# Scaling up CCG-Based Plan Recognition via Monte-Carlo Tree Search

Pavan Kantharaju
*College of Computing and Informatics*
*Drexel University*
Philadelphia, PA, USA
pk398@drexel.edu

Santiago Ontañón*
*College of Computing and Informatics*
*Drexel University*
Philadelphia, PA, USA
pk398@drexel.edu

Christopher W. Geib
*SIFT LLC*
Minneapolis, Minnesota, USA
cgeib@sift.net

***Abstract*—This paper focuses on the problem of scaling Combinatory Categorial Grammar (CCG)-based plan recognition to large CCG representations in the context of Real-Time Strategy (RTS) games. Specifically, we present a technique to scale plan recognition to large domain representations using Monte-Carlo Tree Search (MCTS). CCG-based planning and plan recognition (like other domain-configurable planning frameworks) require domain definitions to be either manually authored or learned from data. Prior work has demonstrated successful learning of these CCG domain definitions from data, but these representations can be very large for complex application domains. We propose a MCTS-based approach to search for explanations and predict the goal of a given sequence of observed actions. We evaluate our approach on the RTS game AI testbed microRTS. Our experimental results show our method scales better to these large, learned CCGs than previous CCG-based approaches.**

***Index Terms*—Plan Recognition, Combinatory Categorial Grammar, Real-Time Strategy Games**

## I. Introduction

Plan recognition focuses on the problem of inferring the goal of some observed sequence of actions [1]. The problem of *plan recognition* has many applications in domains such as robotics and video games. and thus many different algorithms have been developed to address this problem. For example, past work has demonstrated that plan recognition can be viewed as parsing a stream of observations with a grammar that defines the possible plans [2]. In this paper, we focus on using Combinatory Categorial Grammars (CCGs), which have been shown to effectively represent natural language (and more recently, shown to effectively represent and recognize plans in a large number of domains [3], [4]), to perform plan recognition in the context of Real-Time Strategy (RTS) games. Specifically, we present a new approach based on Monte-Carlo Tree Search (MCTS), to scale up to the size of plans that appear in these type of games.

Prior work demonstrated the successful learning of CCG lexicons from training data [5], [6] for the problem of CCG-based plan recognition. However, these learned CCG lexicons can be very large and complex, and standard Breadth-First Search (BFS) approaches for plan recognition will not scale well. For example, consider plan recognition in the domain of Real-Time Strategy (RTS) games where a large number of strategies would have to be encoded into the CCG lexicon (many of these strategies can be hundreds of actions long). If we want to use plan recognition to identify the strategy being deployed by the opponent, this can cause an explosion in the search space. This paper presents a technique to scale CCG-based plan recognition to these large CCG lexicons.

Our main contribution is a Monte-Carlo Tree Search (MCTS)-based plan recognition algorithm that allows us to scale recognition to larger CCG lexicons. Specifically, we apply MCTS to search for a set of explanations and to predict the goal of a sequence of actions given a predefined number of iterations. We evaluate our approach on the Real-Time Strategy (RTS) game AI testbed $\mu$RTS[1]. Our results demonstrate that we can effectively scale plan recognition while maintaining good recognition performance.

This paper is structured as follows. First, we provide background on Combinatory Categorial Grammars (CCGs) and CCG-based plan recognition. Second, we provide related work in the area of plan recognition. Third, we detail our CCG-based MCTS plan recognition algorithm. Fourth, we outline our experimental setup and provide experimental evaluation of our approach. We conclude with remarks for future work.

## II. Background

This section describes a restricted form of Combinatory Categorial Grammars (CCGs), using the definition of CCGs from Geib [3], and defines *CCG-based plan recognition*. Each action in a domain is associated with a set of finite CCG categories $\mathcal{C}$, composed of two types of categories.

*Atomic categories* correspond to a set of indivisible categories A, B, C.. $\in \mathcal{C}$ and can be seen as zero-arity functions. *Complex categories* on the other hand are curried functions [7] and are defined by a finite set of categories of the form Z/X or Z\X, where Z and X can be either atomic or complex categories, and X can also be a set of atomic categories. The operators "\" and "/" each take a set of *arguments* (the categories on the right hand side of the slash, X), and produce the *result* (the category on the left hand side of the slash, Z). The slash operators define ordering constraints for

*Currently at Google

[1]https://github.com/santiontanon/microrts

$$
\begin{aligned}
&f(\textbf{Harvest}(U_1, R_1)) \rightarrow \{\text{Harvest}(U_1, R_1)) : 1\}\\
&f(\textbf{Train}(U_2, T)) \rightarrow\\
&\qquad \{((\text{HeavyRush})/\{\text{Attack}(U_3, U_4)\})\backslash\{\text{Harvest}(U_1, R_1)\} : 1\}\\
&f(\textbf{Attack}(U_3, U_4)) \rightarrow \{\text{Attack}(U_3, U_4)) : 1\}
\end{aligned}
$$

$$\pi = [\ \textbf{Harvest}(Worker_1, Resource_1), \textbf{Train}(Barracks_1, Heavy), \textbf{Attack}(Heavy_1, Base_2)\ ]$$

$$
\begin{array}{llll}
1) & \textbf{Harvest}(Worker_1, Resource_1) & \textbf{Train}(Barracks_1, Heavy) & \textbf{Attack}(Heavy_1, Base_2) \\ \hline
2) & \text{Harvest}(U_1 = Worker_1, R_1 = Resource_1) & ((\text{HeavyRush})/\{\text{Attack}(U_3, U_4)\})\backslash\{\text{Harvest}(U_1, R_1)\} & \text{Attack}(U_3 = Heavy_1, U_4 = Base_2) \\ \hline
3) & \multicolumn{2}{c}{(\text{HeavyRush})/\{\text{Attack}(U_3, U_4)\}} \; < & \\ \hline
4) & & \multicolumn{2}{c}{\text{HeavyRush}} \; > \\
\end{array}
$$

Fig. 1. Sample CCG (top), example plan $\pi$ (middle) and parse of three observed actions for a Heavy Rush (bottom)

plans, indicating where other actions are to be found relative to an action. Those categories associated with the forward slash operator are after the action, and those associated with the backward slash are before it. In summary, categories are functions that take other functions as arguments. We define the *root* of some category G (atomic or complex) if it is the leftmost atomic category in G. For example, for the complex category $((\text{C})\backslash\{\text{A}\})\backslash\{\text{B}\}$, the root would be C.

