Learning Behavior Trees for Robotic Task Planning by Monte Carlo Search over a Formal Grammar

Emily Scheide, Graeme Best, Geoffrey A. Hollinger
Collaborative Robotics and Intelligent Systems (CoRIS) Institute
Oregon State University
Corvallis, Oregon 97331
Email:{scheidee, bestg, geoff.hollinger}@oregonstate.edu

***Abstract*—This extended abstract presents a method of learning behavior trees for robotic task and motion planning, which alleviates the need for time-intensive manual design. Our method involves representing a set of behavior trees as a formal grammar and searching over this grammar by means of a generalization of Monte Carlo tree search for directed acyclic graphs. We present promising preliminary results for a marine target search and response scenario, and the learned behavior tree compares well with a manually designed tree. It is notable that the learned tree contains several, but not all, sub-trees that are crucial and present in the manually designed tree. Our results motivate future investigation of ways to learn for our sparse reward functions and to better combine promising sub-trees.**

I. Introduction

Many modern autonomous robots operate through the execution of actions within a predefined control architecture. The manual design of control architectures is often infeasible due to it being inherently time-intensive and requiring expert knowledge, particularly as robotic tasks become increasingly complex or applications require larger multi-robot teams. This motivates the need to automatically generate control architectures that function well in challenging task domains.

Recently, behavior trees have become a popular control architecture in robotics and computer games. They offer advantages in readability, recursivity, and modularity [1, 3, 10, 13, 15] as compared to finite state machines, decision trees, and various other controlled hybrid systems [3, 6]. These advantages are inherent to the behavior tree design, which is built with respect to tasks rather than states. Put simply, a behavior tree is a directed rooted tree that is comprised of leaf nodes, which evaluate conditions and activate actions, and internal nodes, which describe a logic structure. Behavior tree operation occurs through the switching between a number of tasks, based on changing observed input signals [3]. Due to the advantages such a structured tree brings, it is more feasible to manually design behavior trees than state-based methods.

Even so, the manual design of a behavior tree can still become too time-intensive. To meet the increasing demand for autonomous robotics, it is vital that this design be as expedited as possible, without sacrificing functionality. In order to expedite this process given the difficulties of manual design, we require an automatic behavior tree generation method that is robust to task complexity. Given a set of robot capabilities and a task simulator, the goal of the method is to find the behavior tree structure that maximizes the expected reward.

In this paper, we propose a novel behavior tree generation method. This method learns an optimal behavior tree by searching over our formal grammar that represents the set of valid, well-structured behavior trees. The search is carried out by a generalization of Monte Carlo tree search (MCTS). Unlike MCTS [2], our method searches over a directed acyclic graph, which allows us to further optimize the learning process. Our MCTS generalization is based on [7], with improvements for propagating information. It also periodically restarts, which prevents the search from becoming stuck at local maxima, and updates the grammar to subsequently guide the search toward desirable behavior trees.

We demonstrate this method within the domain of marine target search and response. The results show that our method is capable of generating a behavior tree with several optimized sub-trees pertaining to relevant tasks. While these results are promising, they also motivate several avenues of improvement that we intend to work on in the future.

II. Related Work

Due to the manipulability and modularity of behavior trees, they are feasible to restructure using reinforcement learning methods. Genetic programming [5, 11, 12, 14] and Q-learning [1, 8, 9] have both been implemented with the goal of learning optimal behavior tree structure with various guiding criteria. A key difference between these various approaches is how they represent a behavior tree. One such representation is a grammar, which has the advantage of having a well-defined set of rules that tell you how to incrementally build all behavior trees. This representation makes it feasible to search over the set of all valid behavior trees. The grammar we design in this paper generalizes the grammars in [8, 12, 14] to enforce a more functional behavior tree structure.

Given this grammar as our behavior tree representation, we need to design algorithms to search for the sequence of production rules that constructs the optimal behavior tree. Monte Carlo search has been shown to be an effective algorithm for searching over a grammar [7]. In this paper, we adapt MCTS [2] to search over a grammar by means of a directed acyclic graph instead of a tree. This allows for the easier propagation of rewards throughout the tree, which further optimizes the learning process.

