ECAI 2023
K. Gal et al. (Eds.)

doi:10.3233/FAIA230469

Using Earley Parser for Recognizing Totally Ordered Hierarchical Plans

Kristýna Pantůčková * **and Roman Barták**

Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic

Abstract. Earley Parser is a top-down parser proposed for context-free grammars and used, for example, in the grammar constraint. Parsing trees of context-free grammars are very close to task decomposition trees used in hierarchical planning, specifically when the actions are totally ordered. This paper suggests using the Earley Parser to recognize totally ordered hierarchical plans. Given a sequence of actions – a prefix of the plan – and a task decomposition model, the plan recognition problem asks which task decomposes to a plan containing the given plan prefix. We will show that the Earley parser significantly increases the speed of plan recognition compared to the existing bottom-up parsing-based plan recognizer. The Earley parser's performance is also on a par with the planning-based plan recognizer despite not using any planning heuristics.

1 Introduction

Plan recognition is the task of recognizing the goal and the plan of an agent based on observed agent's actions. In classical plan recognition, a sequence of observed actions of an agent (a plan prefix) 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 actions), candidate goals, and the initial state.

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 [12]. In computer security, plan recognition can be used to predict cybernetic attacks [10]. Other applications include multi-agent systems [9], where each agent needs to recognize the goals of other agents, or artificial intelligence in computer games [7], which needs to predict the future actions of human players.

A popular approach to plan recognition is based on compilation to planning [16]. Two plans are found for each candidate goal: 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. Another approach is based on landmarks [14]. 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 [19], epistemic plan recognition [18], or using reinforcement learning [1].

Hierarchical planning [4] 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. Hierarchical planning problems are frequently described by hierarchical task networks (HTN). In hierarchical plan recognition, the aim is to find a root task, which decomposes into a plan starting with the observed plan prefix. Results on complexity of plan recognition in hierarchical plan libraries were described in [20].

Currently, two approaches to plan recognition appear in HTNs. The first of these approaches is based on compilation to HTN planning [8]. The second approach was inspired by parsing of grammars [2]. The approach of [8] performs better than the approach of [2] on instances with a high number of missing (unobserved) actions, while [2] is faster on instances with few missing actions. There are also hierarchical plan recognition approaches working with models weaker than HTN, some of which are also based on parsing of grammars (e.g. [5], [6], [11]).

This paper focuses on plan recognition in totally ordered domains, i.e. domains, where subtasks of each decomposition method are linearly ordered. Although total order imposes a certain restriction on HTN planning, totally ordered domains are very common. For instance, 24 out of 33 domains used in the international planning competition (IPC) 2020 were totally ordered.

We present an approach to HTN plan recognition of totally ordered plans based on the Earley parser [3]. This parser has been proposed for context-free grammars and used, for example, in the global grammar constraint [15]. 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 [2]. We will show that HTN plan recognition based on the Earley parser performs significantly faster on totally ordered domains than the original parsing-based approach of [2] and it also outperforms the compilation-based approach [8] on some domains.

2 Background

2.1 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 become true in the state after execution of the action.

* Corresponding Author. Email: pantuckova@ktiml.mff.cuni.cz.

All valid decompositions of tasks into subtasks are described by decomposition methods (rewriting rules). A method describing decomposition of an abstract 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 in the rule can be arbitrary (the ordering might be imposed by the constraints). In totally ordered domains, the tasks in every decomposition method are totally (linearly) ordered.

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''.

A hierarchical task network (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 (primitive tasks), 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 $\pi =< a_1, ..., a_k >$ corresponding to primitive tasks in a task network $w = (T_w, C_w)$, where T_w does not contain any abstract task, w is obtained from w_0 using methods from M, and π is a valid plan applicable to state s_0 (action preconditions are satisfied) satisfying all constraints in C_w.

An HTN plan recognition problem is defined as $R = (P, T, A, M, s_0, O)$, where $O =< a_1, ..., a_k >$ is an observed plan prefix of length k. The aim of plan recognition is to find an abstract task which decomposes into a sequence of n ($k \leq n$) actions $\pi =< a_1, ..., a_k, ..., a_n >$ such that π is a valid plan applicable to state s_0.

2.1.1 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.

2.2 Parsing-based HTN plan recognition

Parsing-based HTN plan recognition was proposed by [2]. 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 (in the unobserved plan suffix) 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.