CCGs are defined by a *plan lexicon*, $\Lambda = \langle \Sigma, \mathcal{C}, f \rangle$, where $\Sigma$ is a finite set of action types, $\mathcal{C}$ is a set of CCG categories (as defined above), and $f$ is a mapping function such that $\forall a \in \Sigma$,

$$f(a) \rightarrow \{c_i : p(c_i|a), ..., c_j : p(c_j|a)\}$$

where $c_i \dots c_j \in \mathcal{C}$ and $\forall a \sum_{k=i}^{j} p(c_k|a) = 1$. This definition implies that action types can have multiple categories associated with it in $\Lambda$. Given a sequence of actions, these probabilities represent the likelihood of assigning a category to an action for plan recognition. For details on these probabilities and how they are used for plan recognition, see Geib [3].

We assume that all complex categories are *leftward applicable* (all arguments with the backward slash operator are discharged before any forward ones), and we only consider complex categories with atomic categories for arguments. We also extend the definitions of action types and atomic categories to a first-order representation by introducing *parameters* to represent domain objects and variables. CCGs with parameterized actions and categories are defined by Geib [8].

Figure 1 provides an example CCG lexicon with parameterized action types and categories for recognizing a heavy rush strategy in $\mu$RTS. Rush strategies consist of quickly constructing many units of a specific type, and having all those units attack the enemy. An example of a rush strategy from the commercial RTS game *Starcraft* is Zerg rush. For this lexicon, $\Sigma = \{$**Harvest**, **Train**, **Attack**$\}$ and $\mathcal{C} = \{$Harvest, Attack, $((\text{HeavyRush})/\{\text{Attack}\})\backslash\{\text{Harvest}\}\}$. The action types **Harvest**$(U_1, R_1)$, **Train**$(U_2, T)$, and **Attack**$(U_3, U_4)$ each have variable parameters representing different RTS units $U_1, U_2, U_3, U_4$, resource $R_1$, and unit type $T$. Since each action type only has a single category, $p(c|a) = 1$.

In CCGs, *combinators* are operators defined over pairs of adjacent categories that are used to parse sequences of tokens. The two most common types of combinators are *application* and *composition*. The application combinator applies an atomic category to a complex category of arity one while composition combines two complex categories:

$$
\begin{aligned}
\textbf{Rightward Application: } & X/Y \quad Y \rightarrow X\\
\textbf{Leftward Application: } & Y \quad X\backslash Y \rightarrow X\\
\textbf{Rightward Composition: } & X/Y \quad Y/Z \rightarrow X/Z
\end{aligned}
$$

Intuitively, we can think of application and composition combinators as functional application and composition.

We now define a few terminology relevant to CCG-based plan recognition. We define $\pi$ as a sequence of observed actions $\sigma_1, \sigma_2, \dots \sigma_k$. Next, we define the *CCG plan recognition problem* as $PR = (\pi, \Lambda, s_0)$, where $\pi$ is a subsequence of observed actions, $\Lambda$ is a CCG lexicon, and $s_0$ is the initial state of the world from where $\pi$ was executed. The solution to the plan recognition problem is a pair $(E, G)$, where $G$ is a list of top-level goals that are being pursued in $\pi$, and $E$ is a list of explanations or hypotheses that define the plan structures in $\pi$. Figure 1 (bottom) shows the recognition of the observed sequence of actions $\pi$ (Figure 1 (middle)), which is a plan for the goal of executing a heavy rush in $\mu$RTS. First, a category is assigned to each action and its parameters bound based on the action ($U_1 = Worker_1, R_1 = Resource_1, U_3 = Heavy_1$, $U_4 = Base_2$). This results in the categories shown in line two of Figure 1. The category $((\text{HeavyRush})/\{\text{Attack}(U_3, U_4)\})\backslash\{\text{Harvest}(U_1, R_1)\}$ requires a Harvest$(U_1, R_1)$ to its left. Binding $U_1$ to $Worker_1$ and $R_1$ to $Resource_1$ unifies Harvest$(U_1 = Worker_1, R_1 = Resource_1)$ with Harvest$(U_1, R_1)$ and allows leftward application to produce $(\text{HeavyRush})/\{\text{Attack}(U_3, U_4)\}$ shown in line 3. This category expects a Attack$(U_3, U_4)$ to the right. Rightward application can be used to produce HeavyRush. Since there are no more category arguments or actions, parsing ends with the hypothesis of the heavy rush plan, HeavyRush.

## III. Related Work

Monte-Carlo Tree Search (MCTS) is a well-known algorithm for game-playing and has been applied to both multiplayer [9], [10] and single player games [11]. For a survey of MCTS algorithms in the literature, see Browne et al. [12]. Our work focuses on applying traditional MCTS for plan recognition for the domain of Real-Time Strategy (RTS)

Games. Prior work applied bayesian programs to represent plan knowledge for plan recognition in RTS games [13]. Our work focuses on representing plan knowledge using CCGs.