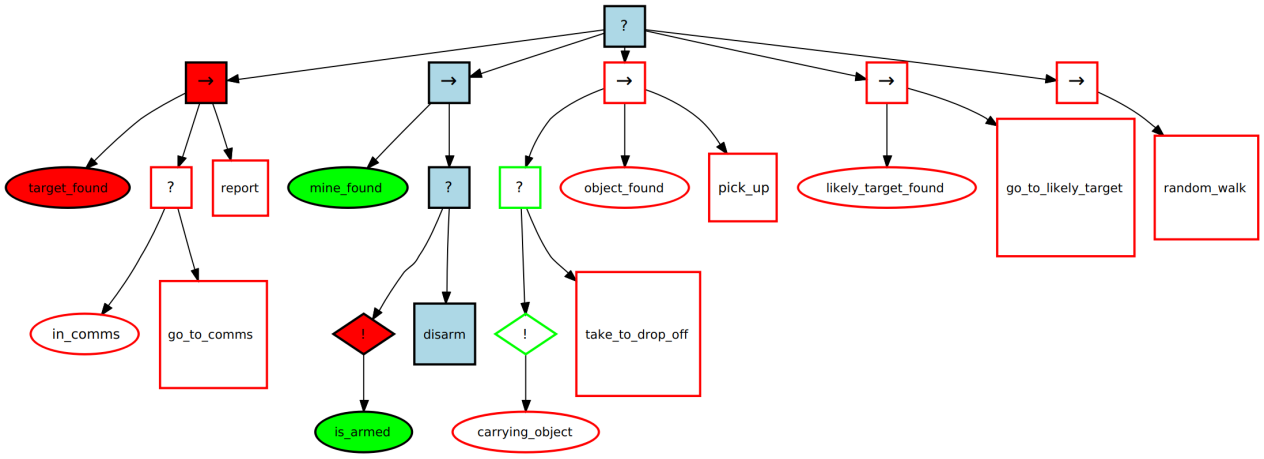

Fig. 1. A manually designed behavior tree for a marine target search and response application. Leaf squares are *actions*, ovals are *conditions*, ? denotes a *fallback*, → denotes a *sequence*, and ! denotes a *not-decorator*. The flow of behavior tree execution is from left to right, and is visualized by the coloring of the nodes. Red denotes a node that has failed, green denotes a node that has succeeded, and blue denotes that a node is currently executing. Unshaded nodes are currently inactive as they are currently not being evaluated or performed by the behavior tree.

III. Background on Behavior Trees

Behavior trees are directed rooted trees, which are comprised of control flow, execution, and decorator nodes. The types of control flow nodes are *fallback*, *sequence*, and *parallel*, and they determine which nodes are active. Execution node types are *action* nodes, which trigger execution of robot actions, and *condition* nodes, which are either true or false depending on the state of the robot and the world. Decorator nodes have one child *condition* node by definition, and, given the state of that *condition* node as input, return a state. A common example is the *not-decorator* node, which returns the logical complement of the child condition [3]. An example behavior tree for our marine target search and response scenario is shown in Fig. 1.

IV. Problem Formulation

A robot's capabilities are described as a set of actions and its sensors are described as a set of Boolean conditions. These actions and conditions are further divided into groups by category of application. We define this set of groups as $\mathcal{G}$, with $g \in \mathcal{G}$ and $g = [\mathcal{A}_g, \mathcal{C}_g]$, where $\mathcal{A}_g$ is the set of actions and $\mathcal{C}_g$ is the set of conditions in a group. These are introduced to guide the learning to connect related actions and conditions.

Additionally, for all $g \in \mathcal{G}$, the associated sets of actions, $\mathcal{A}_g$, and conditions, $\mathcal{C}_g$, must be compiled into a behavior tree, b. Where $\mathcal{B}$ is the set of all valid behavior trees, each $b \in \mathcal{B}$ can be evaluated by running it through a black box simulator that returns a reward, $f(b)$.

Ultimately, we aim to find the behavior tree, $b^* \in \mathcal{B}$, that has the maximum reward; i.e.,

$$b^* = \underset{b \in \mathcal{B}}{\operatorname{argmax}}\, f(b). \tag{1}$$

The goal is for the resulting behavior tree to be the most efficient and well-suited for autonomously guiding the robot through the execution of all tasks within its task domain.

