Learning Combinatory Categorial Grammars for Plan Recognition

Christopher W. Geib
SIFT LLC.
319 1st Ave. South, Suite 400
Minneapolis, MN 55401
cgeib@sift.net

Pavan Kantharaju
Department of Computer Science, Drexel University
3141 Chestnut St
Philadelphia, PA 19104
pk398@drexel.edu

Abstract

This paper defines a learning algorithm for plan grammars used for plan recognition. The algorithm learns Combinatory Categorial Grammars (CCGs) that capture the structure of plans from a set of successful plan execution traces paired with the goal of the actions. This work is motivated by past work on CCG learning algorithms for natural language processing, and is evaluated on five well know planning domains.

Introduction

Combinatory Categorial Grammars (CCGs) (Steedman 2001) are a well known and effective grammar formalism developed for natural language processing (NLP). Past work has shown that *plan recognition* (PR) (the problem of recognizing an agent's plans based on observations of their actions (Schmidt, Sridharan, and Goodson 1978)) can be viewed as parsing a stream of observations using a grammar that defines the possible acceptable plans(Vilain 1990). Recent work, implemented in the ELEXIR system(Geib 2009; Geib and Goldman 2011) has very effectively used probabilistic CCGs to represent and recognize plans in a number of domains. It also demonstrated a number of advantages over previous work including efficient recognition of partially-ordered plans, multiple interleaved plans, and plans with loops. That said, such plan CCGs encode significant domain knowledge, and in past research, required time consuming hand-coding. This paper presents an incremental, domain-independent, supervised, probabilistic learning algorithm, LEX$_{learn}$, for the plan CCGs required by ELEXIR.

This work builds on prior work (Zettlemoyer and Collins 2005; Kwiatkowski et al. 2010), inducing CCGs for natural language grammars. Like this prior work, LEX$_{learn}$ learns probabilistic CCGs based on sequences of observations (in their case words, in ours actions) paired with a representation of the sequence's "meaning" (logical form of the sentence's meaning for them, goal state for us). This said, it is worth noting that plan recognition is inherently different from sentence parsing. Linguists have tentatively identified the basic parts of speech a word could take on (eg. noun, verb, article, adverb, etc...) and this knowledge is used to inform NLP learning algorithms. However, no such set of basic types exists for reasoning about actions. Thus, using these algorithms for plan grammars requires removing some of the linguistic intuitions that guided their formulation.

The rest of this paper is structured as follows: First, it presents related work on action and plan learning, and CCG induction in NLP. Second, it provides an introduction to plan CCGs and ELEXIR. Third, it presents a new plan CCG learning algorithm, LEX$_{learn}$and its experimental evaluation and concludes with a discussion of future work.

Related Work

There are three areas of related work for this research: 1) action learning, 2) plan hierarchy learning, and 2) CCG induction in NLP. We consider each of these in turn starting with action learning. The objective of action learning is to learn a set of precondition and effect rules that describe the results of executing a "basic action" in various world states. Successful work in this area has been based on both PDDL (Yang, Wu, and Jiang 2007) and STRIPS style action descriptions (Mourao et al. 2012). While such work is important, it is not the focus of this work. Instead, this work learns plan hierarchies that abstract common sequences of actions into more abstract plan structures like those in Hierarchical Task Networks (HTNs) (Erol, Hendler, and Nau 1994) .

Next, we consider work on learning such HTN plan structures that is much closer to this work. In addition to having low-level, executable actions, HTNs also have more abstract *tasks* that capture complex sequences of actions. Tasks are defined by one or more *methods*: a production rule with a task name on the left hand side and a sequence of basic actions and task names on the right:

$$\text{task} \rightarrow \{\text{action}^{*}\text{task}^{*}\}^{*}$$

Such methods can straightforwardly be seen as context free grammar (CFG) rules(Erol, Hendler, and Nau 1994). The addition of methods and a decompositional approach to the planning process, similar to CFG grammar rewriting methods, are the hallmark of HTN planning. Some formulations of HTNs augment methods with ordering constraints, preconditions, and even effects that we will not focus on here.

Learning HTN plan hierarchies amounts to learning the methods that define abstract tasks. (Hogg, Muñoz Avila, and Kuter 2008) successfully learns HTN methods from analyzing the state of the world before and after a given sequence of actions. (Zhuo, Muñoz-Avila, and Yang 2014) builds an HTN from partially-observable plan traces. (Nejati, Langley, and Konik 2006) learns teleoreactive logic programs, a specialized class of HTNs, from expert traces. There are two central differences between this work and ours. First, all three works focus on learning non-probabilistic HTNs, whereas our work focuses on probabilistic CCGs. Second, our work learns in the action space rather than the state space. That is, this work learns abstractions based on frequent sequences of actions rather than state changes.

The two closest related works to our approach are (Bisson, Larochelle, and Kabanza 2015; Li et al. 2014). (Li et al. 2014) successfully learn probabilistic HTNs (pHTNs) using techniques from Probabilistic CFG induction. However, they use expectation maximization for estimating the probability model, whereas our work uses gradient ascent. While related, (Bisson, Larochelle, and Kabanza 2015) use neural networks to solve the more limited problem of learning the probability model given an HTN. Our work, learns the plan grammar and its probability model. Finally, we would note, this is the first work on learning plan CCGs.