The problem of plan recognition can be viewed in many different ways. Some approaches view the problem as a planning problem (known as *inverse-planning*) [14]–[16]. Another way to view the problem is in the form of John McCarthy's circumscription [17]. Specifically, the seminal work by Kautz and Allen [18] viewed plan recognition this way and represented plan knowledge in the form of a plan graph.

Other plan recognition approaches view plan recognition as a form of parsing from Natural Language Processing using a formal grammar that defines plan knowledge. To the best of our knowledge, the first work to provide a correspondence between the formal theory presented by Kautz and Allen [18] and Context-Free Grammars (CFGs) was Vilain [2]. This was done by providing a mapping between plan graphs and CFGs.

Since then, many other grammar formalisms have been used for plan recognition. Probabilistic State-Dependent Grammars, an extension of Probabilistic Context-Free Grammar with state information, have also been used to represent plan knowledge [19]. The plan recognition algorithm PHATT uses plan tree grammars [20] while YAPPR represents plan knowledge using Plan Frontier Fragment Grammars [21], [22]. Our work focuses on using CCGs for plan recognition.

Plan recognition is tangentially related to goal recognition and has also been applied to games. One work applied Input-Output Hidden Markov Models (IOHMM) to recognize player goals from low-level actions in a top-down action adventure game [23]. Other works have applied deep learning techniques for goal recognition, such as stacked denoising autoencoders [24] and Long Short-Term Memory [25], [26].

The work presented in this paper is closely related to the application of CCGs for plan recognition. The Breath-First Search (BFS) CCG-based plan recognition algorithm ELEXIR was first described by Geib [3]. ELEXIR has demonstrated a number of advantages over other work including efficient recognition of partially-ordered plans and multiple interleaved plans. This work was later extended to recognize plans with loops [4].

## IV. Monte-Carlo Tree Search Algorithm

The complexity of CCG-based plan recognition is defined by the number of categories per action type in a CCG lexicon, and the number of actions in an observed action sequence. As the number of categories per action increases, the branching factor from search increases. Coupled with a large number of actions in an observed action sequence, the search space for plan recognition can significantly increase. As a result, Breath-First Search (BFS) plan recognition does not scale well for large CCG lexicons. Prior CCG learning algorithms like $Lex_{Greedy}$ were successfully able to scale learning to long plans. However, the learned CCG lexicons were too large for BFS plan recognition. Our solution is the development of an anytime approximation algorithm plan recognition algorithm.

```
procedure MCTS($\pi, \Lambda, N, \mathcal{G}$)
    $T = root$
    for $iter = 1 \ldots N$ do
        $n = selection(T)$
        if $n_E$ explains $\pi$ then
            $backpropagate(T, n, p(n_E))$
        else
            if $n_v = 0$ then
                $reward = simulation(n)$
                $backpropagate(T, n, reward)$
            else
                $n' = expansion(n)$
                if $n' = nil$ then
                    $backpropagate(T, n', 0)$
                else
                    $reward = simulation(n')$
                    $backpropagate(T, n', reward)$
                end if
            end if
        end if
    end for
end procedure
```

Fig. 2. MCTS CCG-based Plan Recognition

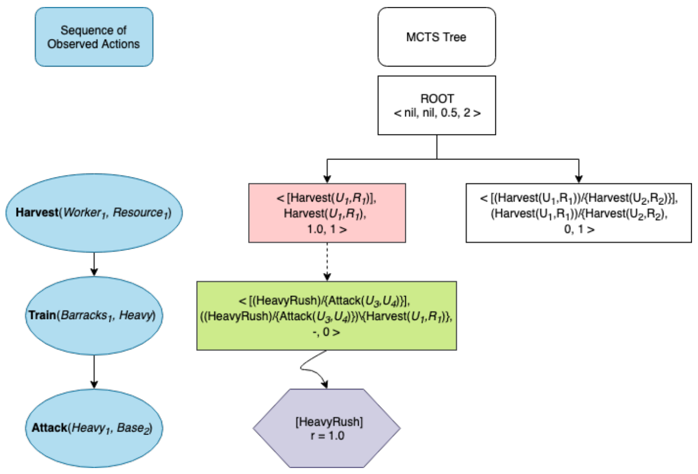
```
procedure SELECTION($T$)
    $n = T_{root}$
    $N = n_{children}$
    while $N \neq \emptyset$ do
        if $n$ can have more children then
            return $n$
        else
            $s \sim U(0, 1)$
            if $s > \epsilon$ then
                $N' = argmax_{n' \in N} n'_r$
                $N'' = argmax_{n' \in N'} n'_v$
                return $n' \sim U(N'')$
            else
                return $n' \sim U(N)$
            end if
        end if
    end while
end procedure
```

Fig. 3. MCTS Selection Function

Specifically, we employ Monte-Carlo Tree Search to recognize sequences of observed actions.

Figure 2 provides high-level pseudocode of our CCG-based Monte-Carlo Tree Search (MCTS) plan recognition algorithm. The MCTS algorithm takes as input a sequence of observed actions $\pi = \langle \sigma_1, ..., \sigma_k \rangle$, a CCG lexicon $\Lambda$, the maximum number of iterations for search $N$, and a set of query goals $\mathcal{G}$ (goals that we want to recognize). We will describe how to compute $p(g|\pi)$ ($g \in \mathcal{G}$) after describing the MCTS algorithm. In our description of the MCTS approach, we use $U$ to refer to a uniform distribution and $x \sim U(X)$ or $k \sim U(i, j)$ as sampling from a set $X$ from the interval $[i \ldots j]$.

