Recognizing Hierarchical Plans via Earley Parsing

Kristýna Pantůčková, Roman Barták

Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic
{pantuckova, bartak}@ktiml.mff.cuni.cz

Abstract

Hierarchical planning is a knowledge-based approach to planning that extends the traditional causal relations among actions by grouping actions into a hierarchical structure of tasks. This structure describes how tasks decompose into subtasks until primitive tasks - actions - are obtained. The hierarchical structure speeds up planning by providing guidelines on which actions are necessary to achieve a given task. Vice versa, given a sequence of observed actions, one can use the hierarchical structure to recognize which task the agent under observation is trying to achieve. This paper solves this second problem, plan recognition, by exploiting techniques from formal grammars. In particular, we show the application of Earley Parsing, developed for context-free grammars, to hierarchical plan recognition.

Introduction

Plan recognition is the task of recognizing the goal and the plan of an agent based on its action. In classical plan recognition, a sequence of observed actions of an agent is given as an input, and the aim is to find the complete plan and the goal of an agent based on the knowledge of the planning domain (the preconditions, effects, and costs of the actions), candidate goals and the initial state.

Ramírez and Geffner (2009) proposed a plan recognition approach based on compilation to planning. For each candidate goal, they find two plans: the optimal plan and the best plan, which uses the observed actions. Based on the assumption of the agent's rationality, the goal with the minimal difference of costs of these two plans should be the agent's true goal. Pereira, Oren, and Meneguzzi (2017) proposed an approach based on computation of landmarks. A landmark of a goal is a fact that must be true at a point in each plan for this goal. Instead of compilation to planning, they rank candidate goals based on the achieved landmarks. Other approaches to classical planning deal, for example, with on-line plan recognition (Vered et al. 2018) or epistemic plan recognition (Shvo et al. 2020).

Hierarchical planning (Erol, Hendler, and Nau 1996) deals with problems where tasks form a hierarchy. Tasks can be decomposed into subtasks until indecomposable tasks (actions), which are executable by an agent, are reached. The resulting sequence of actions is a plan of an agent. In hierarchical plan recognition, the aim is to find the root task, which decomposes into an observed prefix of a plan. Hierarchical planning problems are frequently described by hierarchical task networks (HTN). Results on complexity of plan recognition in hierarchical plan libraries were described in (Vilain 1990).

Plan recognition is relevant to many fields of artificial intelligence. For instance, plan recognition is related to behavior recognition, such as recognizing suspicious behavior in public space (Niu et al. 2004). In computer security, plan recognition can be used to predict cybernetic attacks (Li et al. 2020). Other applications include multi-agent systems (Kaminka, Pynadath, and Tambe 2002), where each agent needs to recognize the goals of other agents, or artificial intelligence in computer games (Ha et al. 2011), which needs to predict the future actions of human players.

Currently, two approaches to plan recognition appear in hierarchical task networks. The first of these approaches is based on compilation to HTN planning (Höller et al. 2018). The second approach was inspired by parsing of grammars (Barták, Maillard, and Cardoso 2020). The approach of (Höller et al. 2018) performs better than the approach of (Barták, Maillard, and Cardoso 2020) on instances with a high number of missing (unobserved) actions. However, the disadvantage of the compilation-based approach is that the recognizer requires a list of possible goals as an input. In contrast, the parsing-based approach does not require any additional knowledge apart from the description of the planning domain and the list of objects that can appear as parameters of tasks. Other hierarchical plan recognition approaches work with models weaker than HTN (e.g., (Avrahami-Zilberbrand and Kaminka 2005), (Mirsky, Gal, and Shieber 2017), or (Geib, Maraist, and Goldman 2008)).

In this paper, we present an approach to HTN plan recognition of totally ordered plans based on the Earley parser (Earley 1970). This parser has been proposed for context-free grammars and used, for example, in the global grammar constraint (Quimper and Walsh 2006). Thanks to the structural similarity between parsing trees for context-free grammars and decomposition trees for hierarchical plans, it is natural to exploit techniques from formal grammars in hierarchical planning (Barták, Maillard, and Cardoso 2020). We will show that HTN plan recognition based on the Earley parser performs significantly faster than the original parsing-based approach of (Barták, Maillard, and

Cardoso 2020) while maintaining the advantage of not being restricted to a set of concrete goal tasks.