There is also related work from the NLP literature on learning CCGs for language. (Thomforde and Steedman 2011) presents Chart Inference, an unsupervised method used to derive structured CCG categories for unknown words using a partial parse chart. (Bisk and Hockenmaier 2012) provide an unsupervised CCG learner to generate categories from part-of-speech (PoS) tagged text that relies on minimal language-specific knowledge. (Kwiatkowski et al. 2012) define an CCG induction algorithm to learn sentence and semantic representation pairs from child utterances. (Zettlemoyer and Collins 2005) use supervised CCG induction to learn a mapping between sentences and their semantic representations. However, the different domains of application necessitate changes to the learning approach. First the different domains require different input representations for the observed tokens, (ie. words vs. actions) and meanings (ie. sentential logical forms vs. goal states). Second, our method of generating new CCG categories is different than those used by natural language CCG learners because, as we have already mentioned, the prior work is informed by linguistic intuitions that do not hold in the action domain. Finally, the gradient used to learn our probability model is different than (Zettlemoyer and Collins 2005). That said, this work takes this prior work as inspiration.

Representing Plans using CCGs

Next we will briefly describe plan CCGs and their use for plan recognition following the definitions used for the ELEXIR system in (Geib 2009; Geib and Goldman 2011). In this work, each action in a planning domain is associated with a set of CCG categories that can be thought of as functions and are defined recursively.

Definition 1.1 *We define a set of* ***CCG categories*** $\mathcal{C}$ *as:*

Atomic categories: *A finite set of base categories denoted* $\{A, B, C...\} \in \mathcal{C}$.

Complex categories: *Given a set of categories* $\mathcal{C}$, *where* $Z \in \mathcal{C}$ *and* $\{W, X, Y, ...\} \neq \emptyset$ *and* $\{W, X, Y, ...\} \in \mathcal{C}$, *then* $Z/\{W, X, Y, ...\} \in \mathcal{C}$ *and* $Z\backslash\{W, X, Y, ...\} \in \mathcal{C}$.

Atomic categories can be thought of as a zero-arity function that transitions from any initial state to a state associated with the atomic category. Complex categories define curried functions (Curry 1977) based on the two left associative operators "\" and "/". These operators each take a set of *arguments* (the categories on the right hand side of the slash, $(\{W, X, Y...\})$ and produces the state identified with the atomic category specified as its *result* (the category on the left hand side of the slash, Z). The slash also defines ordering constraints for plans, by indicating where the category's arguments are to be found relative to the action. Forward slash categories find their arguments temporally after the action, and backslash categories before it. The following definition will also be helpful.

Definition 1.2 *A category R is the* ***root*** *or* ***root result*** *of a category G if it is the leftmost atomic result category in G.*

For example, for a complex category $(C\backslash\{A\})\backslash\{B\}$, the root would be C. We can now define a CCG plan lexicon.

Definition 1.3 *A* ***plan lexicon*** *is a tuple,* $\Lambda = \langle\Sigma, \mathcal{C}, f\rangle$, *where* Σ *is a finite set of action types,* $\mathcal{C}$ *is a set of possible CCG categories, and* f *is a function such that* $\forall\sigma_i \in \Sigma f(\sigma_i) \rightarrow \{(c_{i,j} : P(c_{i,j}|\sigma_i))\}$ *such that* $c_{i,j} \in \mathcal{C}$ *and* $\forall\sigma_i, c_{i,j}, \sum_j P(c_{i,j}|\sigma_i) = 1$.

The function f maps each observable action, σ_i, to a non-empty set of pairs each made up of a category, $c_{i,j}$, and the conditional probability the action is assigned the category during parsing, P($c_{i,j}|\sigma_i$), given that σ_i is observed. Because f implicitly defines both Σ and $\mathcal{C}$, we will refer to learning a lexicon and learning its function interchangeably. Given this definition, any learning algorithm for a CCG must do two things. First, it must construct and associate with each action a set of categories that produce the correct set of parses, and second, correctly estimate each of the $P(c_{i,j}|\sigma_i)$.

However, before we discuss learning CCGs, we must complete our discussion of how they are used to represent plans and their use for plan recognition. To combine CCG categories into a parse we use three combinators defined over pairs of categories: leftward and rightward application and rightward composition. We define them as:

$$\textit{Rightward Application: } \frac{X/\alpha \cup \{Y\} \quad Y}{X/\alpha}$$

$$\textit{Leftward Application: } \frac{Y \quad X\backslash\alpha \cup \{Y\}}{X/\alpha}$$

$$\textit{Rightward Composition: } \frac{X/\alpha \cup \{Y\} \quad Y/\beta \cup \{Z\}}{X/\alpha \cup \beta}$$

where α and β represent possibly empty sets of categories. Intuitively, we can think of application and composition combinators as function application and composition.

The above definitions of actions, categories and combinators are extended to a first-order representation by introducing *parameters* for actions and atomic categories to represent domain objects and variables. For a combinator to be used both the atomic category and the parameters of its argument categories must unify. The substitution required to unify the categories is applied to the result category. We refer the reader to (Geib 2016) for a full discussion of action and category parameters.

At this point, an example will help clarify the use of CCGs for plan recognition. Consider the following simple three action CCG for delivering packages using a truck.

