Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 176 (2020) 878–887

Procedia Computer Science

www.elsevier.com/locate/procedia

24th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems

Behaviour recognition of planning agents using Behaviour Trees

Stanislav Sitanskiy, Laura Sebastia, Eva Onaindia[a,*]

[a] *Valencian Research Institute for Artificial Intelligence, Universitat Politècnica de València, Valencia - Spain*

Abstract

Research on AI is more and more focusing towards *explainable* technology that accounts for the outcomes of programs and products. One important aspect in this direction is the ability to recognize the behaviour patterns of our application in order to make sensible and informed decisions. In this work, we aim to uncover the behaviour of a planning agent by means of Behaviour Trees (BTs), a flexible and controllable mathematical model of plan execution used to describe flows between tasks in a modular fashion. We analyze the behaviour of a planning agent when solving a simple logistics problem by comparing the agent's plan with the plans resulting from various BTs, each representing a different behaviour. We propose different distance metrics as a similarity measurement between plans and we evaluate their accuracy at identifying similar behaviours.

Keywords: Type your keywords here, separated by semicolons ;

1. Introduction

Understanding the behaviour of a system is crucial to be able to make predictions or explaining selected choices. The behaviour of a system goes beyond the model that governs the physics of the domain; i.e., the *model* that determines the legal actions that can be performed and under which conditions whereas the *behaviour* determines the reasons for choosing one action among many options or the order in which actions will be performed. Generally speaking, the model tells us *what* can be done, the behaviour tell us *how* this is done.

In this work, we aim to uncover the behaviour of a planning agent. The agent is given various planning problems, each one consisting in achieving a set of goals, and returns a solution plan for each of them. Our objective is to find out the strategy that the agent uses when calculating the plans that solve the planning problems. In order to understand the behaviour of the planning agent, we put the focus on the order of

* Corresponding author. Tel.: +34-963877000.
E-mail address: stasiig@inf.upv.es; {lsebastia, onaindia}@dsic.upv.es

1877-0509
10.1016/j.procs.2020.09.083

Node type	Succeeds	Fails	Running
Fallback	If one child succeeds	If all children fail	If one child returns *Running*
Sequence	If all children succeed	If one children fails	If one child returns *Running*
Parallel	If $\geq M$ children succeed	If $> N - M$ children fail	Otherwise
Decorator	Custom	Custom	Custom
Action	Upon completion	If impossible to complete	During completion
Condition	If true	If false	Never

Table 1: BT node types

solving the problem goals. Particularly, we are interested in studying if the agent follows some behavioural pattern to address the problem; e.g., sequentially solving the problem goals, interleaving actions oriented at achieving different goals or any combination of these strategies.

Discovering the behaviour of a planning agent is closely related to the concept of activity and plan recognition [15] but it stresses a different dimension of the acting agent. While plan recognition puts the focus on identifying the goal or intention of the agent, and the corresponding path to achieve it, given some observations of its behaviour [11, 5, 13], a behavioural analysis aims at finding the patterns that govern the know-how of the agent. In this sense, it can be argued that analyzing the behaviour of a planning agent resembles the purpose pursued within pattern discovery in data mining [6]. The objective is certainly similar but a key distinction lies in that a planning agent builds up a plan by using an action model and so the behaviour of the agent does not solely respond to some goal-driven strategy but also to some (unknown) model that governs the physics of the domain. In a nutshell, pattern discovery is suited for data-driven approaches while our proposal aims for finding patterns in data that come from model-based approaches, specifically from action models.

There exist various methods that can be exploited for discovering the behaviour of an agent. Finite State Machines (FSMs) are widely used, for instance, for decision making in game AI. The difficulty in reusing transitions in FSMs is a major limitation in planning since addressing the same goal for a number of objects of the same type requires using the same type of actions. Another solution is using Behaviour Trees (BTs), which focus on making individual states modular and so they can be easily reused in different parts of the behaviour [10, 3, 2].

In this paper we exploit BTs to discover the behaviour of a planning agent when solving problems of a *Logistics* domain, a classical planning domain introduced in the first International Planning Competition[1]. We will define three different types of logic (behaviours) for solving the problems of this domain and we propose a recognition method as a mapping from the plans that solve these problems to the behaviour that best explains them.

2. Behaviour trees

A Behaviour Tree (BT) is a directed rooted tree where the internal nodes are called *control flow nodes* and leaf nodes are called *execution nodes* [3]. The **execution** of a BT starts from its root node, generating signals (*ticks*) that are sent to the nodes with a given frequency. When a node receives a tick, this is sent to its children and so on until an execution node is reached. A tick received in a leaf node enables the execution of such a node. The leaf node immediately returns *Running* to its parent, if its execution is under way; *Success* if it has achieved its goal; or *Failure* otherwise. Subsequently, the signal emitted by the execution nodes is successively backed up to the upper nodes.