V. Behavior Tree Learning Method

We are proposing a new method for this behavior tree design problem. This method involves representing a set of behavior trees as a formal grammar. A generalization of MCTS is used to search for the optimal behavior tree over a directed acyclic graph. We describe these components as follows.

A. Behavior Tree Formal Grammar

We design a set of production rules, which together form a formal grammar. This set of rules guides the autonomous production of all valid behavior trees, and ensures that the structure of each is functional. This set of rules is universal, and is applicable to any robot or scenario, as it takes in the groups of actions, $\mathcal{A}_g$, and conditions, $\mathcal{C}_g$, as input.

The structure the formal grammar enforces has a *fallback* node at the root, and then a layer of *sequence* nodes. Next, it allows *fallback*, *action*, *condition*, or *not-decorator* nodes with the constraint that *conditions* must appear to the left of *actions*. Finally, if a *fallback* was chosen at the previous level, the deepest level is allowed to contain *action*, *condition*, or *not-decorator* nodes only. We also enforce that each sub-tree only contains *actions* and *conditions* that are in the same group.

We define this formal grammar, $F = \{T, N, P, S\}$. Terminal characters are $T = \{?, \rightarrow, !, [a], (c),), (\}$, where ? denotes *fallback*, $\rightarrow$ denotes *sequence*, ! denotes *not-decorator*, $[a]$ denotes an *action* $a \in A_g$, $g \in \mathcal{G}$, (c) denotes a *condition* $c \in C_g$, $g \in \mathcal{G}$, (denotes going down a level, and) denotes going up a level. Non-terminal characters are $N = \{S, s, s^+, s_g, f_g, A_g, C_g, \check{C}_g, r_g^1, l_g^1, r_g^2, l_g^2\}$, where subscript g denotes a character for every $g \in \mathcal{G}$. The production rules, P, are defined in Table I, and S is the start character. Note that r_g^1 and l_g^1 pertain to nodes added below *sequence* nodes, $\rightarrow$, and r_g^2 and l_g^2 appear only in the deepest level of the tree. We also enforce that a particular *action* or *condition* can only appear once in a sub-tree, and all s_g must have a unique group, g.

TABLE I
BEHAVIOR TREE FORMAL GRAMMAR PRODUCTION RULES.
Terminal characters are shown in red and non-terminals in blue.

Setup a *fallback* node with *sequence* sub-trees:		
$S \rightarrow ?\ (\ s\ s^+\ )$	$s^+ \rightarrow s\ s^+$	$s^+ \rightarrow s$
Sub-tree structure, $\forall g \in \mathcal{G}$**:**		
$s \rightarrow s_g$	$s_g \rightarrow \rightarrow\ (\ A_g\ r_g^1\ )$	$s_g \rightarrow \rightarrow\ (\ f_g\ r_g^1\ )$
$s_g \rightarrow \rightarrow\ (\ l_g^1\ A_g\ )$	$s_g \rightarrow \rightarrow\ (\ l_g^1\ f_g\ )$	$f_g \rightarrow ?\ (\ A_g\ r_g^2\ )$
$f_g \rightarrow ?\ (\ l_g^2\ A_g\ )$	$r_g^1 \rightarrow f_g\ r_g^1$	$r_g^1 \rightarrow f_g$
$r_g^1 \rightarrow A_g\ r_g^1$	$r_g^1 \rightarrow A_g$	$l_g^1 \rightarrow l_g^1\ f_g$
$l_g^1 \rightarrow f_g$	$l_g^1 \rightarrow l_g^1\ \check{C}_g$	$l_g^1 \rightarrow \check{C}_g$
$r_g^2 \rightarrow A_g\ r_g^2$	$r_g^2 \rightarrow A_g$	$l_g^2 \rightarrow l_g^2\ \check{C}_g$
$l_g^2 \rightarrow \check{C}_g$	$\check{C}_g \rightarrow C_g$	$C_g \rightarrow !\ (\ C_g\ )$
Insert actions $\forall a \in \mathcal{A}_g$ **and conditions** $\forall c \in \mathcal{C}_g$**:**		
$A_g \rightarrow [a]$	$C_g \rightarrow (c)$	

B. Monte Carlo Search over a Formal Grammar

Given this grammar, F, which represents all possible behavior trees, we need a method of determining which behavior tree word within F is optimal. In other words, which valid behavior tree, b^*, is best suited for a given robot and application. We achieve this through a new generalization of MCTS [2].