Our algorithm starts by initializing a search tree $T$ with a single root node. Each node in the search tree is defined as $n = \langle E, c_i, r, v \rangle$, where $E$ is an explanation of $\sigma_1 \ldots \sigma_i \in \pi$ (where $i$ is the depth of the node and $1 \leq i \leq k$), $c_i$ is the


Sequence of Observed Actions
MCTS Tree
ROOT < nil, nil, 0.5, 2 >
< [Harvest(U1,R1)], Harvest(U1,R1), 1.0, 1 >
< [(Harvest(U1,R1))/{Harvest(U2,R2)}], (Harvest(U1,R1))/{Harvest(U2,R2), 0, 1 >
< [(HeavyRush)/{Attack(U3,U4)}], ((HeavyRush)/{Attack(U3,U4)})\{Harvest(U1,R1)}, -, 0 >
[HeavyRush] r = 1.0
Harvest(Worker1, Resource1)
Train(Barracks1, Heavy)
Attack(Heavy1, Base2)

(a) Example Execution

$$
\begin{aligned}
&f(\textbf{Harvest}(U_1, R_1)) \rightarrow \{\text{Harvest}(U_1, R_1)) : 0.5,\\
&\quad (\text{Harvest}(U_1, R_1))/\{\text{Harvest}(U_2, R_2)\} : 0.5\}\\
&f(\textbf{Train}(U_2, T)) \rightarrow\\
&\quad \{((\text{HeavyRush})/\{\text{Attack}(U_3, U_4)\})\backslash\{\text{Harvest}(U_1, R_1)\} : 1\}\\
&f(\textbf{Attack}(U_3, U_4)) \rightarrow \{\text{Attack}(U_3, U_4)) : 1\}
\end{aligned}
$$

(b) CCG Lexicon

Fig. 4. Example of MCTS Execution (Left) with CCG (Right)

```
 1: procedure SIMULATION($n$)
 2:     Let $E = n_E$ (explanation in $n$ for $\sigma_1 \ldots \sigma_i$)
 3:     for $\sigma_l \in \sigma_{i+1} \ldots \sigma_k$ do
 4:         Let $a$ be the action type of $\sigma_l$
 5:         $\mathcal{C} = valid(\Lambda_{f(a)})$
 6:         $c \sim U(\mathcal{C})$
 7:         $E' = extend(E, c)$
 8:         $E \sim U(E')$
 9:     end for
10: end procedure
```

Fig. 5. MCTS Simulation Function

category assigned to an observed action $\sigma_i$, $r$ is the reward, and $v$ is the number of times the node was visited. The root node (depth 0) is initialized to $root = \langle nil, nil, -, 0\rangle$.

Each edge in the search tree represents the extension of an explanation $E$. Explanations define plan structures found in the observed actions $\pi$ and are represented as a list of atomic or complex categories. An explanation with multiple categories means that the agent is pursuing multiple, potentially interleaved plans. Atomic categories in explanations define top-level plans that the agent is pursuing while complex categories mean that one of the plans in $\pi$ is incomplete, and more actions are needed to complete that plan. The extension method for an explanation used by MCTS is similar to the one used by the ELEXIR plan recognition algorithm [3]. An explanation $E$ for $\sigma_1 \ldots \sigma_{i-1}$ is extended by assigning a category $c$ to an observed action $\sigma_i \in \pi$, and adding it to $E$. Next, known CCG combinators (defined in Section II) are used to parse these categories which results in a new explanation (list of potentially new atomic or complex categories). See Figure 4a for an example of a few explanations (explanations are closed in brackets "[" and "]").

Our approach follows the standard approach to MCTS: *selection*, *expansion*, *simulation*, and *back-propagation*. We look at each function in detail. First, selection chooses the next node in the search tree to expand using a *tree policy*. Our approach uses $\epsilon$-greedy as the tree policy. Algorithm 3 provides pseudocode for the selection function. We refer to $n_{children}$ as the set of node $n$'s children. First, the function checks to see if the current node $n$ can have more children (lines 5-6 of Figure 3). If all children nodes have been constructed for $n$, then the $\epsilon$-greedy policy selects the next node based on the highest reward and times visited (lines 8-15 of Figure 3).

If the selected node $n$ (line 4 of Figure 2) has an explanation $n_E$ for $\pi$ (i.e. node is a leaf), there is no need to expand that node and thus, MCTS back-propagates the reward $p(n_E)$ (line 4-5 of Figure 2). This reward is the probability of the explanation $n_E$, and will be explained below.

Second, *expansion* constructs successor nodes $\mathcal{N}$ given the selected node $n = \langle E, c_{i-1}, r, v\rangle$ from the selection function. Intuitively, expansion extends the explanation $n_E$ for $\sigma_1 \ldots \sigma_{i-1}$ given the next observed action $\sigma_i$ and one of its mapped categories in the lexicon $\Lambda$. First, expansion gets a list of valid categories $\mathcal{C}$ (categories whose leftward arguments can be parsed) for an observed action $\sigma_i \in \pi$. Expansion then selects a category $c'_i \in \mathcal{C}$ with a uniform weighted distribution based on the category's conditional probabilities $p(c'_i|a)$, where $a$ is the action type of $\sigma_i$. Next, expansion extends the current explanation $E$ with $c'_i$, resulting in a set of explanations $E'$. For each explanation $e \in E'$, the expansion function constructs a new node $n' = \langle e, c'_i, -, 0\rangle$ and adds it to the set of $n$'s children. The last constructed node is then returned for simulation.

We note that nodes not returned for simulation will not be part of the search tree until later. These nodes will have their times visited set to 0 until they have the opportunity to be placed into the search tree when the $\epsilon$-greedy tree policy selects them during selection (line 14 of Figure 3). If the selection function chooses one of these nodes, then MCTS directly simulates and back-propagates from the node as it was already constructed during expansion (lines 8-10 in Figure 2). If there are no valid categories or no explanations are generated by extending $E$ with $c_i'$, then expansion fails (returns *nil*) and a reward of 0 is back-propagated (lines 13-14 of Algorithm 2).