Background

HTN plan recognition

Hierarchical planning focuses on planning problems where goals (tasks) can be hierarchically decomposed into subgoals (subtasks). Indecomposable (primitive) tasks are called actions. Actions are defined by preconditions and effects. Preconditions of an action are propositions which must be true in the state where the action is executed and effects are propositions which will be true in the state after execution of the action.

All valid decomposition of tasks into subtasks are described by methods (rules). A method describing decomposition of a task T into subtasks $T_1, ..., T_n$ corresponds to a grammar rewriting rule $T \rightarrow T_1, ...T_n[C]$, where $T_1, ..., T_n$ are either abstract or primitive tasks, C are constraints, and the order of subtasks may be arbitrary with respect to the constraints.

There are three types of constraints:

- $t_1 \prec t_2$ indicates that the last action from the decomposition of the task t_1 must be executed before the first action of the task t_2;
- *before*(T', p), where T' is a set of tasks and p is a proposition, indicates that p must hold in the state where the first action of the first task in the set T' is executed; and
- *between*(T', T'', p) indicates that p must be true in all states between the execution of the last action of the tasks in T' and the execution of the first action of the tasks in T''.

An HTN is described by a pair $w = (T, C)$, where T is a set of tasks and C is a set of constraints over tasks. A planning problem can be defined as $P = (P, T, A, M, s_0, w_0)$, where P is a set of predicates describing states, T is a set of abstract tasks, A is a set of actions, M is a set of decomposition methods, s_0 is an initial state, and w_0 is the initial task network. The task of a planner is to decompose the tasks in the initial network into actions. A solution to an HTN planning problem is an ordered sequence of actions $\pi =< a_1, ..., a_k >$ such that $w = (T, C)$ is a task network where all tasks are decomposed, which is obtained from w_0 using methods from M, π are actions, the ordering of actions in π corresponds to the ordering of nodes in w, a_1 is executable in the state s_0 and π satisfies all constraints in C.

An HTN plan recognition problem is defined as $R = (P, T, A, M, s_0, O)$, where $O =< o_1, ..., o_k >$ is an observed plan prefix of length k. The aim of plan recognition is to find a task which decomposes into a sequence of n ($k \leq n$) actions $\pi =< o_1, ..., o_k, ..., o_n >$ such that π is a valid plan applicable in s_0.

Example As an example, consider the domain transport, which describes the process of loading packages to trucks and unloading them at a different location. Figure 1 contains an example plan for the root task

$$deliver(package1, location1).$$

The initial state specifies that there is a truck *truck1* at the location *truck1_location* and there is package *package1* at the location *package1_location*. Additionally, the initial state enumerates the roads between locations. For instance, there is a road between *truck1_location* and *package1_location* and between *package1_location* and *location1*. The objective is to transport *package1* to the location *location1*.

The root task can be decomposed into four abstract subtasks:

$$\begin{aligned} &get\_to(truck1, package1\_location),\\ &load(truck1, package1\_location, package1),\\ &get\_to(truck1, location1)\\ &unload(truck1, location1, package1). \end{aligned}$$

Each of these tasks can be decomposed into a single action. For example, the first subtask

$$get\_to(truck1, package1\_location)$$

decomposes into the action

$$drive(truck1, truck1\_location, package1\_location).$$

This action has two preconditions:

$$\begin{aligned} &at(truck1, truck1\_location),\\ &road(truck1\_location, package1\_location), \end{aligned}$$

which are both satisfied in the initial state. The action has two effects:

$$\begin{aligned} &not(at(truck1, truck1\_location)),\\ &at(truck1, package1\_location). \end{aligned}$$

In HTN planning, the root task would be given as an input and the objective would be to decompose it into actions using the available decomposition methods such that all constraints of methods and all preconditions of actions are satisfied. In HTN plan recognition, the input would contain a prefix of the plan, e.g., the action sequence

$$\begin{aligned} &drive(truck1, truck1\_location, package1\_location),\\ &pickup(truck1, package1\_location, package1,\\ &capacity0, capacity1),\\ &drive(truck1, package1\_location, location1), \end{aligned} \tag{1}$$

and the objective would be to find the fourth action of the plan and the root task which can cover all actions in the prefix.

Parsing-based HTN plan recognition

Parsing-based HTN plan recognition was proposed by (Barták, Maillard, and Cardoso 2020). The algorithm iteratively extends the plan prefix by one action in each iteration and verifies whether the obtained plan is valid. This is done by composing all abstract tasks using the available methods in the bottom-up manner, until a task which covers all observations (all actions in the current plan) is found.

