Optimized Execution of PDDL Plans using Behavior Trees

Extended Abstract

Francisco Martín Rico
Intelligent Robotics Lab
Rey Juan Carlos University
Fuenlabrada, Madrid
francisco.rico@urjc.es

Matteo Morelli
CEA list
Palaiseau, France
Matteo.MORELLI@cea.fr

Huascar Espinoza
CEA list
Palaiseau, France
Huascar.ESPINOZA@cea.fr

Francisco J. Rodríguez-Lera
University of León
León, Spain
fjrodl@unileon.es

Vicente Matellán Olivera
University of León
León, Spain
vicente.matellan@unileon.es

ABSTRACT

Robots need task planning to sequence and execute actions toward achieving their goals. On the other hand, Behavior Trees provide a mathematical model for specifying plan execution in an intrinsically composable, reactive, and robust way. PDDL (Planning Domain Definition Language) has become the standard description language for most planners. In this paper, we present a novel algorithm to systematically create behavior trees from PDDL plans to execute them. This approach uses the execution graph of the plan to generate a behavior tree. The most remarkable contribution of this approach is the algorithm to build a Behavior Tree that optimizes its execution by paralyzing actions, applicable to any plan, taking into account the actions' causal relationships. We demonstrate the improvement in the execution of plans in mobile robots using the ROS2 Planning System framework.

KEYWORDS

AI Planning; Multirobot; Behavior Trees

ACM Reference Format:
Francisco Martín Rico, Matteo Morelli, Huascar Espinoza, Francisco J. Rodríguez-Lera, and Vicente Matellán Olivera. 2021. Optimized Execution of PDDL Plans using Behavior Trees : Extended Abstract. In *Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), Online, May 3–7, 2021*, IFAAMAS, 3 pages.

1 INTRODUCTION

In this work we want to improve the execution of PDDL[4][3] plans and tasks in mobile service robots [8]. The execution of the plans is a crucial aspect for AI planning in robotics. Once the plan is generated in a computer, it must be carried out in the real world. This execution includes handing over the actions to the components that must perform them and keep checking the action requirements during their execution.

Currently, other approaches in ROS (Robotic Operating System), the *de facto* standard for robotic software development, are ROSPlan [1], only available in ROS 1, and SKIROS [9]. The work presented in this paper has been developed for the ROS2 Planning System (PlanSys2[1] in short) that aspires to be the reference framework for AI Planning in the next version of ROS: ROS2.

In particular, the main contribution of this paper is a novel algorithm to build a Behavior Tree[2] that optimizes the execution time of plans by explicitly producing parallel actions. Behavior Trees can adequately represent plans, including the causal constraints between the actions, and the evaluation of preconditions. We consider that this approach to apply Behavior Trees in Planning improves other approaches such as approaches like [7], [10], and [5]. The technical report at [6] contains a detailed comparison, along a detailed description of the algorithms of this contribution.

2 GRAPH REPRESENTATION FROM PLANS

Before applying our algorithm, we must create a *planning graph* G (Figure 1) that contains the effect-requirement dependencies of the actions of the plan (Listing 1). It is a directed acyclic graph defined by the tuple $G =< A, C >$, where A is the set of actions in a plan, corresponding to nodes of the graph. C is the set of directed arcs that represents the execution precedence of the actions.

PDDL listing 1 A PDDL plan.

```
0.00: (move r2d2 bedroom living)
5.00: (move r2d2 living kitchen)
```

Starting from plans generated by a PDDL planner, like the one in Listing 1, we generate a graph like the one shown in figure 1.

3 BEHAVIOR TREE BUILDING

Behavior Trees are used to code the execution of actions through a tree that contains nodes, which are leaves if they are end nodes (without child nodes). The leaves represent actions to be carried out, or conditions to check. The rest of the nodes define the execution flow of the tree. The main contribution is an algorithm automatically builds a Behavior Tree from any type planning graph. It is based on three main concepts:

Execution flows An execution flow starts from an action a_i node whose $R_{a_i} = \emptyset$, and add the actions in the path following the arc effect → requirement. In Figure 1 shows the three different flows in the planning graph shown in Figure 1.

[1] https://github.com/IntelligentRoboticsLabs/ros2_planning_system/

Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), U. Endriss, A. Nowé, F. Dignum, A. Lomuscio (eds.), May 3–7, 2021, Online.

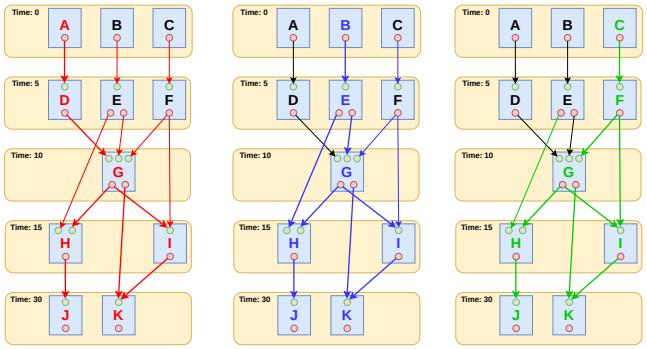

Figure 1: Different execution flows of the same planning graph.

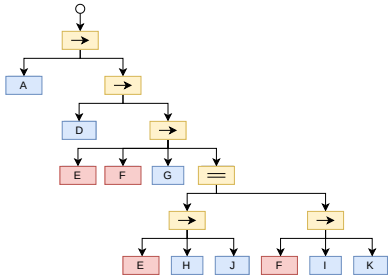

Figure 2: Behavior Tree generated from the red flow of Figure 1. Red rectangles are Waiting Nodes.