Third, *simulation* "plays" out plan recognition from node $n' = \langle E, c_i', -, 0 \rangle$ using a random playout policy. We chose a random playout policy because it worked well in our experiments. Figure 5 provides pseudocode for the simulation function. Given an observed action $\sigma_l \in \sigma_{i+1} \ldots \sigma_k$ (where $k$ is the index of the last observed action, and $i+1 \leq l \leq k$), simulation first retrieves a list of valid categories $\mathcal{C}$ for $\sigma_{i+1}$ from the set of all mapped categories $\Lambda_{f(a)}$, where $a$ is the action type of $\sigma_{i+1}$, and samples a category $c \in \mathcal{C}$ uniformly at random (lines 4-6). Next, simulation extends $E'$ with $c$, gets a list of new explanations, and samples an explanation uniformly at random (lines 7-8). Once the simulation successfully plays out plan recognition, a numerical reward, computed as the probability of the explanation at the leaf node $p(E)$, is returned. This probability is computed in the same manner as that described by Geib [3]. Intuitively, this probability represents how well the explanation describes the observed actions $\pi$. A higher probability explanation means that this is a more likely description of the plans being executed in $\pi$. Finally, *back-propagation* updates the statistics (reward $r$ and number of times visited $v$) along the path from $n$ to the root. For each ancestor $n = \langle E, c_i, r, v \rangle$ on the path, back-propagation updates $r$ and normalizes the result, and increments $v$.

Plan recognition entails finding a set of the most probable top-level goals being pursued $G \in \mathcal{G}$ and a set of explanations $E$ for an observation sequence $\pi$. In this work, we assume $\pi$ is pursuing a single goal $g$ (i.e. $|G| = 1$). The result of MCTS is a tree $T$ where the leaves are explanations for $\pi$. Thus, $E$ is defined as the leaves of $T$. Given $E$, we can compute $p(g|\pi)$ (conditional probability that $\pi$ is pursuing $g$) in the same manner as ELEXIR [3],

$$p(g|\pi) = \frac{\sum_{e \in E_g} p(e)}{\sum_{e \in E} p(e)}$$

where $E_g$ is the set of explanations where $g$ is the root of at least one of the categories in the explanation.

Figure 4a provides the execution of a single iteration of MCTS given the CCG lexicon in Figure 4b and the sequence of observed actions $\pi$ from Figure 1. For simplicity, we provide these actions as blue circles in Figure 4a (first action is the top circle). First, the selection function chooses a node in the tree without children (red box): $n = \langle [\text{Harvest}(U_1, R_1)], \text{Harvest}(U_1, R_1), 1.0, 1 \rangle$. Second, the expansion function constructs a new node $n'$ by extending explanation $n_E = [\text{Harvest}(U_1, R_1)]$ (green box). Third, simulation plays out plan recognition to get an explanation for $\pi$ (purple hexagon): $[HeavyRush], r = 1.0$. Finally, the reward of 1 is back-propagated through the ancestor nodes ($n_r = 2.0, n_r' = 1.0, root_r = 0.66$, and $n_v = 2, n_v' = 1, root_v = 3$).

## V. Experimental Evaluation

The purpose of MCTS plan recognition is to scale search to large CCG lexicons while maintaining good performance. To this end, we run two different experiments to test the effectiveness of MCTS plan recognition. First, we analyze the performance of MCTS to understand how well MCTS can successfully recognize sequences of observed actions. Second, we analyze the scalability of the search tree against observation length. For each experiment, we compare against a Breath-First Search (BFS) plan recognition algorithm in ELEXIR [3]. Both approaches are evaluated on the $\mu$RTS domain. $\mu$RTS is a minimalistic RTS game designed to evaluate AI research in an RTS setting [27]. Compared to commercial RTS games such as *StarCraft*, $\mu$RTS maintains those properties of RTS games that make them complex from an AI point of view (i.e. durative and simultaneous actions, real-time combat, large branching factors, and full or partial observability). This work focuses on recognizing strategies in deterministic, fully observable games.

All CCGs are constructed using the CCG learning algorithm $Lex_{Greedy}$ [6]. $Lex_{Greedy}$ requires an initial CCG lexicon $\Lambda_{init}$, and a training dataset consisting of sequences of actions (called *plan traces*). We use a collection of scripted game agents to generate learning datasets and $\Lambda_{init}$. The process is described below.

Plan traces are constructed using *replay* data from gameplay of the scripted agents. A *replay* is a trace of a game represented as a sequence of state, player action tuples containing the evolution of the game state as well as the actions performed by each player during the game. We construct a replay dataset was created by running a 5-iteration round-robin tournament using the following built-in scripted agents: *POLightRush*, *POHeavyRush*, *PORangedRush*, *POWorkerRush*, *EconomyMilitaryRush*, *EconomyRush*, *HeavyDefense*, *LightDefense*, *RangedDefense*, *WorkerDefense*, *WorkerRushPlusPlus*. Each agent played against each other as both player 1 and player 2 on the open maps of the CIG 2018 $\mu$RTS tournament.[2] We chose these agents over other agents such as *NaiveMCTS* because they execute a defined strategy which we could use as the goal for learning and plan recognition. We then generated a learning dataset of plan traces with their corresponding goals using the replay dataset. For each replay in the replay dataset, we generated two plan trace/goal pairs: one for each agent in the replay. Each pair was constructed by parsing player actions done by an agent, and using the agent itself as the goal (i.e., the goal of plan recognition will be to identify which of the agents does a plan trace come from).