In each iteration, the algorithm increments the plan length and attempts to put all possible actions at the new position at the end of the plan. Their algorithm considers all actions whose preconditions could be satisfied. As a consequence, the runtime of the parsing-based algorithm grows exponentially with the increasing length of the unobserved plan suffix.

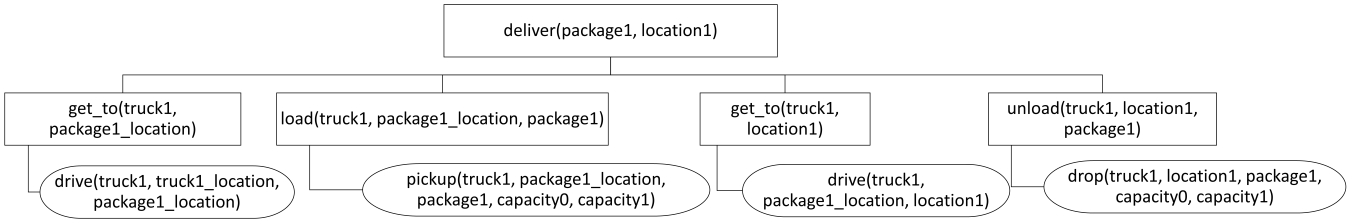

Figure 1: A hierarchical task network of the task *deliver(package1, location1)*.

Example Consider a plan recognition problem from Figure 1, whose input is the prefix (1). In the first iteration, the algorithm would create all tasks that can be composed from the actions in the prefix:

$$get\_to(truck1, package1\_location),$$

$$load(truck1, package1\_location, package1)$$

and

$$get\_to(truck1, location1).$$

As none of these tasks can cover all actions, the plan needs to be extended.

In the next iteration, the algorithm will try to put all actions at the position 4, e.g.,

$$drop(truck1, location1, package1, capacity0, capacity1),$$
$$drive(truck1, location1, package\_location),$$
$$drive(truck1, location1, truck\_location),$$
$$drive(truck1, package\_location, location1), ...,$$

and it will create all tasks that can be composed using the subset of actions in the prefix and one action in the suffix. Eventually, the root task will be composed and returned as a solution.

Earley parser and grammar constraints

Earley parser proposed by (Earley 1970) is an algorithm for parsing context-free grammars. A context-free grammar (CFG) is a formal grammar with production rules of the form $A \rightarrow \alpha$, where A is a non-terminal symbol and α is a string of terminal and non-terminal symbols.

(Quimper and Walsh 2006) showed how a parser can be used in implementation of a grammar constraint. Global grammar constraints are constraints which restrict values of variables with respect to rules of a given grammar. They proposed a constraint of the form $\mathit{CFG}(G, X_1, ..., X_n)$, where $X_1...X_n$ is a string accepted by the context-free grammar G. Given domains of the variables $X_1, ..., X_n$, the authors suggest polynomial algorithms that enforce the generalized arc consistency. A constraint is generalized arc consistent (GAC) if for each value in each domain there exists a value in every other domain such that the constraint is satisfied.

The authors propose two algorithms for enforcing GAC on $\mathit{CFG}(G, X_1, ..., X_n)$. The first algorithm is a bottom-up propagator based on the algorithm CYK and the second one is a top-down propagator based on the Earley-style propagator. Both of them have the time complexity $\mathcal{O}(|G|n^3)$. The space complexity is $\mathcal{O}(|G|n^2)$ for CYK-based algorithm and $\mathcal{O}(|G|n^3)$ for the Earley-style propagator.

We decided to use the Earley-style propagator for HTN plan recognition. Firstly, the Earley parser does not require translation to the Chomsky normal form. Secondly, in experiments of Quimper and Walsh (2006) Earley parser performed better than CYK when more than about a half of the variables in the grammar constraint were fixed. As we use the parser for plan recognition and in our testing instances the observed plan prefix is usually longer than the unobserved suffix, we assumed that the Earley propagator should perform better than CYK.

HTN plan recognition via Earley Parsing

We suggest to improve the performance of parsing-based HTN recognition of totally ordered plans by using the Earley parser as a base of the algorithm. Firstly, we use the Earley propagator to eliminate from the plan suffix those actions that are unreachable by the CFG rules. Secondly, we use only the decomposition rules which were used by the parser as production rules to compose a root task.

By omitting constraints of methods and preconditions and effects of actions, we get an abstraction of an HTN planning problem to a context-free grammar. Rules of an HTN planning domain with totally ordered subtasks correspond to production rules of a CFG. However, our approach can be used only on domains with totally ordered subtasks as it does not support task interleaving. We currently do not support domains with empty rules.

