the former if one is pessimistic concerning the satisfiability of the complex constraints, and the latter if one is optimistic). Indeed, this perspective lead to the development of bottom-up approaches to HTN planning [Barrett and Weld 1994].

Despite this dichotomy, most prior work on learning HTN models (e.g., [Ilghami et al. 2002; Langley and Choi 2006; Yang et al. 2007; Hogg et al. 2008]) has focused only on the domain abstraction angle. Typical approaches here either require the structure of the reduction schemas to be given as input, or need additional information such as annotated plan traces and tasks to assist the learning process, whereas our work only requires plan traces as input. Moreover, these efforts focus on learning domain physics or search control. As we mentioned, a significant amount of work was also directed at learning domain physics as action schemas [Yang et al. 2005]. In contrast, our work focuses on learning HTNs as a way to capture user preferences, given only successful plan traces. The difference in focus also explains the difference in evaluation techniques. While most previous HTN learning efforts are evaluated in terms of whether the learned schemas and applicability conditions are able to assist the planner to find feasible and goal-achieving plans, we evaluate them in terms of how close the distribution of plans generated by the learned model is to the

[14]Non-primitives are seen as abstract actions, mediating access to the concrete actions.
[15]Non-primitives are seen as standing for complex preferences (or even physical constraints).

distribution generated by the actual model.

An intriguing question is whether pHTNs learned to capture user preferences can, in the long run, be over-loaded with domain semantics. In particular, it would be interesting to combine the two HTN learning strands by sending our learned pHTNs as input to the method applicability condition learners. Presuming the user's preferences are amenable, the applicability conditions thus learned might then allow efficient top-down interpretation (of course, the user's preferences could, in light of the complexity results for HTN planning, be so antithetical to the nature of the domain that efficient top-down interpretation is impossible). An interesting theoretical result is that context-free languages are *not* closed under intersection; one might be unable to effectively merge two HTNs modeling different sets of complex constraints.

The connection between HTN schemas and grammars has been identified by several authors [Kambhampati et al. 1998; Geib and Steedman 2007]. Recently, Geib [2009] proposes an algorithm, ELEXIR, which represents plans to be recognized with Combinatory Categorial Grammar(CCG), and shows that this mechanism prevents early commitment to plan goals. Our work exploits the same connections to learn the pHTNs as grammars.

Our framework also incorporates ideas from other research on grammar induction. For example, the E-step in the algorithm to build the most probable parse trees bears a clear resemblance to the parsing algorithms in [Collins 1997; Charniak 2000]. Collin's [1997] parser represents parse trees using probabilities of dependencies, while our approach uses reduction schemas to represent parse trees. Charniak's [2000] work defines a maximum-entropy-inspired score function based on features chosen from certain feature schemata to measure the quality of the parse. The parser then returns the parse tree with the highest score. In contrast, our approach scores parse trees based on a probabilistic model of the derivation process (i.e. Equation 5).

Other research on pCFG acquisition is also quite relevant. Most work in this area divides the learning process into two steps as we do. The learning algorithms first acquire the grammar rules using CFG induction algorithms. Due to the high complexity of CFG learning, typical approaches in the direction either require additional structural information besides training examples to be given (e.g., [Sakakibara 1992]), or focus on restricted classes of CFGs (e.g., [Takada 1988; Ishizaka 1989]). In the second step, the learning algorithm uses the grammar rules acquired from previous step, and estimates the probabilities that fit best (e.g., [Lari and Young 1990; Ra and Stockman 1999; Sakakibara et al. 1994]). Genetic algorithms are also used to acquire pCFGs directly [Kammeyer and Belew 1996; Keller and Lutz 2005]. To the best of our knowledge, we are the first to apply pCFG learning to the area of user preference acquisition. An interesting future study would be to anyways compare the performance of the other learning algorithms on the problem of acquiring user preferences, despite the fact that these algorithms were not designed for this application. However, considering the performance of the inside-outside algorithm in our empirical evaluations, and the performance of our base learner once we took feasibility constraints into account, it does not seem likely that these algorithms would perform well when the surrounding context is dramatically altered.

Besides pHTNs, there are other representations for expressing user preferences, such as trajectory constraints expressed in linear temporal logic e.g., [Baier and McIlraith 2008; Gerevini et al. 2009]. Sohrabi et al. [2009], in particular, extend the Planning Domain Definition Language PDDL3 [Gerevini et al. 2009] with HTN-like preference constructs.