$Lex_{Greedy}$ has two tunable parameters: *abstraction threshold* $\gamma$ and *pruning threshold* $\tau$. $\gamma$ controls the hierarchical structures constructed during learning, and is defined by a percentage of the training dataset. The pruning threshold removes categories from the lexicon whose conditional probability $p(c|a)$ (a is an action type, c is a category) that are below $\tau$. For our experiments, we set $\gamma = 50\%$ and $\tau = 0$ (which indicates no pruning of the CCG lexicon). Our MCTS plan recognition approach also has two tunable parameters: *number of iterations* $N$ and $\epsilon$-*greedy* threshold $\epsilon$. For our experiments, we set $N = 100, 500, 1000, 2000, 50000$, and $\epsilon = 0.4$.

[2]https://sites.google.com/site/micrortsaicompetition/rules

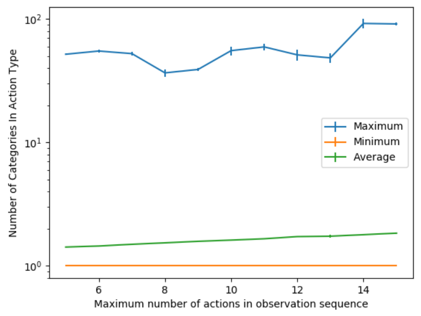


Fig. 6. Average Number of Categories For Action Types

We evaluate MCTS and BFS using four metrics: average goal prediction accuracy, average percentage of actions seen before recognition, average number of explanations generated for plan recognition, and average recognition execution time. We compute goal prediction accuracy as follows:

$$Accuracy = 100 * \frac{\textit{Number of correctly predicted goals}}{\textit{Total number of instances}}$$

The percentage of actions before recognition is computed as the minimum number of actions in an observation sequence required before a plan recognition algorithm predicts the goal, and continues to predict that goal until the end of the observation sequence. The explanations generated for plan recognition are those that explain the observed sequence of actions given to the plan recognition algorithm. As such, to compute the number of explanations, we count the number of leaves in the search tree. Finally, recognition execution time is the number of seconds that a plan recognition algorithm executed. We gather these metrics for observations sequences with a maximum of 5 to 15 actions. All metrics are averaged over all testing instances, and then averaged over 5 runs. To provide additional insight for each metric, we also compute the standard deviation over the 5 runs. For each run, we randomly shuffled all instances in the learning dataset, and split the dataset into 80% training and 20% testing.

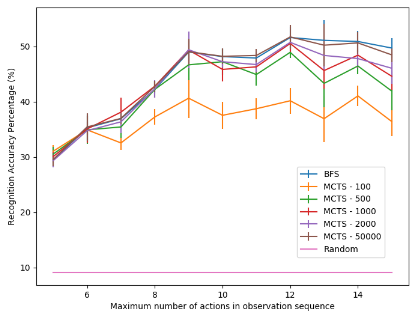
Figure 6 plots the number of categories for action types in the learned CCG lexicon against the maximum number of actions in the observation sequence. All learned CCG lexicon contain 465 action types, each corresponding to player actions done in game by the scripted agents. We see that the minimum number of categories for an action type is 1. This is because one of the requirements for $Lex_{Greedy}$ is that all action types in the learned CCG must have at least one category associated with it. We also see that the maximum number of categories for an action type is rather high, even going up to 100 categories for a single action type. However, we note that the average number of categories per action type is quite low. This implies that our learned CCGs are sparse in that many of the action types in the CCG are not associated with learned categories. However, as we will see, this does not reduce the complexity of plan recognition. Another interesting point is that, the maximum number of categories for an action type and the number of categories per action type did not deviate from the average by a significant amount over the 5 runs.

Figure 7a provides average recognition accuracy for observation sequences with a maximum of 5 to 15 actions (higher is better). Overall, we see an expected trend for all iterations of MCTS and BFS. As the number of iterations for MCTS increases, the goal prediction accuracy increases. We also see that, as we increase the number of iterations, MCTS converges to the performance of BFS. This is because predicting a goal for BFS and MCTS relies on the set of generated explanations, and the set of explanations constructed by MCTS is a subset of the explanations constructed by BFS. Thus, BFS provides an upper bound on the performance of CCG-based plan recognition. Furthermore, both MCTS and BFS outperformed a random predictor. We define a random predictor as one that chooses a goal at random to recognize an observation sequence. Since we have 11 possible goals, a random predictor would have an average accuracy of $9.09\%$.

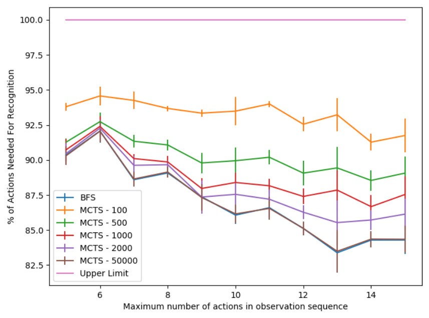
We also see that lower number of actions in an observation sequence requires less iterations of MCTS to converge to BFS. For example, for observation sequence lengths of 5-8 actions, we see that we only need 500 iterations of MCTS to converge. On the contrary, as the number of actions in the observation sequence increases, we see that complete convergence requires more iterations. In Figure 7a, we see that 50000 iterations of MCTS was not enough to completely converge to BFS. Given this trend, we can see that longer observation sequences will require more iterations of MCTS.