Algorithm 1 describes the general idea of plan recognition via Earley parsing. The algorithm firstly executes the Earley parser on the observed plan prefix. Then the algorithm checks whether any goal task can be composed from the ac-

Algorithm 1: HTN plan recognition via Earley parsing

Function: RecognizePlan
Input: a sequence of observed actions $a_1, ..., a_k$
Output: a $goal$ task covering all observed actions
Variables: R – decomposition rules, A – available actions, $domain$ – list of domains for all positions in a plan, $queue$ – pending states, $C[]$ – processed states

```
for i = 0, 1, ..., k do
    domain(X_i) = {a_i}
end for
initialize queue with starting states of all abstract tasks
for i = 0, 1, ..., k do
    execute iteration i of Earley parser (fill C[i])
end for
for i = k + 1, k + 2, ... do
    for all state in C[i − 1] do
        if some goal task can be computed from state then
            return goal
        end if
    end for
    domain(X_i) = A
    execute iteration i of Earley parser
end for
```

tions in the prefix. If not, the prefix is extended by one action by setting the domain of an action at the next position in the plan to all possible actions. After each extension, the next iteration of Earley parsing is executed and the algorithm attempts to compute the goal task covering an extended plan. For each possible root task found by Earley parsing, we determine whether the root task can be composed with respect to the definition of the plan recognition problem. Details of the Earley propagator can be found in (Quimper and Walsh 2006). The following text describes how we used the algorithm for HTN plan recognition.

Earley propagator is based on dynamic programming. A state represents a partially processed production rule which covers a continuous subsequence of a plan. A state is defined by a rule, a separator symbol separating processed and unprocessed tasks on the right side of the rule and the index of the iteration in which the state was created (related to the index of the first action covered by the state). Consider a state

$$s = (T_1 \rightarrow T_2...T_r \bullet T_s...T_t, j).$$

The symbol $\bullet$ separates the subtasks that were already processed from the rest. This state represents a method which decomposes the task T_1 to the subtasks $T_2, ..., T_t$. T_s is the first subtask which has not been decomposed yet and the number j is the index of the first action covered by this state.

In iteration i, the parser stores the processed states in the set $C[i]$. Pending states are stored in a queue. Initially, the queue is filled with artificial starting states of the form

$$(I \rightarrow \bullet T, 0)$$

for each task T, where I is an artificial starting (goal) task. In iteration 0, the parser processes artificial starting states. To find a plan of length n, the parser has to execute n + 1 iterations.

There are three procedures for state processing depending on the state type: completer, scanner, and predictor. A completed state is processed by the *completer*. Consider a state

$$(T_1 \rightarrow T_2...T_m \bullet, j),$$

which represents a completely decomposed task T_1. Processing this state in iteration i corresponds to processing a rule which covers actions between indexes j and i. We can use the decomposition tree represented by this state to decompose T_1 in the rules where T_1 is not decomposed yet and all tasks preceding T_1 on the right side of the rule are decomposed into a subsequence of a plan ending right before the index j. Therefore, for each state

$$(T_0 \rightarrow ... \bullet T_1..., l)$$

from the set $C[j]$, we push to the queue a new state

$$(T_0 \rightarrow ...T_1 \bullet ..., l),$$

which will cover actions between indexes l and i.

Scanner processes the states whose first unprocessed subtask is an action. Consider a state

$$(T_1 \rightarrow T_2... \bullet T_m..., j),$$

where T_m is an action. If the state is processed in iteration i, its decomposed subtasks cover actions between indexes j and i. Therefore, T_m must be an action which is available at index $i + 1$ in a plan. If $domain(X_{i+1})$ contains the action T_m, the parser adds a new state

$$(T_1 \rightarrow T_2...T_m \bullet ..., j)$$

to $C[i + 1]$ and prepares it to be processed in the next iteration.

Grounding of tasks and rules can be postponed until this step. During initialization, we fill the queue with uninstantiated rules. We set the domains as follows: if i is an index from the observed prefix, $domain(X_i)$ contains only the fully instantiated action a_i. Otherwise, $domain(X_i)$ is a set of all available actions (uninstantiated). Values of variables of instantiated actions are propagated upwards to the rules and the values from the rules are then propagated downwards to the actions in plan suffix.