CCG: 1

$$f(\textbf{load}(\boldsymbol{P})) \rightarrow \{(\ ld(\boldsymbol{P})\ :1)\}$$
$$f(\textbf{drive}(\boldsymbol{Loc})) \rightarrow$$
$$\{((\ dlv(\boldsymbol{P},\boldsymbol{Loc})/\{unld(\boldsymbol{P})\})\backslash\{ld(\boldsymbol{P})\}:1)\}$$
$$f(\textbf{unload}(\boldsymbol{P})) \rightarrow \{(\ unld(\boldsymbol{P})\ :1)\}$$

The actions **load**(***P***), **unload**(***P***), and **drive**(***Loc***) each have a single parameter capturing either the package being loaded or unloaded, ***P***, or the delivery destination, ***Loc***. Since each action has only a single category, $P(c_{i,j}|\sigma_i) = 1$ for each of them. Further notice that each mapping defines how the action's parameters are used in the category. Note that a category can have parameters that are not bound by its own action, but will be bound by other actions during parsing (consider the category for **drive**(***Loc***)).

Now, consider the sequence of ground action instances:

$$[\ \textbf{load}(p23), \textbf{drive}(l2), \textbf{unload}(p23)\ ].$$

Figure 1 shows the recognition of this sequence of actions as a plan to deliver package, *p23*, to location *l2*, using the given CCG. First, a category is assigned to each action and its parameters bound on the basis of the action. This results in the categories shown in line two of Figure 1. The category $((dlv(\boldsymbol{P},l2))/\{unld(\boldsymbol{P})\})\backslash\{ld(\boldsymbol{P})\}$ requires a $ld(\boldsymbol{P})$ to its left. Binding **P** to $p23$ unifies $ld(\boldsymbol{P})$ with $ld(p23)$ and allows leftward application to produce $dlv(p23,l2)/\{unld(p23)\}$ shown in line three. This category expects a $unld(p23)$ to the right. Therefore, rightward application can be used to produces $dlv(p23,l2)$. Since there are no more category arguments or observed actions, parsing ends with the hypothesis of a single package delivery plan, $dlv(p23,l2)$.

$$\begin{array}{llll}
1) & \textbf{load}(p23) & \textbf{drive}(l2) & \textbf{unload}(p23) \\
2) & \overline{ld(p23)} & \overline{(dlv(\boldsymbol{P},l2)/\{unld(\boldsymbol{P})\})\backslash\{ld(\boldsymbol{P})\}} & \overline{unld(p23)} \\
3) & \multicolumn{2}{c}{\overline{\quad dlv(p23,l2)/\{unld(p23)\}\quad}^{<}} & \\
4) & & \multicolumn{2}{c}{\overline{\quad dlv(p23,l2)\quad}^{>}}
\end{array}$$

Figure 1: Parsing observations with CCGs

In general, a lexicon will produce multiple parses for each set of observed actions. We refer to each such parse as an *explanation*, and note that unlike NLP, a final explanation may contain more than one category, each of which denotes a possibly partial plan the agent is hypothesized to be pursuing in parallel. That said, for learning, we will be assuming that each training instance presented to the system will contain only a single, complete plan instance. Thus, like our example, a full parse of a training instance should yield an explanation with a single atomic category. We will call such explanations *complete*, and any explanation with more than one category or containing complex categories as *incomplete*. Intuitively, if parsing only produces incomplete explanations then either we have observed only a plan fragment, or our grammar needs to be extended.

While our example does not show it, ELEXIR does compute a conditional probability for each explanation given the observed actions, and we refer the reader to (Geib 2009; Geib and Goldman 2011) for a complete discussion of its probability model. This probability is computed by multiplying the conditional probability that each action was initially assigned the category used in the explanation (the $P(c_{i,j}|\sigma_i)$ in the lexicon) by the prior probability of the root results of the categories in the explanation. This work will not learn the priors for root results, and instead assumes these are tunable parameters assigned constant values. However, our learning algorithm will be estimating the values of $P(c_{i,j}|\sigma_i)$ for each action category mapping in the lexicon.

Learning Method

Learning a CCG grammar is just the modification of a lexicon's action to category, conditional probability mapping, (ie. $f(\sigma_i) \rightarrow (c_{i,j} : P(c_{i,j}|\sigma_i))$) to produce more accurate parsing. As such, learning requires two interleaved processes: generating action, category pairs (LexGen), and estimating action, category pair conditional probabilities (ParamEst). We will discuss these processes separately below. However, Algorithm 1 is the high level pseudo code for our learning algorithm, LEX$_{learn}$, where Λ^{cur} refers to the current lexicon.

Algorithm 1 Overall Learning Algorithm

```
procedure LEX$_{learn}$(( $T_i$, $G_i$), $\Lambda^{init}$ )
    Let $\Lambda^{cur}$ = $\Lambda^{init}$
    for i = 1 to n do
        $\Lambda^{cur} = \Lambda^{cur} \cup$ LexGen ($T_i$,$G_i$)
        ParamEst ($T_i$, $G_i$, $\Lambda^{cur}$)
    end for
    Return $\Lambda^{cur}$
end procedure
```

LEX$_{learn}$ takes two inputs, an initial lexicon, Λ^{init}, and a set of training pairs, { (T_i, G_i) : i = 1 ... n }, where each T_i denotes a *plan trace*, a sequence of action observations, $\sigma_1 ... \sigma_m$, that achieve a goal state, G_i. LEX$_{learn}$ assumes that each each T_i results in its respective G_i although the trace may be noisy which we will discuss shortly. The initial lexicon, Λ^{init}, assigns a single atomic category to each action. This atomic category's parameters are identical to those of its action. This has the effect of limiting the prior knowledge of the domain to the available actions and their parameters.