Our algorithm searches over a directed acyclic graph, $\mathcal{D}$, in which a node can have multiple parents due to a word in F having multiple possible derivations. This means we need to generalize MCTS to be applicable to DAGs. Like MCTS, our algorithm is comprised of four stages: *selection*, *expansion*, *simulation*, and *backpropagation*. Each edge in $\mathcal{D}$ represents a single application of a production rule. Each node represents a partial or complete behavior tree, where a behavior tree is partially complete if it contains at least one non-terminal character. Similarly, terminal nodes are valid behavior trees.

The main differences to standard MCTS are in how we do *expansion* and *backpropagation* for a DAG. In the *expansion* phase, when a new node is added, edges are connected to all possible parents. During *backpropagation*, the reward is propagated up all possible trajectories from a terminal node to the root node. This method is based on [7] with two novel improvements. Firstly, we actively search for and connect ancestors within a fixed number of rules to each new child node. This allows information to more quickly propagate throughout the DAG. Secondly, we periodically restart the search, and during each restart we add additional rules to the grammar that represent the most promising learned sub-trees.

VI. MARINE TARGET SEARCH AND RESPONSE

We evaluate the efficacy of our method with respect to a marine target search and response scenario simulation, although our method is applicable to any task planning problem that can be represented by behavior trees. The robot moves through a marine environment locating targets, disarming sea mines, and retrieving objects of interest. The goal is to maximize the occurrence of these actions. Each of the actions associated with these goals has a reward, and these rewards are linearly combined to evaluate the behavior tree guiding the robot's actions. The robot's tasks and states are sorted into groups of related actions and conditions. These groups are as follows: g_1 = [{go to comms, report},{in comms, target found, at surface}], g_2 = [{disarm},{mine found, is armed}], g_3 = [{pick up, take to drop off},{object found, carrying object}], g_4 = [{go to likely target},{likely target found}], g_5 = [{random walk},{}], and g_6 = [{shortest path},{}]. The robot moves on a roadmap with a random distribution of targets, mines and objects of interest over the vertices. We define the robot's sensor model to be a probability distribution, which is updated via Bayes' rule.

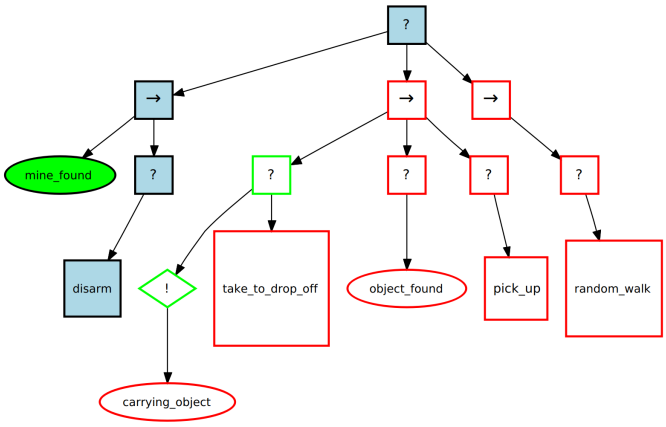

Fig. 2. A behavior tree generated by our method for the same scenario as the manually designed tree shown in Fig. 1. Three sub-trees were successfully learned for groups g_1, g_2, and g_5, but is not as complete the manually designed designed tree.

The learned behavior tree is shown in Fig. 2. For comparison, we manually designed a behavior tree (Fig. 1), which is comprised of a series of sub-trees, each only containing actions and conditions from the same group. We can see that several functional sub-trees were successfully learned. This is a promising result, but our goal is to generate a behavior tree containing a sub-tree for each crucial group within $\mathcal{G}$. Currently, our learned tree is missing sub-trees relating to groups that are present in the manually designed tree.