In the classical formulation, there exist four categories of control flow nodes (Fallback, Sequence, Parallel, and Decorator) and two categories of execution nodes (Action and Condition). Table 1 summarizes the conditions for each type of node to emit a tick of Success, Failure or Running.

[1] http://ipc98.icaps-conference.org/

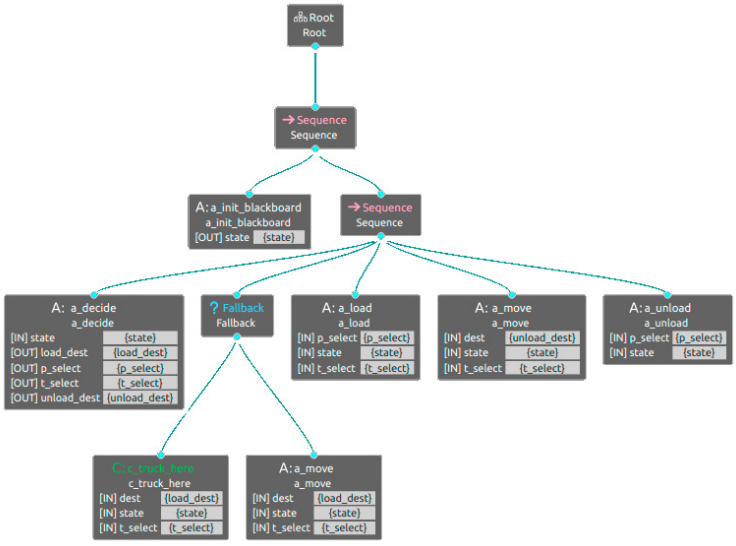


Fig. 1: Example of BT in our domain

While a BT executes the logic, a separate asset called *Blackboard* is used to store information (called Blackboard Keys) that the BT needs to know about in order to make informed decisions. Each action and condition node may receive some Blackboard Keys through input ports (which can be seen as the input parameters of a function) and they may modify the value of these Blackboard Keys through output ports (such as the returned values of a function).

The BT in Figure 1 pictures a simple behaviour for our *Logistics* domain. This is a transportation domain involving aircraft and trucks, with trucks constrained to movement within cities and aircraft constrained to movement between (inter-city) airports. The three primitive actions defined in this planning domain are 'loading a package into a truck at a given location', 'driving a truck between two locations' and 'unloading a package from a truck at a given location'[2]. . For the sake of clarity, and without any loss of generality, we use a simplified version of the *Logistics* domain, where all the packages must be transported within the same city and so airplanes are not needed.

In Figure 1, the child of the root is a Sequence node, which indicates that the action `a_init_blackboard` and the subtree below the second Sequence node will be executed from left to right. When the `a_init_blackboard` action is ticked, the Blackboard is initialized with the initial state and the goal of the problem and the node returns *Success*. Then, the next node that receives the tick is the `a_decide` action. This node reads the variable state from the blackboard and fills the output *blackboard keys* with the package p_select to be transported, its current location load_dest, the truck t_select to be used and the destination unload_dest. Then, the Fallback node is reached, which transfers the tick to the condition node `c_truck_here`. This condition node checks whether the truck t_select is at location load_dest. If so, it returns *Success*, the Fallback node also returns *Success* and the BT execution continues with the `a_load` action; otherwise, the `a_move` action is ticked, and it 'drives' t_select to load_dest. This really means that the state is modified to reflect the current location of the truck. In fact, the three remaining actions behave similarly: when they are ticked, they modify the state to indicate that package p_select is in truck t_select (action `a_load`), that truck t_select is driven to location load_dest and, finally, that package p_select is unloaded at location load_dest.

[2] The language used for the definition of the planning domains and problems in the IPC is PDDL (Planning Domain Description Language)[8, 4]

Plan A	Plan B	Plan C	Plan D
A1: LOAD P1 T1 L1	A1: LOAD P1 T1 L1	A7: LOAD P1 T2 L1	A2: LOAD P2 T1 L1
A2: LOAD P2 T1 L1	A3: DRIVE T1 L1 L2	A8: LOAD P2 T2 L1	A1: LOAD P1 T1 L1
A3: DRIVE T1 L1 L2	A4: UNLOAD P1 T1 L2	A9: DRIVE T2 L1 L2	A3: DRIVE T1 L1 L2
A4: UNLOAD P1 T1 L2	A6: DRIVE T1 L2 L1	A10: UNLOAD P1 T2 L2	A5: UNLOAD P2 T1 L2
A5: UNLOAD P2 T1 L2	A2: LOAD P2 T1 L1	A11: UNLOAD P2 T2 L2	A4: UNLOAD P1 T1 L2
	A3: DRIVE T1 L1 L2		
	A5: UNLOAD P2 T1 L2		