2.2.1 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.

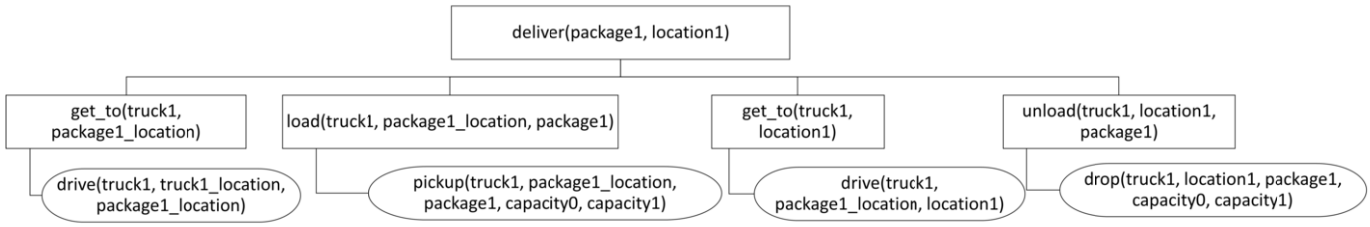

Figure 1. A hierarchical task network of the task *deliver(package1, location1)*.

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

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

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.

2.3 Earley parser and grammar constraints

Earley parser [3] is an algorithm for parsing context-free grammars. A context-free grammar (CFG) is a formal grammar with rewriting 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.

The parser can be used to implement a grammar constraint [15]. 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 $CFG(G, X_1, ..., X_n)$, where $X_1...X_n$ is a string accepted by the context-free grammar represented by the set of grammar rewriting rules 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 $CFG(G, X_1, ..., X_n)$. The first algorithm is a bottom-up propagator based on the algorithm CYK [17] and the second one is a top-down propagator based on the Earley-style propagator [3]. 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 have decided to use the Earley-style propagator for HTN plan recognition. Firstly, the Earley parser does not require translation to the Chomsky normal form with at most two subtasks in the decomposition. Secondly, in experiments of [15] 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. Another motivation for the Earley parser is that top-down parsing includes some form of planning (to find not-yet observed actions); therefore, it could be better suited for plan recognition. Moreover, due to similarity to HTN planning, the Earley parser can exploit heuristics developed for HTN planning, which brings opportunities for future improvement of its efficiency.

3 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. We present two novel approaches to HTN plan recognition based on the Earley parser. Our first approach works as the original parsing-based approach, but plan suffixes are limited to those that are accepted by the corresponding CFG, in contrast to the approach of [2], which tries all suffixes. Our second approach uses the Earley parser to identify the decomposition trees which are valid with respect to the CFG. We then use only these decomposition trees to attempt to compose a root task, in contrast to the approach of [2], which tries all decomposition trees that can be created by available methods.

By omitting preconditions and effects of actions and constraints of methods 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 rewriting 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 actions in the prefix. If not, the prefix is extended by one action by setting the domain of an action variable 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 [15]. 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 rewriting 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