Finally, the *predictor* processes uncompleted states where the next task to be processed is an abstract task. Consider a state

$$(T_1 \rightarrow T_2... \bullet T_m..., j),$$

where T_m is an abstract task, being processed in iteration i. The predictor generates all possible decompositions of T_m, which will cover actions from index i. The predictor pushes to the queue the state

$$(T_m \rightarrow \bullet T_o...T_p, i)$$

for each decomposition rule with root task T_m.

If $C[n]$ contains a state

$$(I \rightarrow T \bullet, 0), \tag{2}$$

then decomposition of the artificial task I covers actions from the artificial starting index 0 to n, which is the last index in the suffix. Therefore, T is a task which can be decomposed to a plan of length n with the given prefix and we may attempt to obtain the desired goal task by grounding T.

Computing a goal

After running an iteration of Earley parsing for a plan length n, we attempt to find a possible goal task covering all observations by extracting possible solutions from root tasks that were found by a parser. For each root state (state of the form (2)) from the set $C[n]$, we test if there is any grounding of the uninstantiated variables in any decomposition tree of task T found by the Earley parser such that all rules that were used to compose T satisfy the constraints of the HTN planning domain. For each subtask in each processed state, we keep a list of all states that can be used to complete the subtask. Starting in a root state, we traverse the resulting graph using the depth-first search to obtain a suitable grounding of all variables in the rule.

Leaf nodes of the graph correspond to actions. If the action belongs to the observed prefix, there is only one possible grounding. Otherwise, the node returns all groundings of the variables that have not been grounded by the parser.

Each internal node corresponds to one rule. For each possible grounding of its subtasks (using any completing rule for each subtask), we determine whether the rule satisfies all constrains of the corresponding HTN rule. If the complete decomposition tree of the rule of any root state can be constructed, the plan recognition problem is solved.

Example

Consider again the plan recognition problem from the previous section. The algorithm starts by enqueueing the starting states for all uninstantiated abstract tasks:

$$\begin{gathered}(I \rightarrow \bullet get\_to(?,?),0),\\ (I \rightarrow \bullet load(?,?,?),0),\\ (I \rightarrow \bullet unload(?,?,?),0),\\ (I \rightarrow \bullet deliver(?,?),0).\end{gathered} \tag{3}$$

Iteration 0: In iteration 0, the starting states will be expanded by predictor and five new states will be created

$$(get\_to(?,?) \rightarrow \bullet drive(?,?,?),0), \tag{4}$$

$$\begin{gathered}(load(?,?,?) \rightarrow \bullet pickup(?,?,?,?,?,?),0),\\ (unload(?,?,?) \rightarrow \bullet drop(?,?,?,?,?,?),0),\\ (deliver(?,?) \rightarrow \bullet get\_to(?,?),\\ load(?,?,?),\\ get\_to(?,?),\\ unload(?,?,?),0),\\ (deliver(?,?) \rightarrow \bullet load(?,?,?),\\ get\_to(?,?),\\ unload(?,?,?),0)\end{gathered} \tag{5}$$

(the last two states correspond to two methods that both decompose the task *deliver*). All states processed in iteration 0 will be added to the set $C[0]$.

The state (4) is processed by the scanner creating the state

$$(get\_to(truck1, package1\_location) \rightarrow drive(truck1, truck1\_location, package1\_location)\bullet, 0). \tag{6}$$

As $domain(X_1)$ contains the action *drive(truck1, truck1_location, package1_location)*, the new state is finished. The resulting state will then be added to the set $C[1]$ and it will be processed in iteration 1.

Iteration 1: In iteration 1, the completer will create a new state from states (5) and (6):

$$\begin{gathered}(deliver(?,?) \rightarrow get\_to(truck1, package1\_location),\\ \bullet load(truck1, package1\_location, ?),\\ get\_to(truck1, ?),\\ unload(truck1, ?, ?), 0).\end{gathered} \tag{7}$$

The predictor will expand (7) to

$$\begin{gathered}(load(truck1, package1\_location, ?) \rightarrow\\ \bullet pickup(truck1, package1\_location, ?, ?, ?), 1)\end{gathered} \tag{8}$$

The state (8) will be processed by the scanner creating a new state in $C[2]$:

$$\begin{gathered}(load(truck1, package1\_location, package1) \rightarrow\\ pickup(truck1, package1\_location, package1,\\ capacity0, capacity1)\bullet, 1)\end{gathered} \tag{9}$$