These works consider modeling user preferences, not automatically learning them. It will be interesting to explore methods for learning preferences in those representations too, and to see to what extent typical user preferences are actually expressible in (p)HTNs and alternatives.

6. CONCLUSION

Learning in the context of planning has received significant interest. However, most prior work focused only on learning domain physics or search control. In this paper, we expanded this scope by learning user preferences concerning plans. We developed a framework for learning probabilistic HTNs from a set of example plans, drawing from the literature on probabilistic grammar induction. Assuming the input distribution is in fact sampled from a pHTN, we demonstrated that the approach finds a pHTN generating a similar distribution. It is, however, a stretch to imagine that we can sample directly from preference distributions — observed behavior arises from a complex interaction between preferences and domain physics. We demonstrate a technique overcoming the effect of such feasibility constraints, by reasoning about the available alternatives to the observed user behavior. The technique is to rescale the distribution to fit the assumptions of the base pHTN learner developed in the first part. We evaluate our approach, and demonstrate both that the base learner is easily confounded by constraints placed upon the preference distribution, and that rescaling is effective at reversing this effect.

We also discussed several remaining important directions for future work to address. Of these, the most directly relevant technical pursuit is learning parameterized pHTNs, or more generally, learning conditional preferences. Fully integrating an automated planner with the learner would gain the important abilities to (a) automatically generate feasible alternatives prior to learning, and (b) exploit the learned knowledge during planning (so as to make better recommendations). Subsequently running user studies, i.e., on humans, is a very important pursuit.

Acknowledgments: Kambhampati's research is supported in part by ONR grants N00014-09-1-0017 and N00014-07-1-1049, and the DARPA Integrated Learning Program (through a sub-contract from Lockheed Martin).

REFERENCES

BAIER, J. A. AND MCILRAITH, S. A. 2008. Planning with preferences. *AI Magazine 29,* 4, 25–36.

BARRETT, A. AND WELD, D. S. 1994. Task-decomposition via plan parsing. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*. AAAI Press, Menlo Park, CA, 1117–1122.

CHARNIAK, E. 2000. A maximum-entropy-inspired parser. In *Proceedings of the First Conference on North American Chapter of the Association for Computational Linguistics*. Morgan Kaufmann Publishers Inc., San Francisco, CA, 132–139.

COLLINS, M. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, Morristown, NJ, USA, 16–23.

EROL, K., HENDLER, J. A., AND NAU, D. S. 1996. Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence 18,* 1, 69–93.

FIKES, R. AND NILSSON, N. J. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artif. Intell. 2,* 3/4, 189–208.

GEIB, C. W. 2009. Delaying commitment in plan recognition using combinatory categorial grammars. In *Proceedings of the 21st international jont conference on Artifical intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1702–1707.

GEIB, C. W. AND STEEDMAN, M. 2007. On natural language processing and plan recognition. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*. Hyderabad, India, 1612–1617.

GEREVINI, A. E., HASLUM, P., LONG, D., SAETTI, A., AND DIMOPOULOS, Y. 2009. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence 173*, 619–668.

GUYON, I. AND ELISSEEFF, A. 2003. An introduction to variable and feature selection. *The Journal of Machine Learning Research 3*, 1157–1182.

HOGG, C., MUÑOZ-AVILA, H., AND KUTER, U. 2008. HTN-MAKER: Learning HTNs with minimal additional knowledge engineering required. In *Proceedings of the 23rd National Conference on Artificial Intelligence*. AAAI Press, 950–956.

ILGHAMI, O., NAU, D. S., MUÑOZ-AVILA, H., AND AHA, D. W. 2002. CaMeL: Learning method preconditions for HTN planning. In *Proceedings of the Sixth International Conference on AI Planning and Scheduling*. AAAI Press, Toulouse, France, 131–141.

ISHIZAKA, H. 1989. Learning simple deterministic languages. In *Proceedings of the second annual workshop on Computational learning theory*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 162–174.

KAMBHAMPATI, S., MALI, A., AND SRIVASTAVA, B. 1998. Hybrid planning for partially hierarchical domains. In *Proceedings of the 15th National Conference on Artificial Intelligence and the 10th Conference on Innovative Applications of Artificial Intelligence*. AAAI Press, Menlo Park, CA, 882–888.