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

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
```

The symbol • separates the subtasks that were already processed (left from •) from the rest (right from •). 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.

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

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

for each task T, where I is a dummy starting (goal) task. At iteration 0, the parser processes the 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 at 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 at 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 at 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 method with root task T_m.

If $C[n]$ contains a state

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

then decomposition of the dummy task I covers actions from the artificial starting index 0 to n, which is the last index in the suffix. Therefore, T is a root 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.

3.1 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 check whether the rule satisfies all constraints of the corresponding HTN rule; i.e., we check whether all ordering, before, and between constraints of the rule are satisfied for the selected subtasks. Detailed algorithm of constraint checking can be found in [2]. If the complete decomposition tree of the rule of any root state can be constructed, the plan recognition problem is solved.

3.2 Example

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

$$(I \rightarrow \bullet get\_to(?,?),0),$$
$$(I \rightarrow \bullet load(?,?,?),0),$$
$$(I \rightarrow \bullet unload(?,?,?),0),$$
$$(I \rightarrow \bullet deliver(?,?),0). \quad (3)$$

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

$$(get\_to(?,?) \rightarrow \bullet drive(?,?,?),0), \quad (4)$$

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

(the last two states correspond to two methods that both decompose the task *deliver*). All states processed at iteration 0 will be added to the set $C[0]$.
The state (4) is processed by the scanner creating the state

$$\begin{aligned}(get\_to(truck1, package1\_location) \rightarrow\\ drive(truck1, truck1\_location, package1\_location)\bullet,0).\end{aligned} \quad (6)$$

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 at iteration 1.

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

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

The predictor will expand (7) to

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

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

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

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

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

The predictor will expand the state (10) to

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

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

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

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

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

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

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

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

$$\begin{aligned}(unload(truck1, location1, package1) \rightarrow\\ drop(truck1, location1, package1,\\ capacity0, capacity1)\bullet, 3).\end{aligned} \quad (14)$$

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

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

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 $C[4]$.

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

domain	parsing-based	with pruning	Earley	Panda
monroe	0	165	247	342
satellite	98	129	141	141
transport	13	23	266	43
blocksworld	1	2	49	68

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, our recognizer based on Earley parsing and the recognizer from the system Panda.

domain	parsing-based	with pruning	Earley	Panda
monroe	–	10	10	21
satellite	2	2	3	3
transport	2	3	3	4
blocksworld	1	1	5	5

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 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:

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

i.e., we check whether all constraints of the method that decomposes the root task to the subtasks are satisfied. Then, we recursively check the same for all methods used in decomposition trees of the subtasks. If all constraints in the decomposition tree are satisfied, the task

$$deliver(package1, location1)$$

is a solution to this plan recognition problem.

4 Empirical evaluation

We have empirically compared the performance of four HTN plan recognizers: the original parsing-based recognizer [2], a modified parsing-based recognizer, which uses the Earley parser only to prune actions from the plan suffix [13], our recognizer based entirely on Earley parsing, and the compilation-based recognizer from the planning system Panda [8] (with the default settings).