Iteration 2: The completer will produce a new state based on the states (9) and (7):

$$\begin{gathered}(deliver(package1, ?) \rightarrow\\ get\_to(truck1, package1\_location),\\ load(truck1, package1\_location, package1),\\ \bullet get\_to(truck1, ?),\\ unload(truck1, ?, package1), 0).\end{gathered} \tag{10}$$

The predictor will expand the state (10) to

$$\begin{gathered}(get\_to(truck1, ?) \rightarrow\\ \bullet drive(truck1, ?, ?), 2)\end{gathered} \tag{11}$$

The scanner will create a new state based on (11):

$$\begin{gathered}(get\_to(truck1, location1) \rightarrow\\ drive(truck1, package1\_location, location1)\bullet, 2).\end{gathered} \tag{12}$$

Iteration 3: Using the states (12) and (10), the completer creates the state

$$\begin{gathered}(deliver(package1, location1) \rightarrow\\ get\_to(truck1, package1\_location),\\ load(truck1, package1\_location, package1),\\ get\_to(truck1, location1),\\ \bullet unload(truck1, location1, package1), 0).\end{gathered} \tag{13}$$

The state (13) will be expanded (by predictor) to the state:

$$\begin{gathered}(unload(truck1, location1, package1) \rightarrow\\ \bullet drop(truck1, location1, package1, ?, ?), 3).\end{gathered}$$

From the uninstantiated actions *drive(?, ?, ?), pickup(?, ?, ?, ?, ?), drop(?, ?, ?, ?, ?)* from the set $domain(X_4)$, this state

matches for example an action instantiation *drop(truck1, location1, package1, capacity0, capacity1)* (or any other instantiation by possible values of type *capacity* – all these instantiations will be created in this step). Therefore, the scanner creates the state

$$(unload(truck1, location1, package1) \rightarrow drop(truck1, location1, package1, capacity0, capacity1)\bullet, 3). \quad (14)$$

Iteration 4: Based on the states (14) and (13), the completer will create the state

$$(deliver(package1, location1) \rightarrow get\_to(truck1, package1\_location), load(truck1, package1\_location, package1), get\_to(truck1, location1), unload(truck1, location1, package1)\bullet, 0). \quad (15)$$

Finally, the completer will use the states (15) and (3) to create the state

$$(I \rightarrow deliver(package1, location1)\bullet, 0). \quad (16)$$

The last state will be added to $C[4]$.

A goal task can be extracted from the state (16). In order to obtain the root task, we have to verify that it can be composed from the four subtasks

$$get\_to(truck1, package1\_location), load(truck1, package1\_location, package1), get\_to(truck1, location1), unload(truck1, location1, package1),$$

and that these subtasks can be composed respectively from actions

$$drive(truck1, truck1\_location, package1\_location), pickup(truck1, package1\_location, package1, capacity0, capacity1), drive(truck1, package1\_location, location1) drop(truck1, location1, package1, capacity0, capacity1).$$

If all constraints of all rules used in the resulting decomposition trees are satisfied, the task

$$deliver(package1, location1)$$

is a solution to this plan recognition problem.

Empirical evaluation

We have empirically compared the performance of three HTN plan recognizers: the original parsing-based recognizer of (Barták, Maillard, and Cardoso 2020), a modified parsing-based recognizer, which uses the Earley parser only to prune actions from the plan suffix (Pantůčková, Ondrčková, and Barták 2023), and our recognizer based entirely on Earley parsing.

domain	parsing-based	with pruning	Earley
monroe	0	165	247
satellite	98	129	141
transport	13	23	769
blocksworld	1	2	49

Table 1: Number of problems solved by the original parsing-based recognizer, parsing-based recognizer using Earley propagator to prune actions in the suffix, and our recognizer based on Earley parsing.

domain	parsing-based	with pruning	Earley
monroe	–	10	10
satellite	2	2	3
transport	2	3	3
blocksworld	1	1	5

Table 2: Maximum suffix length of plans found by the original parsing-based recognizer, parsing-based recognizer using Earley propagator to prune actions in the suffix, and our recognizer based on Earley parsing.

For experiments, we used the domains[1] and plans[2] from the International Planning Competition 2020. We used totally-ordered domains monroe (fully observable), transport, blocksworld, and satellite. The experiments were run on a computer with the Intel Core i7-11370H @ 3.30GHz processor and 16 GB of RAM. Maximum allowed runtime was set to 15 minutes for one problem. The test instances (plan prefixes) were created by deleting at least 1 and at most 1/3 of trailing actions from plans.