Table 2: Four example plans

3. Behaviour trees for the *Logistics* domain

In this section, we first describe three different behaviours to solve problems of the *Logistics* domain and subsequently we show the construction of the BT for each behaviour. The three defined behaviours are:

- **Behaviour 1:** All the packages are transported one by one. That is, one truck is driven to the location of one package; the package is loaded into the truck; the truck is driven to the destination, where the package is unloaded; then, another package is processed in the same way. The corresponding regular expression of this behaviour is $(DRIVE^*\ LOAD\ DRIVE^+\ UNLOAD)^+$.
- **Behaviour 2:** All the packages are loaded into a truck before starting unloading them. That is, one (or several trucks) are driven to different packages locations and each package is loaded into the corresponding truck; once all of the packages have been loaded, the trucks go to to their destinations and the packages are unloaded. The corresponding regular expression of this behaviour is $(DRIVE^*\ LOAD)^+\ (DRIVE^*\ UNLOAD)^+$.
- **Behaviour 3:** This behaviour does not impose any on the order in which the actions are executed. The corresponding regular expression of this behaviour is $(DRIVE^*\ LOAD^+\ DRIVE^*\ UNLOAD^+)^+$.

Table 2 shows four plans that solve the same *Logistics* problem: transporting packages `P1` and `P2` from location `L1` to location `L2` with two available trucks, `T1` and `T2`. Plans A, C and D follow Behaviour 2, whereas plan B follows Behaviour 1.

3.1. Construction of BTs

We distinguish the following types of nodes for the construction of the BTs:

- Condition nodes: denoted by `c_*`, these nodes evaluate whether a condition is fulfilled in the current state.
- Action nodes: they are in turn classified as:
  - Primitive action nodes: denoted by `a_{PDDL_action}`, these nodes simulate the execution of a PDDL action and they are the only ones that modify the current state.
  - Decision nodes: denoted by `a_decide_*`, these nodes simulate the decisions taken by a planner with respect to which truck to move or which package to load, using, for example, the current state as input.
- Decorator nodes: they are used to control the execution flow of the tree.

An additional action, named `a_init_bb`, is responsible for reading the initial and goal states and building the current state for the BT. Figures 2 and 3 show the BTs corresponding to the Behaviour 1 and Behaviour 2 described above. Following we describe the meaning of the nodes.

There are two **condition nodes**:

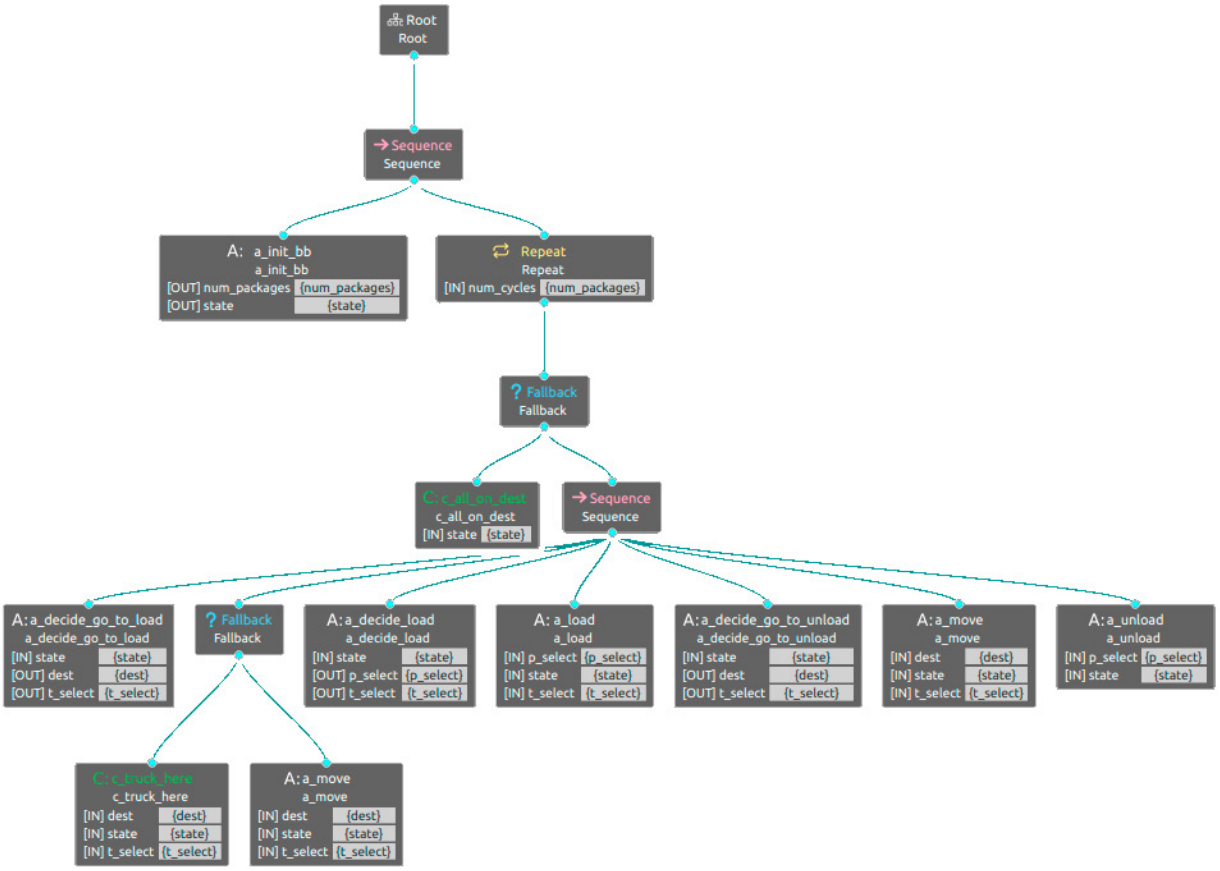


Fig. 2: Behaviour 1: transport packages one by one

1. `c_all_at_dest`: this condition node returns *Success* if all the packages are already at their corresponding destination or *Failure* otherwise.
2. `c_truck_here`: this condition node receives a truck identifier and a location; it returns *Success* if the truck is at the given location in the current state or *Failure* otherwise.

Action nodes represent nodes wherein some action is taken:

1. The **primitive action nodes** represent the execution of a planning action. Specifically, a primitive action node is defined for each action of the *Logistics* domain, namely `a_drive_truck`, `a_load_truck` and `a_unload_truck`. These nodes receive the current state as well as the parameters needed to execute the action that cannot be inferred from the current state. For example, the `a_drive_truck` action receives the current state, the truck to drive and the destination to which the truck heads for. Unlike the corresponding PDDL action, the node does not receive the current location of the truck because this can be retrieved from the current state.
   A primitive action node returns *Success* and modifies the current state according to the semantics of the PDDL action. In our example, the execution of the `a_drive_truck` action will modify the position of the truck to the destination received as an input parameter.
2. The **decision nodes** are aimed at selecting the truck or package that will be used in the subsequent primitive action. The decision nodes defined in our BTs are:

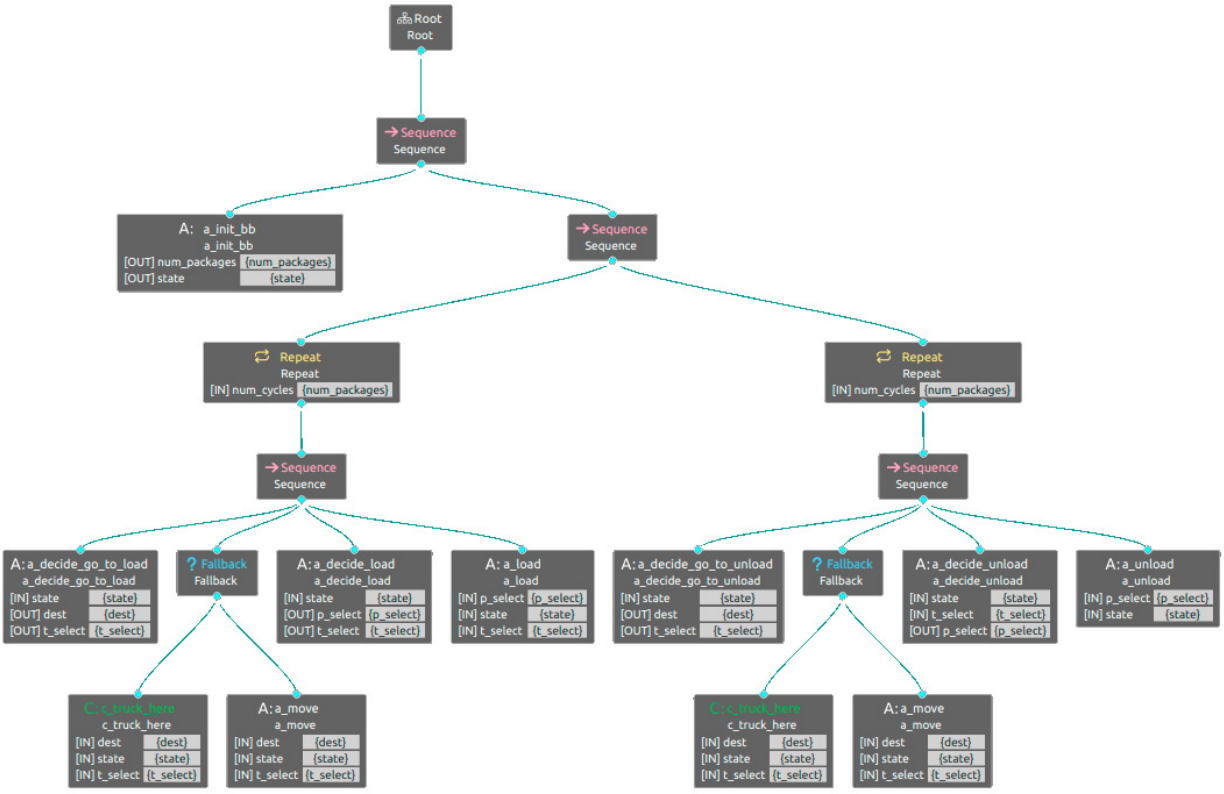


Fig. 3: Behaviour 2: first load all packages, then unload them

- `a_decide_go_to_load`: this node analyzes the current state and selects the location from which loading the next package. It selects a location l in which there is at least a package that needs to be transported to a different location, such that l is the closest location to any truck.
- `a_decide_go_to_unload`: this node analyzes the current state and selects the location for unloading the next package. It selects the closest location l to a truck t which contains a package that must be unloaded in l.
- `a_decide_load`: this node returns which package to load from the current location of the truck, which is passed as input parameter. The action is executed once a destination is selected after executing a `a_decide_go_to_load` action and it may also require a `a_drive_truck` action to move the truck to the selected location.
- `a_decide_unload`: this node returns which package to unload from the truck, which is passed as input parameter, at the current truck location. The action is executed once a destination for unloading the package is selected with `a_decide_go_to_unload`. It may possibly require as well to execute a `a_drive_truck` action to move the truck to the selected location, which will become the current truck location.

As a side note, these are all greedy decisions that do not guarantee the optimality of the plan.

Finally, **decorator nodes** are aimed at controlling the execution flow of the BT. All the nodes used in this work are predefined by the BT environment:

- `RetryUntilSuccessful`: this node forces to execute the sub-tree until it returns *Success*, or otherwise until the maximum number of attempts is reached.

- `ForceSuccess`: this node forces the result of executing a sub-tree to be always *Success*.
- `Repeat`: this node forces to repeat the execution of the sub-tree a given number of times.

Figure 3 shows the BT for Behaviour 2 where the tree below the second Sequence node is divided into two sub-trees. The execution of the tree on the left is repeated as many times as number of packages are defined in the problem in order to decide (i) the location to head for to pickup a package, (ii) the truck with which to move to such location (if necessary), and (iii) the package to load from the location. The sub-tree on the right performs a similar process but for unloading the packages previously loaded into the trucks.

4. Behaviour recognition process

Our behaviour recognition process works as follows. First, we define a set B of behaviours for the domain *Logistics* via BTs as described in section 3. Then, we calculate a plan π for solving a problem P of this domain with a planner. The goal of the recognition process is to identify which behaviour of B is followed by the plan π. In order to do so, we define several distance metrics. More particularly, we solve the problem P with each BT, obtaining a plan π_b for every $b \in B$. Then, π is compared to each π_b by applying some distance metrics; it is assumed that π follows the behaviour b that minimizes the distance to π_b; i.e., $\arg\min_{b \in B} dist(\pi, \pi_b)$.

We use different metrics to measure the distance between the behaviour defined by a BT, π_b, and a given plan, π. This section summarizes some existing metrics and it also introduces a new distance that allows us to capture subtle differences between plans that may have a significant impact in the behaviour recognition process.

4.1. Metrics for measuring distances between plans

This section summarizes several distance metrics found in the literature and shows how they apply to the plans in Table 2. The aim is to detect that plans A, C and D follow the same behaviour whereas plan B follows a different one. The work in [9], based on some previous work by the same authors [14], formalizes three different plan-plan distance metrics using plan actions or causal links, and sequence of states:

1. **Plan actions and causal links**: given a plan π, let $A(\pi)$ and $C(\pi)$ be the set of actions and causal links of π, respectively. The distance between π and π' is measured as the ratio of the number of actions (causal links) that do not appear in both plans to the total number of actions (causal links) appearing in one of them:

$$\delta_a(\pi, \pi') = 1 - \frac{|A(\pi) \cap A(\pi')|}{|A(\pi) \cup A(\pi')|} \qquad \delta_{cl}(\pi, \pi') = 1 - \frac{|C(\pi) \cap C(\pi')|}{|C(\pi) \cup C(\pi')|}$$