For experiments, we used the domains[1] and plans[2] from IPC 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. For each plan we generated all prefixes that could be created by deleting at least 1 and at most 1/3 of trailing actions, i.e., we created approximately n/3 prefixes from a single plan of length n. From the domain blocksworld, we used the first 200 problems as none of the recognizers was able to solve more than the first few problems. From the domain transport, we used only the first 266 problems as the other recognizers except for the recognizer based on Earley parser were able to solve only the first few problems in the given time limit.

The results show that the recognizer based on the Earley parser was more efficient than the other parsing-based approaches. 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 in the given time limit 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 10-times more instances than the other parsing-based 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 parsing-based recognizers. Table 2 compares the maximum lengths of suffixes of plans found by each recognizer. In the domain monroe, our recognizer found plans with unobserved suffixes of length up to 10.

Table 1 shows that our recognizer was outperformed by Panda on the domains monroe and blocksworld. In the domain transport (see Figure 4), our recognizer solved considerably more problems than Panda. Both recognizers solved the same number of problems in the domain satellite; however, Figure 3 shows that our recognizer solved the problems significantly faster than Panda. Moreover, Figures 2 and 5 show that even though Panda eventually solved more problems in the given time limit, our recognizer solved smaller problems much faster. The recognizer from the system Panda works with grounded problems, which requires a certain overhead. On the other hand, Panda is less affected by the length of unobserved suffix (see Table 2).

Generally, the compilation-based approach (Panda) is slower than Earley parser on simple instances (shorter unobserved suffixes) as Panda requires an expensive overhead due to grounding. Therefore, it is less efficient in the domain satellite, which consists of the simplest problems. The domain monroe defines a rich hierarchy of object types, and it contains many different methods with arguments of different types. We assume that the compilation-based approach benefits from grounding there. Goal tasks in the domain transport correspond to sequences of tasks of type "deliver package x to location y". The description of the domain is quite simple, it contains few methods and types. We suspect that in this case grounding is not very efficient and the compilation-based approach deals with very large problem descriptions. The domain blocksworld defines only one type of object (a block) and the goals describe which blocks should be put on other blocks. It seems that these problems are very hard for all recognizers as there are too many possibilities. The compilation-based approach performs better, but the difference in absolute numbers of solved problems is not very significant.

Panda uses an HTN planner, which uses grounding and heuristics to speed up planning, while our recognizer works with uninstantiated tasks and rules, which are grounded on demand when a solution is being extracted, and does not use any heuristics. Therefore, performance of our recognizer might be improved by more sophisticated

[1] https://github.com/panda-planner-dev/ipc2020-domains
[2] https://github.com/panda-planner-dev/ipc-2020-plans

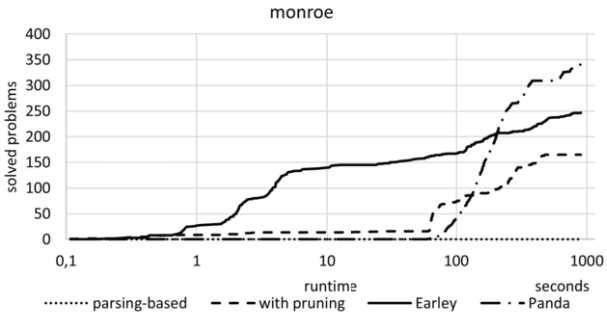

Figure 2. Number of problems in the domain monroe solved by the original parsing-based recognizer, parsing-based recognizer using Earley propagator to prune actions in the suffix, our recognizer based on Earley parsing and the recognizer from the system Panda (horizontal axis is logarithmic).

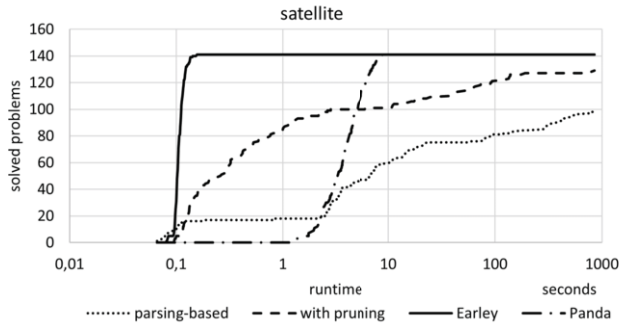

Figure 3. Number of problems in the domain satellite solved by the original parsing-based recognizer, parsing-based recognizer using Earley propagator to prune actions in the suffix, our recognizer based on Earley parsing and the recognizer from the system Panda (horizontal axis is logarithmic).

grounding of actions and by modifying the parsing algorithm using heuristic selection of states.

5 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 subtasks 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 are added greedily. Still, as the Earley parser uses a top-down approach, this is less of an issue than in the original 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. On some domains, the new recognizer also outperforms the compilation-based approach [8].