Generating Action Category Pairs (LexGen)

The core of generating action category pairs is the addition of new complex categories to an individual action's entry in the lexicon. However, ELEXIR does not support the full space of potential complex categories. Therefore, the complex categories LEX$_{learn}$ considers for addition are limited in two ways to produce useable ELEXIR lexicons. First, ELEXIR requires that all complex categories are *leftward applicable*, that is, all leftward arguments to complex categories are "outside" (must be discharged by application or composition before) any rightward-looking categories. Second, ELEXIR does not fully support the use of complex categories as arguments to other complex categories. Therefore, LEX$_{learn}$ only considers leftward applicable categories with atomic categories for arguments.

LEX$_{learn}$ places one further requirement on complex categories considered for addition to the lexicon. New categories can have no more than two argument categories to the left or right. This limitation biases LEX$_{learn}$ to abstract frequent subsequences and learn hierarchical plan structures. If the number of argument categories is unbound, lexicons can be learned that have a single complex category for each observed plan instance that has an argument for each action in the observation sequence. This effectively reduces the lexicon to a lookup table of possible plans rather than a generative grammar. While this can be effective for very simple domains, it requires no abstraction, and will not scale to real world sized domains. Abstracting common action subsequences into sub-plans that can be reused and result in more compact lexicons is preferred. Therefore we limit new LEX$_{learn}$ categories to at most two learned argument categories, and leave exploring larger limits to future work.

Given these three limitations, all possible new categories are captured in one of following five category templates:

$$Template_{cats} = \begin{Bmatrix} C_{new}\backslash\{C_x\} \\ C_{new}/\{C_y\} \\ C_{new}/\{C_x\}/\{C_y\} \\ C_{new}/\{C_x\}\backslash\{C_y\} \\ C_{new}\backslash\{C_x\}\backslash\{C_y\} \end{Bmatrix}$$

where C_{new} is either the atomic category representing the trace's goal, G_i, or a new atomic category introduced to represent a potential common sub-sequence. C_x and C_y are existing atomic categories in the lexicon other than G_i.

These templates are used to construct new complex categories for each action. For each action, the algorithm considers the set of all categories consistent with at least one of the templates that produce a yield of actions consistent with the observed T_i and Λ^{cur}. For each C_{new} generated by this process, the system defines its parameters as the union of the current action's parameters and the new category's argument's parameters. If the action and argument categories have parameters bound to the same object in the domain, the newly created category is given only a single parameter for this object. This effectively requires that in the future these action and argument parameters must always be bound to the same object. Note that if another trace does not have this co-reference a new category with a larger set of parameters will be considered. Thus the algorithm will consider both possibilities for the lexicon when evidence is provided.

Initial experiments showed that actions that occurred more than once in a trace, frequently converged to an atomic category as the most likely. This is consistent with the intuition from information theory that more frequently occurring observations convey less information(Shannon 1951). Further, considering all possible categories for such actions resulted in a significant reduction in the speed of the learner. Therefore, to improve the algorithm's runtime, for actions that are observed more than once, action category pairs are only generated for a single randomly chosen instance of the action in each trace. Assuming the action occurs often enough in a large number of traces, this process will converge to the full set of action category pairs for these actions while reducing the processing load for each individual trace.

Finally, this process computes an initial conditional probability, K, for the new hypothesized action category pairs before adding them to Λ^{cur}. Let Λ denote the set of new, unique action category pairs and $\Lambda(\sigma_i)$ denote the set pairs in Λ for $\sigma_i \in T_i$. Also, let M be number of unique pairs in $\Lambda(\sigma_i) \cup \Lambda^{cur}(\sigma_i)$. We define the conditional probability of each category in the grammar as:

$$K_{\sigma_i, c_{i,j}} = \begin{cases} (1 - \frac{|\Lambda(\sigma_i)|}{M}) * P(c_{i,j}|\sigma_i) & \sigma_i := c_{i,j} \in \Lambda^{cur} \\ \frac{1}{M} & \sigma_i := c_{i,j} \text{ is new} \end{cases}$$

Critically, this formula allows us to maintain the probability distribution associated with the categories for an action from Λ^{cur} while adding a new action category pair.

A small example will illustrate the whole process of generating action category pairs for the **drive**() action in the earlier delivery example. Consider the initial lexicon, Λ^{init}, for the three domain actions:

CCG: 2

$$f(\textbf{load}(\boldsymbol{P})) \rightarrow \{(\ ld(\boldsymbol{P})\ : 1)\}$$
$$f(\textbf{drive}(\boldsymbol{Loc})) \rightarrow \{(\ drive(\boldsymbol{Loc})\ : 1)\}$$
$$f(\textbf{unload}(\boldsymbol{P})) \rightarrow \{(\ unld(\boldsymbol{P})\ : 1)\}$$

Next, consider again the simple plan trace T_i made up of three observable actions,

[**load**(*p23*) , **drive**(*l2*), **unload**(*p23*)],

and the goal of delivering *p23* to *l2*, *dlv(p23,l2)*.

First, when considering the action **drive**(*l2*), we choose the set of category templates consistent with the order of actions in T_i:

$$\begin{Bmatrix} C_{new}\backslash\{C_x\} \\ C_{new}/\{C_y\} \\ C_{new}/\{C_x\}\backslash\{C_y\} \end{Bmatrix}$$

Note that we excluded $C_{new}\backslash\{C_x\}\backslash\{C_y\}$ and $C_{new}/\{C_x\}/\{C_y\}$ because these templates require either two actions to the left or right of **drive**(*l2*) and are therefore inconsistent with the trace.