   In our example, $\delta_a(A, B) = 0.1667$ because plans A and B have 5 out of 6 actions in common whereas $\delta_a(A, C) = 1$ because plans A and C are completely different. Thus, we can say that δ_a properly discriminates between similar and dissimilar plans. However, while plans A and C are different, they follow they same behaviour since they only differ in the truck used in each plan; that is, one action parameter is different which implies that the corresponding actions are different. Moreover, despite the low value of $\delta_a(A, B)$, plans A and B follow different behaviours. Thus, two plans that look alike according to this metric is not an indication that they follow the same behaviour.
2. **Sequence of states**: given two sequences of states $(s_0, s_1, \ldots, s_k)$ and $(s'_0, s'_1, \ldots, s'_k)$ resulting from executing two plans π and π', and assume that $k' \leq k$. The distance between π and π' is defined as the average of the distances between state pairs $(s_i, s'_i), 0 \leq i \leq k'$, and each state $s_{k'+1}, \ldots, s_k$ is considered to contribute maximally (i.e., one unit) into the difference between two plans:

$$\delta_s(\pi, \pi') = \frac{1}{k} \times \left[\sum_{i=1}^{k'} \Delta^-(s_i, s'_i) + k - k' \right], \; where \quad \Delta^-(s, s') = 1 - \frac{|s \cap s'|}{|s \cup s'|}$$

   $\delta_s(A, D)$ involves calculating the distance between each pair of states s_i, s'_i: states s_2, s_3 and s_5 are equal to s'_2, s'_3 and s'_5, respectively; therefore, $\Delta(s_i, s'_i) = 0, i \in \{2, 3, 5\}$. Having that s_1 ={`(in P1 T1),(at`

`P2 L1),(at T1 L1),(at T2 L1)}` and s'_1=`{(at P1 L1),(in P2 T1),(at T1 L1),(at T2 L1)}`, then $\Delta(s_1, s'_1) = 0.667$ because these states have 2 out of 6 facts in common. The same value is obtained for $\Delta(s_4, s'_4)$. Therefore: $\delta(A, D) = 1/5 * (0.667 + 0 + 0 + 0.667 + 0 + 5 - 5) = 0.2668$. We also have that $\delta_s(B, A) = 0.6667$ and $\delta_s(A, C) = 0.7182$. Thus, δ_s enables to identify that A and D follow a similar behaviour and that A and B do not. However, this measure has to be tuned in the case of plans A and C, whose only difference lies in that they use different trucks.

3. **Landmark-based distance** [1]: this metric is based on the notion of *disjunctive landmark* of a problem, which is defined as a set of facts such that at least one of them must be achieved in every solution plan [7]. For example, three disjunctive landmarks can be found in the problem described in section 3: `{(in P1 T1),(in P1 T2)},{(in P2 T1),(in P2 T2)},{(at T1 L2),(at T2 L2)}`, meaning that every solution plan must achieve either the fact `(in P1 T1)` or `(in P1 T2)` (likewise for the other two disjunctive landmarks). Each fact in a disjunctive landmark is called a *disjunct*. The landmark distance is defined as the size of the symmetric difference of disjunctive landmark disjuncts satisfied by the two plans:

$$\delta_L(\pi, \pi') = \frac{1}{|\mathcal{L}_{>1}|} \sum_{\phi \in \mathcal{L}_{>1}} \frac{|\phi(\pi) \Delta \phi(\pi')|}{|\phi(\pi) \cup \phi(\pi')|}$$

where $\mathcal{L}_{>1}$ is the set of disjunctive landmarks of the problem, $\phi(\pi)$ is the collection of disjuncts satisfied by π and $A \Delta B = (A \cup B) - (A \cap B)$. In our example, $\delta_L(A, B) = 0$ because the disjuncts satisfied by plans A are B are the same, and $\delta_L(A, C) = 1$ because plan C satisfies the disjuncts other than the ones achieved by plan A. Therefore, for this example, δ_L suffers from the same limitations than δ_a.

4.2. A new plan distance

From the metrics exposed above, only δ_s takes into account the order of the actions in the plans. However, when different resources are used (e.g., trucks), the distance between two plans increases even though the actions are essentially the same. We introduce a novel metric, δ_{s+DL}, which is an extension of δ_s using disjunctive landmarks. The idea is that facts that belong to the same disjunctive landmark are considered to be the "same" fact. δ_{s+DL} is formulated in the same form as δ_s except that the state s_i generated after the application of an action is processed as: $s_i^{DL} = \phi(s_i) \cup \{f \in s_i : \neg \exists l \in \mathcal{L}_{>1} / f \in l\}$.

Let's compute $\delta_{s+DL}(A, C)$. The first state of plan A is s_1 =`{(in P1 T1),(at P2 L1),(at T1 L1),(at T2 L1)}` and the first state of plan C is s'_1=`{(in P1 T2),(at P2 L1),(at T1 L1),(at T2 L1)}`. After processing both states we obtain: $s_1^{DL} = s_1'^{DL}$ =`{{(in P1 T1),(in P1 T2)},(at P2 L1),(at T1 L1),(at T2 L1)}`, that is, both processed states indicate that `P1` is loaded in a truck, regardless which one. This way, $\Delta(s_1^{DL}, s_1'^{DL}) = 0$. In other words, those states that are different in δ_s are considered to be the same in s_i^{DL} after processing, and thus the distance returned by δ_{s+DL} is lower compared to δ_s.