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 decomposition methods or by doing grounding before parsing, or by introducing heuristics into the parser. 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 subtasks in decomposition rules.

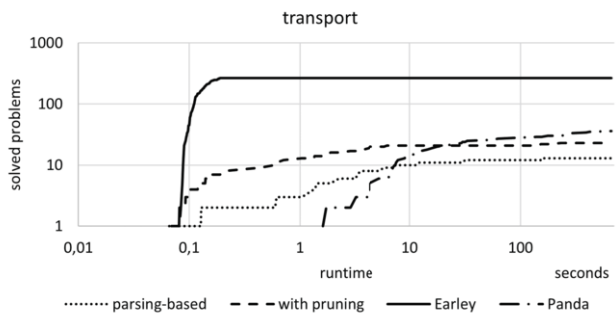

Figure 4. Number of problems in the domain transport solved by the original parsing-based recognizer, parsing-based recognizer using Earley propagator to prune actions in the suffix, our recognizer based on Earley parsing and the recognizer from the system Panda (both axes are logarithmic).

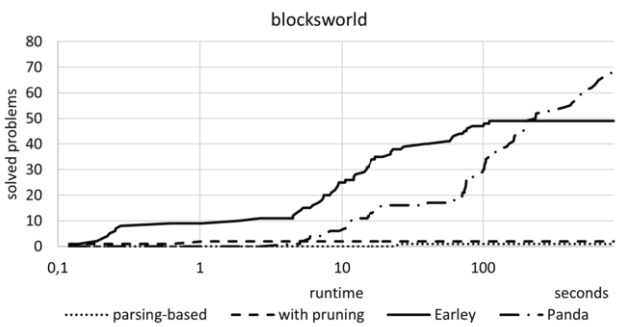

Figure 5. Number of problems in the domain blocksworld solved by the original parsing-based recognizer, parsing-based recognizer using Earley propagator to prune actions in the suffix, our recognizer based on Earley parsing and the recognizer from the system Panda (horizontal axis is logarithmic).

Acknowledgements

Research is supported by the Charles University, project GA UK number 156121, by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215 and by SVV project number 260 698.

References

[1] Leonardo Amado, Reuth Mirsky, and Felipe Meneguzzi, 'Goal recognition as reinforcement learning', *Proceedings of the AAAI Conference on Artificial Intelligence*, **36**(9), 9644–9651, (Jun. 2022).

[2] Roman Barták, Adrien Maillard, and Rafael C Cardoso, 'Parsing-based approaches for verification and recognition of hierarchical plans', in *The AAAI 2020 Workshop on Plan, Activity, and Intent Recognition*, (2020).

[3] Jay Earley, 'An efficient context-free parsing algorithm', *Communications of the ACM*, **13**(2), 94–102, (1970).

[4] Kutluhan Erol, James A. Hendler, and Dana S. Nau, 'Complexity Results for HTN Planning', *Annals of Mathematics and AI*, **18**(1), 69–93, (1996).

[5] Christopher Geib and Robert Goldman, 'A probabilistic plan recognition algorithm based on plan tree grammars', *Artificial Intelligence*, **173**(11), 1101–1132, (2009).

[6] Christopher Geib, John Maraist, and Robert Goldman, 'A new probabilistic plan recognition algorithm based on string rewriting.', in *Proceedings of the 18th International Conference on Automated Planning and Scheduling*, pp. 91–98, (2008).

[7] Eun Ha, Jonathan Rowe, Bradford Mott, and James Lester, '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, (2011).

[8] Daniel Höller, Gregor Behnke, Pascal Bercher, and Susanne Biundo, 'Plan and goal recognition as HTN planning', in *2018 IEEE 30th International Conference on Tools with Artificial Intelligence*, pp. 466–473, (2018).

[9] Gal A Kaminka, David V Pynadath, and Milind Tambe, 'Monitoring teams by overhearing: A multi-agent plan-recognition approach', *Journal of artificial intelligence research*, **17**, 83–135, (2002).

[10] Tun Li, Yutian Liu, Yanbing Liu, Yunpeng Xiao, and Nang An Nguyen, 'Attack plan recognition using hidden Markov and probabilistic inference', *Computers & Security*, **97**, 101974, (2020).

[11] Reuth Mirsky, Ya'akov Gal, and Stuart M Shieber, 'CRADLE: an online plan recognition algorithm for exploratory domains', *ACM Transactions on Intelligent Systems and Technology*, **8**(3), 1–22, (2017).

[12] Wei Niu, Jiao Long, Dan Han, and Yuan-Fang Wang, 'Human activity detection and recognition for video surveillance', in *Proceedings of the 2004 IEEE International Conference on Multimedia and Expo (ICME)*, volume 1, pp. 719–722, (2004).

[13] Kristýna Pantůčková, Simona Ondrčková, and Roman Barták, 'Parsing-based recognition of hierarchical plans using the grammar constraint', *The International FLAIRS Conference Proceedings*, **36**(1), (May 2023).

[14] Ramon Fraga Pereira, Nir Oren, and Felipe Meneguzzi, 'Landmark-based heuristics for goal recognition', in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pp. 3622–3628, (2017).

[15] Claude-Guy Quimper and Toby Walsh, '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*, pp. 751–755. Springer, (2006).

[16] Miquel Ramírez and Hector Geffner, 'Plan recognition as planning', in *Proceedings of the twenty-first International Joint Conference on Artifical Intelligence*, pp. 1778–1783, (2009).

[17] Itiroo Sakai, 'Syntax in universal translation', in *Proceedings of the International Conference on Machine Translation and Applied Language Analysis*, pp. 593 – 608, (1961).

[18] Maayan Shvo, Toryn Q Klassen, Shirin Sohrabi, and Sheila A McIlraith, 'Epistemic plan recognition', in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 1251–1259, (2020).

[19] Mor Vered, Ramon Fraga Pereira, Gal Kaminka, and Felipe Rech Meneguzzi, 'Towards online goal recognition combining goal mirroring and landmarks', in *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems*, pp. 2112–2114, (2018).

[20] Marc Vilain, 'Getting Serious About Parsing Plans: A Grammatical Analysis of Plan Recognition', in *Proceedings of the Eighth AAAI Conference on Artificial Intelligence*, pp. 190–197, (1990).