KAMMEYER, T. E. AND BELEW, R. K. 1996. Stochastic Context-Free Grammar Induction with a Genetic Algorithm Using Local Search. In *Foundations of Genetic Algorithms IV*. Morgan Kaufmann, 3–5.

KANDYLAS, V., UNGAR, L. H., AND FOSTER, D. P. 2007. Winner-Take-All EM clustering. In *NESCAI*.

KEARNS, M., MANSOUR, Y., AND NG, A. Y. 1997. An information-theoretic analysis of hard and soft assignment methods for clustering. In *Proceedings of 13th Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann, 282–293.

KELLER, B. AND LUTZ, R. 2005. Evolutionary induction of stochastic context free grammars. *Pattern Recognition 38,* 9, 1393 – 1406.

KOLLER, D. AND FRIEDMAN, N. 2009. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

LANGLEY, P. AND CHOI, D. 2006. A unified cognitive architecture for physical agents. In *Proceedings of the 21st National Conference on Artificial Intelligence*. AAAI Press, Boston, MA.

LARI, K. AND YOUNG, S. J. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language 4*, 35–56.

LI, N., CUSHING, W., KAMBHAMPATI, S., AND YOON, S. W. 2009. Learning user plan preferences obfuscated by feasibility constraints. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling*. Thessaloniki, Greece.

LI, N., KAMBHAMPATI, S., AND YOON, S. 2009. Learning probabilistic hierarchical task networks to capture user preferences. In *Proceedings of the 21th International Joint Conference on Artificial Intelligence*. Pasadena, CA.

LIU, H. AND YU, L. 2005. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering 17,* 4, 491–502.

MALTE HELMERT, M. D. AND REFANIDIS, I. 2008. The deterministic track of the international planning competition. http://ipc.informatik.uni-freiburg.de/.

NAU, D., GHALLAB, M., AND TRAVERSO, P. 2004. *Automated Planning: Theory & Practice*. Morgan Kaufmann, San Francisco.

NAU, D. S., AU, T.-C., ILGHAMI, O., KUTER, U., MURDOCK, J. W., WU, D., AND YAMAN, F. 2003. Shop2: An htn planning system. *J. Artif. Intell. Res. (JAIR) 20*, 379–404.

NGUYEN, T. A., DO, M. B., KAMBHAMPATI, S., AND SRIVASTAVA, B. 2009. Planning with partial preference models. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, C. Boutilier, Ed. Pasadena, CA, 1772–1777.

RA, D.-Y. AND STOCKMAN, G. C. 1999. A new one pass algorithm for estimating stochastic context-free grammars. *Information Processing Letters 72*, 37–45.

SAKAKIBARA, Y. 1992. Efficient learning of context-free grammars from positive structural examples. *Information and Computation 97*, 23–60.

SAKAKIBARA, Y., BROWN, M., HUGHEY, R., IS, SJÖLANDER, K., RC, AND HAUSSLER, D. 1994. Stochastic context-free grammars for tRNA modeling. *Nucleic Acids Research 22,* 23, 5112–5120.

SOHRABI, S., BAIER, J., AND MCILRAITH, S. A. 2009. HTN planning with preferences. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*. Pasadena, CA, 1790–1797.

SRIVASTAVA, B., NGUYEN, T. A., GEREVINI, A., KAMBHAMPATI, S., DO, M. B., AND SERINA, I. 2007. Domain independent approaches for finding diverse plans. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*. Hyderabad, India, 2016–2022.

TAKADA, Y. 1988. Grammatical inference for even linear languages based on control sets. *Information Processing Letters 28,* 4, 193–199.

YANG, Q., PAN, R., AND PAN, S. J. 2007. Learning recursive HTN-method structures for planning. In *Proceedings of the ICAPS-07 Workshop on AI Planning and Learning*. Providence, Rhode Island, USA.

YANG, Q., WU, K., AND JIANG, Y. 2005. Learning action models from plan examples with incomplete knowledge. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling*. AAAI Press, Monterey, CA, 241–250.

ZIMMERMAN, T. AND KAMBHAMPATI, S. 2003. Learning-assisted automated planning: Looking back, taking stock, going forward. *AI Magazine 24,* 2, 73–96.