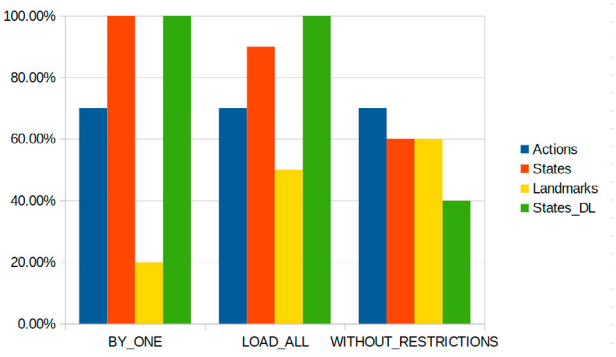
5. Results

For validating the behaviour recognition process described in Section 4 we run experiments with 10 problems of increasing size, from problem P_1 with one package and one truck to problem P_{10} with 19 packages and 5 trucks. We built three BTs representing the three behaviours described in Section 3 using Groot[3]. For each problem P_i and behaviour b_j we calculated a plan, $\pi_{i,j}$, with the BT executor based on the BehaviourTree.CPP library[4]. Likewise, from the original PDDL *Logistics*, we generated three domains to replicate each of the three behaviours by adding preconditions and effects to the basic actions. Figure 4a shows the `LOAD` action used for Behaviour 1 of Figure 2, which prevents from loading two packages in the same truck. Once the three domains are available, for each problem P_i and behaviour b_j we calculated a plan, $\rho_{i,j}$, with the LAMA planner [12].

[3] https://github.com/BehaviorTree/Groot
[4] https://github.com/BehaviorTree/BehaviorTree.CPP

```
( :action LOAD
:parameters
(?obj - obj ?truck - truck ?loc - location)
:precondition ( and
( truck-at ?truck ?loc )
( obj-at ?obj ?loc )
( nfull ?truck )
)
:effect ( and
( not ( obj-at ?obj ?loc ) )
( in-truck ?obj ?truck )
( full ?truck ) ( not (nfull ?truck))
) )
```

(a) `LOAD` action for Behaviour 1



(b) Successful recognition chance

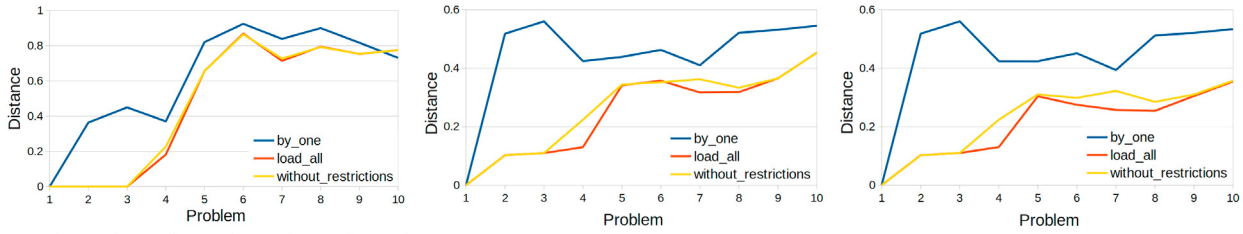
Fig. 4

A PDDL plan $\rho_{i,j}$ is compared to the three BT plans $\pi_{i,1}$, $\pi_{i,2}$ and $\pi_{i,3}$ using the distance metrics described in Section 4 to determine the behaviour followed by $\rho_{i,j}$. Behaviour recognition is considered **successful** if the distance between the $\rho_{i,j}$ and $\pi_{i,j}$ is not greater than the distance between $\rho_{i,j}$ and $\pi_{i,j'}, j' \neq j$. The probability of a successful behaviour recognition for each metric is shown in Figure 4b, where Actions, States, Landmarks and States_DL refer to δ_a, δ_s, δ_L and δ_{s+DL}, respectively. Figure 5 shows the value of each metric when comparing the plan $\rho_{i,load\_all}$ against the three BT plans for the 10 problems:

- Recognition based on δ_a (Actions) provides average 70% hit-rate (Figure 4b). This metric is very dependent on the size of the problem when identifying the `load_all` behaviour (Figure 5a). This is because there is no mechanism for choosing among equivalent resources (trucks) when generating $\rho_{i,load\_all}$, which entails the possibility of syntactical differences in plans representing the same behaviour, which in turn increases the distance between plans.
- Recognition based on δ_s (States) provides a 60-100% accuracy with an average of 83.3%. Figure 5b shows that this metric clearly distinguishes that the behaviour `by_one` is not similar to the `load_all` behaviour. This is because the difference between the selected actions is accumulated in the distance between the pairs of states being compared, which affects the final metric. The disadvantage of this technique, among other things, is that it will give a false negative result in the case of comparing plans that differ in the order of resolving goals in a complex plan.
- Landmark-based recognition is not giving high accuracy (only 40%) and is not stable (worst and best result for load_all is 20% and 60%). This method is not shown in Figure 5 because it cannot be used in simple cases because of the lack of disjunctive landmarks.
- Recognition based on δ_s^{DL} (States_DL) provides a 60-100% accuracy with an average of 86.7%. This metric improves over δ_s for recognizing the behaviour `load_all` (the distance values of the orange line are lower in Fig. 5c than in Fig.5b and the organge line is always below the other two lines) because δ_s^{DL} ignores the differences in resources used to achieve the goal.