Second, for **drive**(*l2*) we construct the new action category pairs consistent with these templates, and compute initial conditional probabilities for them. There are three such possible new pairs, one using each of the above templates, shown in CCG:3, resulting in a total of four action category pairs for **drive**(*l2*). Therefore, $M = |\Lambda(\sigma_i) \cup \Lambda^{cur}(\sigma_i)| = 4$, and all action category pairs for **drive**(*l2*) will have an initial probability of 0.25. The resulting grammar is shown in CCG:3 where Cat_1 and Cat_2 are new root categories created during the action category generation process.

CCG: 3

$f(\textbf{load}(\boldsymbol{P})) \rightarrow \{(\ ld(\boldsymbol{P})\ :1)\}$

$f(\textbf{drive}(\boldsymbol{Loc})) \rightarrow \{(\ drive(\boldsymbol{Loc})\ :0.25),$

$((\ dlv(\boldsymbol{P},\boldsymbol{Loc})\ /\{\ unld(\boldsymbol{P})\ \})\backslash\{ld(\boldsymbol{P})\} : 0.25)\}$

$((Cat_1(\boldsymbol{P},\boldsymbol{Loc})\backslash\{ld(\boldsymbol{P})\} : 0.25)\}$

$((Cat_2(\boldsymbol{P},\boldsymbol{Loc})/\{unld(\boldsymbol{P})\}) : 0.25)\}$

$f(\textbf{unload}(\boldsymbol{P})) \rightarrow \{(\ unld(\boldsymbol{P})\ :1)\}$

Notice that the parameters for the category dlv, Cat_1, and Cat_2 are drawn from both **drive**() and the argument categories for each.

Estimating Parameters (ParamEst)

We now look at the process of parameter estimation. LEX$_{learn}$ follows (Zettlemoyer and Collins 2005) by using gradient ascent to estimate action, category conditional probabilities. Given a plan trace, T_i, and lexicon, Λ^{cur}, we want to update the probability distributions of action, category pairs in Λ^{cur} for actions that occur in T_i. To do this, we use ELEXIR with Λ^{cur} on the current plan trace to produce a set of explanations. Since each trace is known to be complete, we can remove from ELEXIR's output any incomplete explanations or explanations that do not result in G_i. We denote this set as EXP$'_{T_i,\Lambda^{cur}}$. We update the probabilities in Λ^{cur} on the basis of this filtered set of explanations.

For each action category pair that occurs in EXP$'_{T_i,\Lambda^{cur}}$, we update its probability based on the probability of the other categories associated with the action. Let $(\sigma_j, c_{j,k})$ be an action category pair, and let $E \subseteq$ EXP$'_{T_i,\Lambda^{cur}}$ be the set of all explanations in which $(\sigma_j, c_{j,k})$ occurs. We compute $\theta'_{\sigma_j,c_{j,k}}$, the probability of the category for a given trace, as:

$$\theta'_{\sigma_j,c_{j,k}} = \frac{\Sigma_{e\in E} P(e) * f_e(\sigma_j, c_{j,k})}{\Sigma_{c\in\Lambda^{cur}(\sigma_j)} \Sigma_{e\in E} P(e) * f_e(\sigma_j, c)}$$

where $f_e(\sigma_j, c_{j,k})$ denotes the frequency with which $c_{j,k}$ is assigned as the category for σ_j in explanation e, and $P(e)$ denotes probability of the explanation. Note the denominator normalizes this number by summing across all of the categories σ_j could be assigned. The action category pair's conditional probability can now be updated using:

$$\theta_{\sigma_j,c_{j,k}} = \theta_{\sigma_j,c_{j,k}} + \alpha * (\theta'_{\sigma_j,c_{j,k}} - \theta_{\sigma_j,c_{j,k}})^2$$

where the parameter α controls the speed of learning. Notice that $\theta_{\sigma_j,c_{j,k}}$ is an unnormalized estimate of the conditional probability. We therefore compute the new conditional probability by normalizing the estimated probabilities:

$$P(c_{j,k}|\sigma_j) = \frac{\theta_{\sigma_j,c_{j,k}}}{\Sigma_{c\in\Lambda^{cur}(\sigma_j)} \theta_{\sigma_j,c}}$$

To prevent over-population of action category pairs in the lexicon, and the production of large number of incomplete explanations, we next remove pairs with a conditional probability below a threshold, τ. Note that only action category pairs with complex categories can be pruned. Atomic categories with sub-τ probabilities are left in the lexicon to guarantee that there is always at least one category for every action. Since this thresholding may cause the summation of an action's category probabilities to be less than one, we re-normalize the probabilities one final time.

Empirical Evaluation

The objective of this work was to learn plan CCGs for use in ELEXIR, and our experiments use this system for evaluation. We also considered translating our grammars into the forms used by other plan recognition systems to evaluate them. However, this presented two problems. First, this translation would result in a loss of learned domain information for non-hierarchal systems resulting in an "apples to oranges" comparison. Second, we were concerned that translation of the grammars to other representations would open our evaluation to questions concerning the accuracy and optimality of the translation. Thus, given our objective, our evaluations of the learned grammars only use ELEXIR.