Figure 7b provides the average percentage of actions required to recognize the goal of an observation sequence (lower is better). Similar to that of prediction accuracy, MCTS converges to the performance of BFS as the number of iterations increases. We see that by 50000 iterations, MCTS successfully converges to BFS. Additionally, we also see that MCTS and BFS do not require all actions in the observation sequence to recognition a plan (denoted by Upper Limit in Figure 7b). However, unlike prediction accuracy, lower iterations of MCTS do not converge at shorter observation sequence lengths. This is most likely because this metric depends on the structure of the complex categories in the lexicon. Categories with more leftward arguments will delay recognition of a goal to later in the plan while those with more rightward arguments will allow for earlier recognition.

Figure 8a plots the average number of explanations generated for plan recognition against the maximum number of actions in an observation sequence (graphed using logarithmic scale). Overall, we see that MCTS successfully scales the number of generated explanations. Furthermore, we also see that as the number of iterations of MCTS increases, the number of explanations generated also increases and converges to BFS. Recall that the number of explanations generated by MCTS is a subset of the explanations of BFS. Thus, given an infinite number of iterations, MCTS should converge to BFS. An interesting observation is that MCTS still has good performance despite generating less explanations than BFS. In Figure 7a, 500 iterations of MCTS converged to the

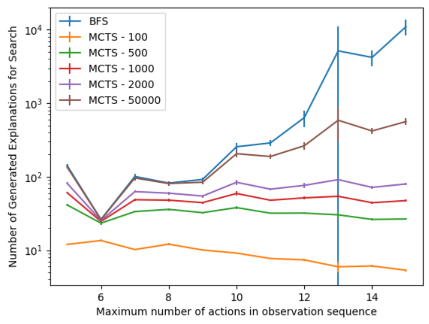


(a) Average Prediction Accuracy (higher is better)

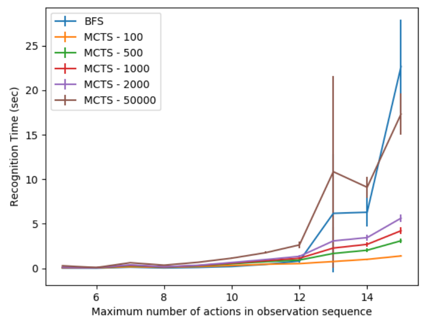


(b) Average Percentage of Actions Before Recognition of Goal (lower is better)

Fig. 7. Average of Performance Metrics for MCTS and BFS



(a) Number of Explanations after Plan Recognition



(b) Recognition Time (seconds)

Fig. 8. Average of Scaling Metrics for BFS and MCTS (lower is better)

performance of BFS given observation sequences with 5-8 actions. However, as seen in Figure 8a, MCTS only generated a fraction of the explanations of BFS. Thus, we believe some explanations are ignored by MCTS as they may not be needed to recognize the goal. We also see a steep reduction in the number of explanations for 6 action observation sequences. This is due to the complex categories in the generated lexicon for that length being more leftward than rightward, which reduces the number of generated explanations [3]. This aligns with Figure 7b, where both BFS and MCTS needed to see more actions in the plan prior to recognition.

Figure 8b provides the average amount of time (in seconds) for plan recognition to complete. Overall, we see that MCTS scales significantly better than BFS after observation sequences with 13 actions. We also observe that recognition time is lower for lower number of MCTS iterations. Figure 8b also shows that the recognition time for MCTS mostly does not vary significantly from the average.

Both BFS and MCTS with 50000 iterations has a very large standard deviation in recognition time, and BFS had a large deviation for the number of explanations for observation sequences of 13 actions. We believe this is due to outliers in the data (observation sequences which generated a large number of explanations and therefore took longer to execute). Looking closely at our data, we noticed that, for one of the runs, BFS generated millions of explanations for one observation sequence, and took approximately 448 seconds to execute. Similarly, for MCTS with 50000 iterations, one observation sequence took approximately 386 seconds. However, looking at the median number of explanation and recognition time, we see significantly lower values (359 explanations and 0.72 seconds for BFS, and 3.55 seconds for MCTS with 50000 iterations) than the mean. This confirms that there are outlier observation sequences that required more time and number of explanations than other sequences. These outliers are still important. In Real-Time Strategy games, long recognition time for even a single observation sequence could prove costly as it could be the difference between victory and defeat in a game.

For observation sequences with a maximum of 13 actions, BFS failed to recognize several of the testing instances because it ran out of memory for one of the runs. To understand why this happened, we looked the learned CCG lexicon that was constructed for that run and the observation sequences that failed. We noticed that one action in particular was repeated in the sequence: the action for constructing a worker unit. This action was repeated 6, 7, and even 11 times, which is 85% of the observation sequence. In the CCG lexicon, this action type contained 23 categories. Assuming all categories are viable and can be assigned to each action in the sequence (all leftward arguments for the categories can be discharged), the number of possible explanations would explode the search space (for 11 repetitions of the action, we can have $11^{23}$ explanations). Thus, even if the observation sequence length is not very high, BFS can exhaust all memory for search if the branching factor is high. However, MCTS did not exhaust all memory for these observation sequences. This demonstrates the effectiveness of MCTS in reducing the search space for plan recognition.

## VI. Conclusion

This paper described a Monte-Carlo Tree Search (MCTS) CCG-based plan recognition algorithm. Specifically, we employed traditional MCTS to find a set of explanations and predict the goal of a given sequence of observed actions. We demonstrated that traditional MCTS was successful in scaling for large CCG lexicons while maintaining good performance (below that of exhaustive search, but significantly better than a random prediction baseline). We saw that MCTS was able to significantly reduce the number of explanations generated during plan recognition compared to BFS. For future work, we would like to improve our MCTS algorithm by looking into different optimizations made for MCTS in the literature. We also would like to look into potentially augmenting MCTS with machine learning techniques, as they has been shown to be successful recently in other game playing domains [10]. Finally, we want to look into applying MCTS for the problem of CCG-based planning.