Table 1 compares how many problems were solved by each recognizer in each domain. In all four tested domains, the proposed recognizer based on Earley parsing solved significantly more problems than the parsing-based plan recognition approach with action pruning, which was already more efficient than the original parsing-based approach. The difference is most visible in the domain transport, where the new recognizer solved more than 30-times more instances than the other recognizers. Figures 2, 3, 4 and 5 show how the number of solved problems increased with increasing runtime. The new recognizer is significantly faster than the other recognizers. Table 2 compares the maximum lengths of suffixes of plans found by each recognizer. In the domain monroe, we found plans with unobserved suffixes of length up to 10.

Conclusion

This paper proposes a novel approach to HTN plan recognition of totally ordered plans based entirely on the Earley parser. We extended the parser to work with decomposition methods having sub-tasks with attributes and to check additional constraints that the HTN methods may use. The parser is called incrementally in a wrapper implementing the HTN plan recognition algorithm. Actions missing in the suffix

[1]https://github.com/panda-planner-dev/ipc2020-domains

[2]https://github.com/panda-planner-dev/ipc-2020-plans

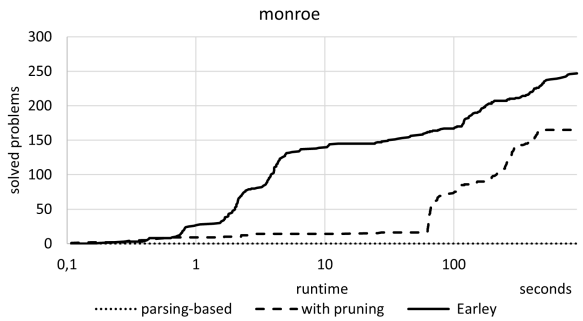

Figure 2: Number of problems in the domain monroe solved by the original parsing-based recognizer, parsing-based recognizer using Earley propagator to prune actions in the suffix, and our recognizer based on Earley parsing (horizontal axis is logarithmic).

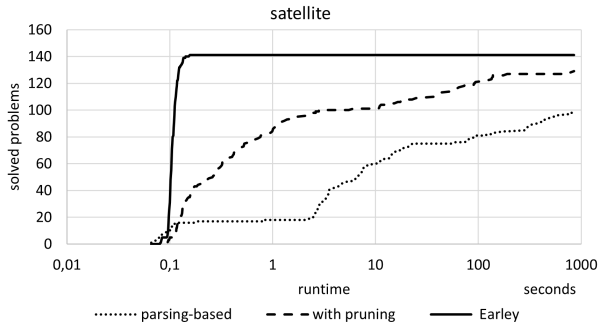

Figure 3: Number of problems in the domain satellite solved by the original parsing-based recognizer, parsing-based recognizer using Earley propagator to prune actions in the suffix, and our recognizer based on Earley parsing (horizontal axis is logarithmic).

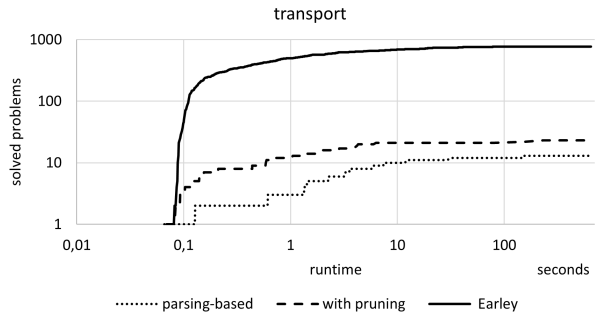

Figure 4: Number of problems in the domain transport solved by the original parsing-based recognizer, parsing-based recognizer using Earley propagator to prune actions in the suffix, and our recognizer based on Earley parsing (both axes are logarithmic).

are added greedily. Still, as the Earley parser uses a top-down approach, this is less of an issue than in the original

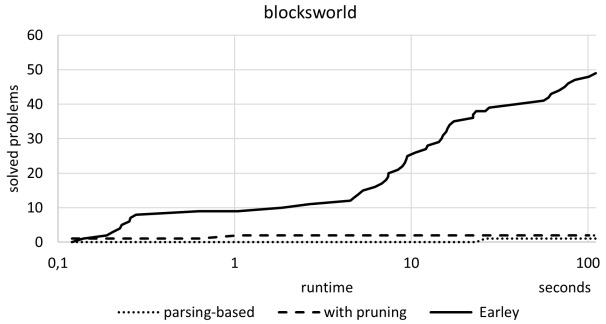

Figure 5: Number of problems in the domain blocksworld solved by the original parsing-based recognizer, parsing-based recognizer using Earley propagator to prune actions in the suffix, and our recognizer based on Earley parsing (horizontal axis is logarithmic).

parsing-based HTN plan recognizer (demonstrated by using the Earley parser to prune actions in the original recognizer). The empirical evaluation shows that the new HTN plan recognizer significantly outperforms other parsing-based recognizers, both in the number of solved problems and in runtime. Moreover, the new recognizer can also recognize plans with longer missing suffixes.