VII. DISCUSSION AND FUTURE WORK

Our method is promising in that the generated behavior tree contains correctly learned sub-trees. However, this tree is not as complete as the manually designed tree, and we believe this is for a number of reasons. Firstly, the reward structure in the DAG is very sparse, which makes it difficult to learn. Secondly, we observed that our method finds correct sub-trees, but does not always learn to compile them in one tree. In the future, we plan to investigate ways to learn for our sparse reward function and to better combine promising sub-trees. We also aim to generalize and adapt our method to multi-robot teams [4], and demonstrate its effectiveness for other robotics applications. We are also interested in extending other learning methods, such as Q-learning and genetic algorithms.

VIII. ACKNOWLEDGEMENTS

This research was funded in part by Office of Naval Research grant N00014-17-1-2581.

References

[1] Bikramjit Banerjee. Autonomous acquisition of behavior trees for robot control. In *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 3460–3467, 2018.

[2] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *IEEE Trans. on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[3] Michele Colledanchise and Petter Ögren. *Behavior trees in robotics and AI: An introduction*. CRC Press, 2018.

[4] Michele Colledanchise, Alejandro Marzinotto, Dimos V Dimarogonas, and Petter Ögren. The advantages of using behavior trees in mult-robot systems. In *Proc. of Int. Symp. on Robotics (ISR)*, 2016.

[5] Michele Colledanchise, Ramviyas Parasuraman, and Petter Ögren. Learning of behavior trees for autonomous agents. *IEEE Trans. on Games*, 11(2):183–189, 2018.

[6] Peter I. Cowling, Michael Buro, Michal Bida, Adi Botea, Bruno Bouzy, Martin V. Butz, Philip Hingston, Hector Muñoz-Avila, Dana Nau, and Moshe Sipper. Search in Real-Time Video Games. In Simon M. Lucas, Michael Mateas, Mike Preuss, Pieter Spronck, and Julian Togelius, editors, *Artificial and Computational Intelligence in Games*, volume 6. Schloss Dagstuhl, 2013.

[7] Frédéric de Mesmay, Arpad Rimmel, Yevgen Voronenko, and Markus Püschel. Bandit-based optimization on graphs with application to library performance tuning. In *Proc. of Int. Conf. on Machine Learning (ICML)*, page 729–736, 2009.

[8] Rahul Dey and Chris Child. QL-BT: Enhancing behaviour tree design and implementation with Q-learning. In *Proc. of IEEE Conf. on Computational Intelligence in Games (CIG)*, 2013.

[9] Yanchang Fu, Long Qin, and Quanjun Yin. A reinforcement learning behavior tree framework for game AI. In *Proc. of Int. Conf. on Economics, Social Science, Arts, Education and Management Engineering (ESSAEME)*, 2016.

[10] Andreas Klöckner. Interfacing behavior trees with the world using description logic. In *Proc. of AIAA Guidance, Navigation, and Control (GNC) Conference*, page 4636, 2013.

[11] Chong-U Lim, Robin Baumgarten, and Simon Colton. Evolving behaviour trees for the commercial game defcon. In *Proc. of European Conference on the Applications of Evolutionary Computation*, pages 100–110, 2010.

[12] Aadesh Neupane and Michael Goodrich. Learning swarm behaviors using grammatical evolution and behavior trees. In *Proc. of Int. Joint Conference on Artificial Intelligence (IJCAI)*, pages 513–520, 2019.

[13] Petter Ögren. Increasing modularity of UAV control systems using computer game behavior trees. In *Proc. of AIAA Guidance, Navigation, and Control Conf.*, page 4458, 2012.

[14] Diego Perez, Miguel Nicolau, Michael O'Neill, and Anthony Brabazon. Evolving behaviour trees for the Mario AI competition using grammatical evolution. In *Proc. of European Conference on the Applications of Evolutionary Computation*, pages 123–132, 2011.

[15] Christopher Iliffe Sprague, Özer Özkahraman, Andrea Munafo, Rachel Marlow, Alexander Phillips, and Petter Ögren. Improving the modularity of AUV control systems using behaviour trees. In *Proc. of IEEE/OES Autonomous Underwater Vehicle Workshop (AUV)*, 2018.