We ran LEX$_{learn}$ on five domains, four of which are from the international planning competition: *Rovers*, *Depots*, *Logistics*, and *Satellites*. The fifth is the Monroe domain, a disaster management domain(Blaylock and Allen 2005). In the rest of this section we will first provide a brief description of each domain and define our metrics, experimental setup, and results. Finally, we describe the structural differences between LEX$_{learn}$ and HTN-Maker's (Hogg, Muñoz Avila, and Kuter 2008) learned representations.

The rovers domain captures plans for an autonomous rover executing data-gathering missions that require gathering and transmitting one of three types of data to a lander: rock, soil, and image. Thus, the rover has three distinct objectives, each with a goal state that we want to learn: *rockMissionComplete*, *soilMissionComplete*, and *imageMissionComplete*. The Depots domain describes plans of an agent whose task is to move cargo from one location to another. The agent accomplishes its tasks by loading a crate onto a truck, driving from one location to another, and unloading the crate at a destination. In our setup, there are two locations, depot and distributor. Thus, there are four different goal states to learn plans for, each corresponding to where the truck travels: *driveFromDepotToDepot*, *driveFromDepotToDistributor*, *driveFromDistributorToDepot*, and *driveFromDistributorToDistributor*.

For our experiments, we slightly modified the Depots domain. The original description of Depots had a single action for driving from a starting location to destination. This meant that the only evidence for the particular goal was in the bound value of the destination parameter of the drive action. This made it impossible for the learner to distinguish between traces for the different goal states, even though it was able to correctly learn the parameters for the complex categories. Therefore, we added four distinct drive actions to the domain description, **driveToXFromY**, where X and Y are distinct locations in our domain.

The Logistics and Satellites domains are based on the HTN domains used by HTN-Maker. Logistics domain captures plans for moving packages from one city to another using trucks and planes. Satellites domain describes plans for capturing images of a given point-of-interest from a satellite. The satellite is required to calibrate its imaging device and reposition itself towards the point-of-interest to capture images. We generate plan traces for both domains by solving a set of one-hundred planning definitions using the HTN-Solver2 planner (Nau et al. 1999). To shorten the learners runtime to allow for more experiments, plan traces were limited to those with a maximum of six actions. However, this reduced the number of available plan traces. We increased the number of traces in two ways. First, we restricted the planning definitions for Logistics to a single goal. Satellites was not restricted in this way. Second, for both domains, we duplicated each of the plan traces in the dataset fifty times. This had the knock-on effect that with high probability, plan instances in the testing dataset were seen during training.

The Monroe domain captures plans for disaster relief including, providing medical aid, plowing snow, and clearing debris from roads. We base our domain off the publicly-available Monroe dataset generated by the SHOP2 planner (Nau et al. 2003). The goals in the Monroe dataset include *plowRoad*, *clearRoadHazard*, and *setUpShelter*. Like the Logistics and Satellites domains, we reduced the original dataset of 5000 plan traces to run more experiments.

Our learner has four tunable parameters. The parameter τ, used to prune low probability categories, is set to 0.0625 for all our experiments. This value was determined by the maximum number of categories generated per action for a plan trace size of 6. The second parameter for our experiments is the prior root probability of all atomic categories in the lexicon. Recall that the probability of an explanation is computed using this prior probability. For all experiments, we set the prior root probabilities for all atomic categories and goal category to 0.01 and 0.99, respectively. This strongly biases the system to learn grammars that have the goals. The third parameter is the number of iterations for gradient ascent, MaxGA, which is set to five, and finally, the learning parameter, α, was set to 0.001 based on initial experiments.

We use two metrics for evaluating the learned grammars. The first metric, following (Zhuo, Muñoz-Avila, and Yang 2014) is the F1 Score:

$$F1 = 2 * \frac{\mathit{Precision} * \mathit{Recall}}{\mathit{Precision} + \mathit{Recall}}$$

where precision and recall (Zettlemoyer and Collins 2005) adapted for this domain are:

$$\mathit{Precision} = \frac{\textit{\# of correct explanations}}{\textit{\# of parsed plan traces}}$$

$$\mathit{Recall} = \frac{\textit{\# of correct explanations}}{\textit{\# of plan traces}}$$

The second metric is Mean-Time-To-Recognition (MTTR) or the average percentage of actions in a trace required to recognize the goal. We formalize MTTR as follows. The conditional probability (CP) of an atomic category, G, for a trace is defined as:

$$CP_G = \frac{\sum_{e \in E} P(e)}{\sum_{e' \in E'} P(e')}$$

where E is the set of explanations such that G is the root result of at least one of the categories in the explanation, and E' is the set of all explanations for the plan. Next, we define Time-To-Recognition (TTR) as the percentage of the plan trace before which the CP of the goal category is higher

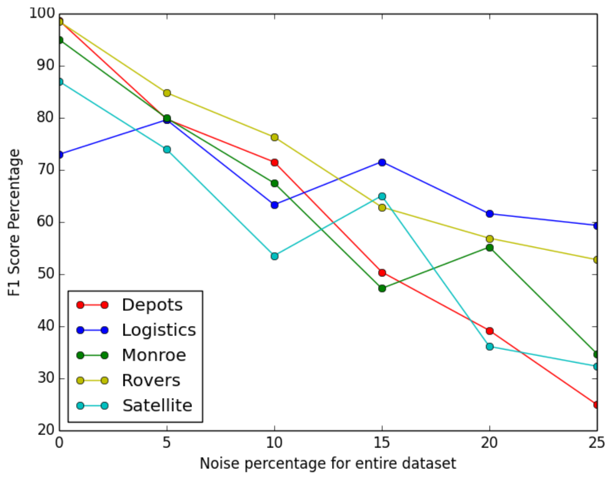


Figure 2: F1 scores for multiple domains as noise increases.

than any other category in the domain and remains so for the duration of the trace. Using TTR, we define MTTR as:

$$\text{MTTR} = \frac{1}{|T|} \sum_{t \in T} t_{TTR}$$

where T is a set of correctly pared plan traces, and t_{TTR} is Time-To-Recognition for the trace t.