Future work may go in several directions. First, the efficiency of the parser can be further improved by applying more sophisticated techniques for grounding actions and rules or by doing grounding before parsing. Another research direction may consider incomplete or wrong information in the plan prefix. Our current approach expects that the observed prefix is a complete prefix of the actual plan. However, the observers might be unable to observe all actions in the prefix, or they may observe some actions incorrectly. Finally, the parsing techniques exploiting algorithms for context-free grammars are currently restricted to totally-ordered domains, where plans for tasks cannot interleave. An open question is whether such parsers can be extended to support the partial order of sub-tasks in decomposition rules.

Acknowledgments

Research is supported by the Charles University, project GA UK number 156121 and by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215.

References

Avrahami-Zilberbrand, D.; and Kaminka, G. A. 2005. Fast and Complete Symbolic Plan Recognition. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, 653–658.

Barták, R.; Maillard, A.; and Cardoso, R. C. 2020. Parsing-based Approaches for Verification and Recognition of Hierarchical Plans. In *The AAAI 2020 Workshop on Plan, Activity, and Intent Recognition*.

Earley, J. 1970. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2): 94–102.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1996. Complexity Results for HTN Planning. *Annals of Mathematics and AI*, 18(1): 69–93.

Geib, C. W.; Maraist, J.; and Goldman, R. P. 2008. A New Probabilistic Plan Recognition Algorithm Based on String Rewriting. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling*, 91–98.

Ha, E.; Rowe, J.; Mott, B.; and Lester, J. 2011. Goal recognition with Markov logic networks for player-adaptive games. In *Proceedings of the seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 6.

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

Kaminka, G. A.; Pynadath, D. V.; and Tambe, M. 2002. Monitoring teams by overhearing: A multi-agent plan-recognition approach. *Journal of artificial intelligence research*, 17: 83–135.

Li, T.; Liu, Y.; Liu, Y.; Xiao, Y.; and Nguyen, N. A. 2020. Attack plan recognition using hidden Markov and probabilistic inference. *Computers & Security*, 97: 101974.

Mirsky, R.; Gal, Y.; and Shieber, S. M. 2017. CRADLE: an online plan recognition algorithm for exploratory domains. *ACM Transactions on Intelligent Systems and Technology*, 8(3): 1–22.

Niu, W.; Long, J.; Han, D.; and Wang, Y.-F. 2004. Human activity detection and recognition for video surveillance. In *Proceedings of the 2004 IEEE International Conference on Multimedia and Expo (ICME)*, volume 1, 719–722.

Pantůčková, K.; Ondrčková, S.; and Barták, R. 2023. Parsing-Based Recognition of Hierarchical Plans Using the Grammar Constraint. In *The International FLAIRS Conference Proceedings*, volume 36.

Pereira, R. F.; Oren, N.; and Meneguzzi, F. 2017. Landmark-based heuristics for goal recognition. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, 3622–3628.

Quimper, C.-G.; and Walsh, T. 2006. Global grammar constraints. In *Principles and Practice of Constraint Programming-CP 2006: 12th International Conference, CP 2006, Nantes, France, September 25-29, 2006. Proceedings 12*, 751–755. Springer.

Ramírez, M.; and Geffner, H. 2009. Plan recognition as planning. In *Proceedings of the twenty-first International Joint Conference on Artifical Intelligence*, 1778–1783.

Shvo, M.; Klassen, T. Q.; Sohrabi, S.; and McIlraith, S. A. 2020. Epistemic Plan Recognition. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, 1251–1259.

Vered, M.; Pereira, R. F.; Kaminka, G.; and Meneguzzi, F. R. 2018. Towards online goal recognition combining goal mirroring and landmarks. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems*, 2112–2114.

Vilain, M. 1990. Getting Serious About Parsing Plans: A Grammatical Analysis of Plan Recognition. In *Proceedings of the Eighth AAAI Conference on Artificial Intelligence*, 190–197.