## References

[1] C. F. Schmidt, N. S. Sridharan, and J. L. Goodson, "The plan recognition problem: An intersection of psychology and artificial intelligence," *Artificial Intelligence*, vol. 11, no. 1-2, pp. 45–83, 1978.

[2] M. Vilain, "Getting Serious About Parsing Plans: A Grammatical Analysis of Plan Recognition." in *Proceedings of the 8th AAAI Conference on Artificial Intelligence*, 1990, pp. 190–197. [Online]. Available: http://www.aaai.org/Papers/AAAI/1990/AAAI90-029.pdf

[3] C. W. Geib, "Delaying commitment in plan recognition using combinatory categorial grammars," in *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, 2009, pp. 1702–1707.

[4] C. W. Geib and R. P. Goldman, "Recognizing plans with loops represented in a lexicalized grammar," in *Proceedings of the 25th AAAI Conference on Artificial Intelligence*, 2011, pp. 958–963. [Online]. Available: http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/viewFile/3698/3985

[5] C. W. Geib and P. Kantharaju, "Learning Combinatory Categorial Grammars for Plan Recognition," in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, 2018.

[6] P. Kantharaju, S. Ontañón, and C. W. Geib, "Extracting CCGs for plan recognition in RTS games," in *In Proceedings of the Workshop on Knowledge Extraction in Games 2019*, 2019.

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

[8] C. W. Geib, "Lexicalized Reasoning About Actions," *Advances in Cognitive Systems*, vol. Volume 4, pp. 187–206, 2016.

[9] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, K. Kavukcuoglu, M. Leach, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, p. 484, 2016.

[10] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[11] M. P. D. Schadd, M. H. M. Winands, H. J. Van Den Herik, G. M.-B. Chaslot, and J. W. H. M. Uiterwijk, "Single-player monte-carlo tree search," in *International Conference on Computers and Games*. Springer, 2008, pp. 1–12.

[12] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 4, no. 1, pp. 1–43, 2012.

[13] G. Synnaeve and P. Bessiere, "A Bayesian Model for Plan Recognition in RTS Games Applied to StarCraft," in *Proceedings of 7th AAAI Artifical Intelligence Interactive Digital Entertainment Conference*, 2011, pp. 79–84.

[14] M. Ramirez, H. Geffner, M. Ram, and H. Geffner, "Probabilistic Plan Recognition Using Off-the-Shelf Classical Planners," in *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, 2010, pp. 1121–1126.

[15] M. Ramírez and H. Geffner, "Goal recognition over POMDPs: Inferring the intention of a POMDP agent," in *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, 2011, pp. 2009–2014.

[16] D. Höller, G. Behnke, P. Bercher, and S. Biundo, "Plan and goal recognition as HTN planning," in *2018 IEEE 30th International Conference on Tools with Artificial Intelligence*. IEEE, 2018, pp. 466–473.

[17] J. McCarthy, "Circumscription—a form of non-monotonic reasoning," *Artificial Intelligence*, vol. 13, no. 1-2, pp. 27–39, 1980.

[18] H. A. Kautz and J. F. Allen, "Generalized Plan Recognition," in *Proceedings of the 5th AAAI Conference on Artificial Intelligence*, ser. AAAI'86. AAAI Press, 1986, pp. 32–37. [Online]. Available: http://dl.acm.org/citation.cfm?id=2887770.2887776

[19] D. V. Pynadath and M. P. Wellman, "Probabilistic state-dependent grammars for plan recognition," in *Proceedings of the 16th Uncertainty in Artificial Intelligence*, 2000, pp. 507–514. [Online]. Available: http://dl.acm.org/citation.cfm?id=2074005

[20] C. W. Geib and R. P. Goldman, "A probabilistic plan recognition algorithm based on plan tree grammars," *Artificial Intelligence*, vol. 173, no. 11, pp. 1101–1132, 2009.

[21] C. W. Geib, J. Maraist, and R. P. Goldman, "A new probabilistic plan recognition algorithm based on string rewriting," in *Proceedings of the 18th International Conference on Automated Planning and Scheduling*, 2008, pp. 91–98. [Online]. Available: http://www.aaai.org/Papers/ICAPS/2008/ICAPS08-012.pdf{%}5Cnpapers2://publication/uuid/BF162044-C60C-4B71-864C-F70C78A39274

[22] C. W. Geib and R. P. Goldman, "Handling Looping and Optional Actions in YAPPR," in *Proceedings of the 5th AAAI Conference on Plan, Activity, and Intent Recognition*, 2010, pp. 17–22. [Online]. Available: http://www.aaai.org/ocs/index.php/WS/AAAIW10/paper/viewPDFInterstitial/2001/2442{%}5Cnpapers2://publication/uuid/BCCB6A23-7F82-4470-AEF6-19EFE298CA11

[23] K. Gold, "Training Goal Recognition Online from Low-Level Inputs in an Action-Adventure Game," in *Proceedings of the 6th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2010, pp. 21–26.

[24] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, pp. 3371–3408, 2010.

[25] W. Min, E. Y. Ha, J. P. Rowe, B. W. Mott, and J. C. Lester, "Deep Learning-Based Goal Recognition in Open-Ended Digital Games." *Proceedings of the 10th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pp. 37–43, 2014.

[26] W. Min, B. W. Mott, J. P. Rowe, B. Liu, and J. C. Lester, "Player Goal Recognition in Open-World Digital Games with Long Short-Term Memory Networks." in *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, 2016, pp. 2590–2596.

[27] S. Ontañón, "The combinatorial multi-armed bandit problem and its application to real-time strategy games," in *Proceedings of the 9th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2013, pp. 58–64.