To evaluate MTTR values, we consider another statistic, Average Goal Location (AGL) in the highest probability parse. We define the goal location as the percentage of the trace that occurs before the action with G_i as the root result of its category. AGL is computed in the same way as MTTR, with goal location t_{GL} replacing t_{TTR}.

For each domain we ran LEX$_{learn}$ five times each time simulating different levels of noise in the plan traces. We tested noise percentages ranging from zero to twenty five percent in increments of five percent. Each run's dataset was generated by randomly dropping actions from each trace based on the given noise percentage and then shuffling the order of the complete plan traces. F1 scores were then averaged over each noise level's five runs (Figure 2). MTTR and AGL were averaged only over the noise free level's five runs (Table 3). For all learning runs, we split the plan traces into 80% training and 20% testing instances. Results in Figure 2 and Table 3 were generated by training our learner on the training instances, and testing on the testing instances.

Looking at F1 scores for each domain across different noise levels, we expected an increase in noise would significantly reduce the F1 score, and most of the domains follow this trend. Surprisingly, logistics remained almost constant as noise increased. Looking at the results of the learning algorithm and plan recognition suggest an explanation for this. First, the noise free F1 score was the lowest tested. Looking at the traces from this dataset revealed that it contained several traces which contained complete plans as a subset. Thus, when parsing the complete trace, the subset trace was parsed to a goal state which led to parsing failure for the rest of the plan in the noise free setting. However, this unintentional redundancy in the plan trace may have forced the learning

Domains	MTTR	AGL
Logistics	85.8%	87.7%
Rovers	56.8%	54.5%
Satellites	82.0%	65.8%
Monroe	93.8%	95.1%
Depots	87.1%	62.2%

Figure 3: MTTR and AGL values for various domains

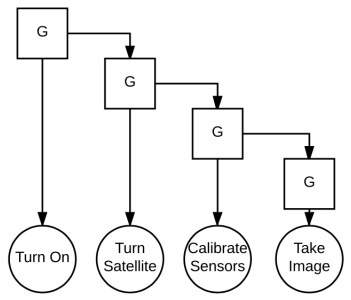
of a larger number of categories allowing the grammar to be more resilient to missing observations. In fact, the grammars learned using noisy traces contained action category pairs with high probabilities, which implies the learner saw similar structures between noisy and noiseless plans confirming our hypothesis, and explaining how ELEXIR was able to correctly parse these noisy plan traces.

Having established the accuracy of the learned grammars even in relatively noisy domains, we turn to the MTTR and AGL values in Table 3. Low MTTR values are preferred. The lower the value, the earlier the system recognizes the correct plan, and the sooner one might be able to act on that knowledge. The relatively high values for the various domains is initially disappointing, but reflect a number of different issues. First with the shorter plans, effects on MTTR are pronounced. Adding even one action to the MTTR for a five step plan raises the MTTR by twenty percent.

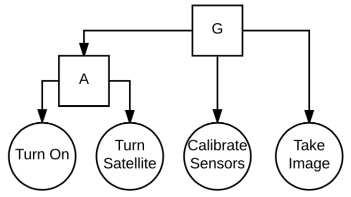
Second, ELEXIR is unable to recognize plans for a given category until an action with that category as its root result is observed in the trace. AGL values measure when on average such an action occurs in each trace. Note that for all but two domains the MTTR and AGL values are very close, indicating that ELEXIR is recognizing the correct plans within a very small window of when it is first able to do so. However, there was a larger difference between MTTR and AGL for both Depots and Satellites. We believe this was the result of ambiguity in the learned grammars and are exploring if the cause of this is endemic to the domain or could be fixed by a change in the learning parameters. Finally, (Geib 2009) has shown that multiple plan CCGs with the same yield can have a significant differences in MTTR. MTTR is dependent both on the complex categories added to the grammar as well as the parameter model and its estimation process. As a result, biasing the learning algorithm to produce grammars with lower MTTR is an exciting area for future work.

Comparison to HTN-Maker

Finally, we briefly compare the structures learned between LEX$_{learn}$ and HTN-Maker. While these systems share a very similar goal their resulting structures are very different. Figure 4 illustrates the different representations learned for the plan **[Turn On, Turn Satellite, Calibrate Sensors, and Take Image]** from the Satellites domain. HTN-Maker was designed to generate right-recursive structures shown in Figure 4a As such, HTN-Maker is unable to efficiently capture frequently occurring sub-sequences of actions that are not right-recursive as separate plan sub-structures in its representation. This prevents HTN-Maker from finding some useful abstractions within the traces and can therefore result



(a) HTN-Maker representation



(b) LexLearn representation

Figure 4: Representations from HTN-Maker and LEX$_{learn}$

in larger representations with less generality than LEX$_{learn}$. For example, LEX$_{learn}$ was able to find such common substructures in the Satellites domain. Figure 4b shows a HTN-Maker style parse tree where LEX$_{learn}$was able to identify the action sequence **[Turn On, Turn Satellite]** as a common sub-sequence of traces in the Satellites domain and learn a single complex category for its recognition.