All the metrics reflect a similarity of `load_all` and `without_restrictions` behaviors. The `without_restrictions` behavior implements the most effective plan for solving the problem without the conditions imposed by other behaviors and this makes the two behaviours very much alike. Moreover, States and StatesDL metrics show a clear difference between `by_one` and the other behaviors, which again is a reflection of the reality, given that `by_one` behavior is quite different with respect to the other two behaviors.

Lower recognition accuracy of δ_s and δ_s^{DL} for `without_restrictions` behavior can be explained by the proximity to `load_all`, by the differences in the order of the actions between the PDDL and BT plan, by the actual differences between the plans. LAMA returns the optimal PDDL plan, while this is not always



(a) Actions (b) States (c) StatesDL

Fig. 5: Comparison of different distance metrics to recognize behaviour 2 (Load_all)

the case for the BT plans. Also, lower accuracy of δ_s^{DL} vs δ_s can be explained by the detection of additional landmarks, which should be tuned manually.

6. Conclusions

This work presents a first step in the behaviour recognition in a planning context by using BTs. Plan distance metrics have been used to recognize the underlying behaviour, concluding that this approach has some limitations, mainly related with the use of resources and the order in which goals are solved. We are currently working on a new technique to directly compare the agent plan against the behaviour in the BT, without need to consider the particular used resource or the specific goal being solved.

References

[1] Bryce, D., 2014. Landmark-based plan distance measures for diverse planning, in: Twenty-Fourth International Conference on Automated Planning and Scheduling.
[2] Colledanchise, M., Almeida, D., Ögren, P., 2019. Towards blended reactive planning and acting using behavior trees, in: International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019, pp. 8839–8845.
[3] Colledanchise, M., Ögren, P., 2018. Behavior trees in robotics and AI: An introduction. CRC Press.
[4] Fox, M., Long, D., 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. Journal of Artificial Intelligence Research 20, 61–124.
[5] Freedman, R.G., Jung, H., Zilberstein, S., 2014. Plan and activity recognition from a topic modeling perspective, in: 24th International Conference on Automated Planning and Scheduling, ICAPS.
[6] Gan, W., Lin, J.C., Fournier-Viger, P., Chao, H., Yu, P.S., 2019. A survey of parallel sequential pattern mining. ACM Trans. Knowl. Discov. Data 13, 25:1–25:34.
[7] Hoffmann, J., Porteous, J., Sebastia, L., 2004. Ordered landmarks in planning. Journal of Artificial Intelligence Research 22, 215–278.
[8] McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., Wilkins, D., 1998. PDDL – The Planning Domain Definition Language.
[9] Nguyen, T.A., Do, M., Gerevini, A.E., Serina, I., Srivastava, B., Kambhampati, S., 2012. Generating diverse plans to handle unknown and partially known user preferences. Artificial Intelligence 190, 1–31.
[10] Palma, R., González-Calero, P.A., Gómez-Martín, M.A., Gómez-Martín, P.P., 2011. Extending case-based planning with behavior trees, in: Proceedings of the Twenty-Fourth International Florida Artificial Intelligence Research Society Conference, May 18-20, 2011, Palm Beach, Florida, USA.
[11] Ramírez, M., Geffner, H., 2009. Plan Recognition as Planning, in: 21th International Joint conference on Artifical Intelligence, IJCAI, AAAI Press. pp. 1778–1783.
[12] Richter, S., Westphal, M., 2010. The lama planner: Guiding cost-based anytime planning with landmarks. Journal of Artificial Intelligence Research 39, 127–177.
[13] Sohrabi, S., Riabov, A.V., Udrea, O., 2016. Plan recognition as planning revisited, in: 25th International Joint Conference on Artificial Intelligence, IJCAI, pp. 3258–3264.
[14] Srivastava, B., Nguyen, T.A., Gerevini, A., Kambhampati, S., Do, M.B., Serina, I., 2007. Domain independent approaches for finding diverse plans., in: IJCAI, pp. 2016–2022.
[15] Sukthankar, G., Goldman, R.P., Geib, C., Pynadath, D.V., Bui, H., 2014. Plan, Activity, and Intent Recognition: Theory and Practice. Morgan Kaufmann.