Singleton action Each action unit a_i in the generated Behavior Tree is a Singleton $S(a_i)$, i.e., it can appear in different points in the Behavior Tree, but it refers to the same action unit. If an action unit a_i has already returned SUCCESS when executed in one branch, it will return SUCCESS if it is ticked in any other branch.

Waiting nodes This control node refers to an action unit a_i, and denominates $W(a_i)$. It returns RUNNING if a_i has never returned SUCCESS.

Our algorithm, detailed in [6], automatically creates a behavior Tree like 2 that represents the plan, where each leaf is a subtree like shown in Figure 3, where the requirements are checked at runtime, the action is executed, and the effects are applied.

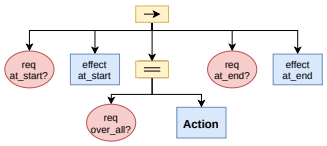

Figure 3: Expansion of each action unit.

4 EVALUATION

The scenario used in this evaluation is the SciRoc Restaurant test, but including one or more robots to see how execution can be optimized by performing various actions in parallel.

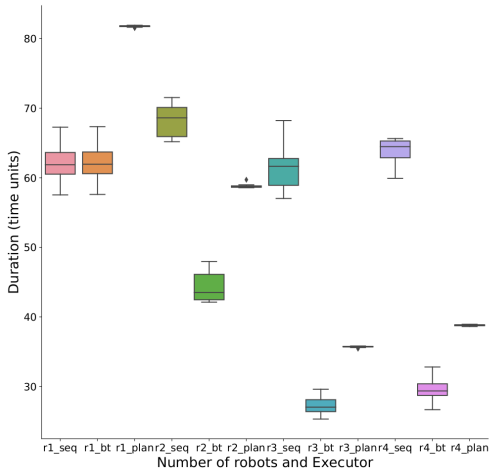

Figure 4: Starting from left, total time for execution with one robot (r1_seq, r1_bt, r1_plan), two robots (r2_seq, r2_bt, r2_plan), three robots (r3_seq, r3_bt, r3_plan) and four robots (r4_seq, r4_bt, r4_plan).

This optimization in the execution of actions is reflected in the complete plan's execution time, shown in Figure 4. As more robots are added, sequential execution maintains the same duration, since it does not optimize anything. Execution time according to plan decreases when more robots are included. The execution based on Behavior Tree shows the best performance in all configurations. The execution based on the planner is the most deterministic one due to planning that takes into account the maximum times per action. It should be noted that when resources are increased, in this case, robots, but there are no tasks (tables in this scenario) for every robot, performance deteriorates.

5 CONCLUSIONS

This paper presents a proposal for using Behavior Trees to execute plans generated by a PDDL-based AI planner. Coding a plan as Behavior Tree is a compact way to represent and execute a robot action plan. Major contribution of the paper is the algorithm capable of transforming any plan into a Behavior Tree in a systematic way. This solution creates a planning graph from the plan and makes the tree recursively. Different types of nodes are used to build the Behavior Tree such as the singleton action node and the wait node to improve the efficiency of parallel excution of actions. The generated Behavior Tree is so optimized to execute in parallel all the possible actions in a plan, preserving the causal relationships of the actions. Another contribution is the execution an action as soon as its requirements are available, even before established in the plan.

6 ACKNOWLEDGEMENT

This work was supported by the EU-funded projects RobMoSys ITP MROS under Grant Agreement No. 732410

REFERENCES

[1] Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni, Bram Ridder, Arnau Carreraa, Narcís Palomeras, Natàlia Hurtós, and Marc Carrerasa. 2015. ROSPlan: Planning in the Robot Operating System. In *Proceedings of the Twenty-Fifth International Conference on International Conference on Automated Planning and Scheduling* (Jerusalem, Israel) *(ICAPS'15)*. AAAI Press, 333–341.

[2] Michele Colledanchise and Petter Ogren. 2018. *Behavior Trees in Robotics and AI: An Introduction*. https://doi.org/10.1201/9780429489105

[3] Maria Fox and Derek Long. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)* 20 (12 2003), 61–124. https://doi.org/10.1613/jair.1129

[4] Malik Ghallab, Craig Knoblock, David Wilkins, Anthony Barrett, Dave Christianson, Marc Friedman, Chung Kwok, Keith Golden, Scott Penberthy, David Smith, Ying Sun, and Daniel Weld. 1998. PDDL - The Planning Domain Definition Language. (08 1998).

[5] Eleonora Giunchiglia, Michele Colledanchise, Lorenzo Natale, and Armando Tacchella. 2019. Conditional Behavior Trees: Definition, Executability, and Applications. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. IEEE, 1899–1906.

[6] Francisco Martín, Matteo Morelli, Huascar Espinoza, Francisco J. G. Lera, and Vicente Matellán. 2021. Optimized Execution of PDDL Plans using Behavior Trees. arXiv:2101.01964 [cs.RO]

[7] Alejandro Marzinotto, Michele Colledanchise, and Petter Ogren. 2014. Towards a unified behavior Trees framework for robot control. *Proceedings - IEEE International Conference on Robotics and Automation*, 5420–5427. https://doi.org/10.1109/ICRA.2014.6907656

[8] Illah Nourbakhsh and Michael Genesereth. 1996. Assumptive Planning and Execution: a Simple, Working Robot Architecture. *Autonomous Robots* 3, 1 (March 1996), 49 – 67.

[9] F. Rovida, B. Grossmann, and V. Krüger. 2017. Extended behavior trees for quick definition of flexible robotic tasks. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 6793–6800.

[10] Haotian Zhou, Huasong Min, and Yunhan Lin. 2019. An Autonomous Task Algorithm Based on Behavior Trees for Robot. In *2019 2nd China Symposium on Cognitive Computing and Hybrid Intelligence (CCHI)*. IEEE, 64–70.