Conclusion and Future Work

This paper presented LEX$_{learn}$, an incremental supervised learning algorithm for plan CCGs. While encouraging, this work is just a first step toward learning such grammars. More testing needs to be performed to identify parameter values for the algorithm that will guarantee high performance and robustness on larger and more complex domains. Further, there are also a number of issues that we have left for future work. These include: learning larger complex categories, concurrent learning of root result priors needed by ELEXIR, learning more expressive parameter models for categories, biasing learning to produce grammars with lower MTTR values, and learning plan CCG categories for partially ordered plans. That said, we believe these results are very promising and demonstrate that the CCG grammars needed for plan recognition by ELEXIR can be learned.

Acknowledgements

This work was funded by the National Science Foundation under the grant EAGER: Lexicalized Reasoning (Award number: 1650204).

References

Bisk, Y., and Hockenmaier, J. 2012. Simple robust grammar induction with combinatory categorial grammars. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial*

Intelligence, AAAI'12, 1643–1649. Palo Alto, California, USA: AAAI Press.

Bisson, F.; Larochelle, H.; and Kabanza, F. 2015. Using a recursive neural network to learn an agent's decision model for plan recognition. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, 918–924. AAAI Press.

Blaylock, N., and Allen, J. 2005. Generating artificial corpora for plan recognition. In *Proceedings of the 10th International Conference on User Modeling*, UM'05, 179–188. Berlin, Heidelberg: Springer-Verlag.

Curry, H. 1977. *Foundations of Mathematical Logic*. Dover Publications Inc.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. UMCP: A sound and complete procedure for hierarchical task network planning. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS 94)*, 249–254.

Geib, C. W., and Goldman, R. P. 2011. Recognizing plans with loops represented in a lexicalized grammar. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, AAAI'11, 958–963. Palo Alto, California, USA: AAAI Press.

Geib, C. W. 2009. Delaying commitment in plan recognition using combinatory categorial grammars. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, 1702–1707. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Geib, C. W. 2016. Lexicalized reasoning about actions. *Advances in Cognitive Systems* Volume 4:187–206.

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 - Volume 2*, AAAI'08, 950–956. AAAI Press.

Kwiatkowski, T.; Zettlemoyer, L.; Goldwater, S.; and Steedman, M. 2010. Inducing probabilistic ccg grammars from logical form with higher-order unification. In *Proceedings of the 2010 conference on empirical methods in natural language processing*, EMNLP '10, 1223–1233. Stroudsburg, PA, USA: Association for Computational Linguistics.

Kwiatkowski, T.; Goldwater, S.; Zettlemoyer, L.; and Steedman, M. 2012. A probabilistic model of syntactic and semantic acquisition from child-directed utterances and their meanings. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '12, 234–244. Stroudsburg, PA, USA: Association for Computational Linguistics.

Li, N.; Cushing, W.; Kambhampati, S.; and Yoon, S. 2014. Learning probabilistic hierarchical task networks as probabilistic context-free grammars to capture user preferences. *ACM Trans. Intell. Syst. Technol.* 5(2):29:1–29:32.

Mourao, K.; Zettlemoyer, L. S.; Petrick, R.; and Steedman, M. 2012. Learning strips operators from noisy and incomplete observations. *arXiv preprint arXiv:1210.4889*.

Nau, D.; Cao, Y.; Lotem, A.; and Munoz-Avila, H. 1999. Shop: Simple hierarchical ordered planner. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'99, 968–973. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Nau, D.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. Shop2: An htn planning system. *Journal of Artificial Intelligence Research* 20:379–404.

Nejati, N.; Langley, P.; and Konik, T. 2006. Learning hierarchical task networks by observation. In *Proceedings of the 23rd international conference on Machine learning*, 665–672. ACM.

Schmidt, C. F.; Sridharan, N.; and Goodson, J. L. 1978. The plan recognition problem: An intersection of psychology and artificial intelligence. *Artificial Intelligence* 11(1-2):45–83.

Shannon, C. E. 1951. Prediction and entropy of printed english. *Bell system technical journal* 30(1):50–64.

Steedman, M. 2001. *The Syntactic Process*. Cambridge, MA, USA: MIT Press.

Thomforde, E., and Steedman, M. 2011. Semi-supervised ccg lexicon extension. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, 1246–1256. Stroudsburg, PA, USA: Association for Computational Linguistics.

Vilain, M. 1990. Getting serious about parsing plans: A grammatical analysis of plan recognition. In *Proceedings of the Eighth National Conference on Artificial Intelligence - Volume 1*, AAAI'90, 190–197. Palo Alto, California, USA: AAAI Press.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted max-sat. *Artificial Intelligence* 171(2):107–143.

Zettlemoyer, L. S., and Collins, M. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *UAI '05, Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence, Edinburgh, Scotland, July 26-29, 2005*, UAI'05, 658–666. AUAI Press.

Zhuo, H. H.; Muñoz-Avila, H.; and Yang, Q. 2014. Learning hierarchical task network domains from partially observed plan traces. *Artificial Intelligence* 212:134 – 157.