BT Expansion: a Sound and Complete Algorithm for Behavior Planning of Intelligent Robots with Behavior Trees

Zhongxuan Cai,[1] **Minglong Li,**[1] **Wanrong Huang,**[2] **Wenjing Yang***[1]

[1] Institute for Quantum Information & State Key Laboratory of High Performance Computing, College of Computer, National University of Defense Technology, China

[2] Artifcial Intelligence Research Center, National Innovation Institute of Defense Technology, China

{caizhongxuan, liminglong10, huangwanrong12, wenjing.yang}@nudt.edu.cn

Abstract

Behavior Trees (BTs) have attracted much attention in the robotics field in recent years, which generalize existing control architectures and bring unique advantages for building robot systems. Automated synthesis of BTs can reduce human workload and build behavior models for complex tasks beyond the ability of human design, but theoretical studies are almost missing in existing methods because it is difficult to conduct formal analysis with the classic BT representations. As a result, they may fail in tasks that are actually solvable. This paper proposes BT expansion, an automated planning approach to building intelligent robot behaviors with BTs, and proves the soundness and completeness through the state-space formulation of BTs. The advantages of blended reactive planning and acting are formally discussed through the region of attraction of BTs, by which robots with BT expansion are robust to any resolvable external disturbances. Experiments with a mobile manipulator and test sets are simulated to validate the effectiveness and efficiency, where the proposed algorithm surpasses the baseline by virtue of its soundness and completeness. To the best of our knowledge, it is the first time to leverage the state-space formulation to synthesize BTs with a complete theoretical basis.

Introduction

Behavior Trees (BTs) originate from the computer games industry to control non-player characters (NPCs) (Millington and Funge 2009), e.g. computer controlled opponents, and become a popular control architecture for intelligent robots in recent years due to their unique properties (Colledanchise and Ögren 2018): (1) *Generality*. BTs are proven to generalize many well-known control architectures including finite state machines, the subsumption architecture (Brooks 1986), teleo-reactive programs (Nilsson 1994) and so on. (2) *Modularity*. Each subtree of a BT is also a BT that can run independently as a modular sub-behavior. (3) *Reactivity*. BTs can react to unexpected environment changes by frequent ticks to activate behaviors in runtime. BTs have been used in a wide range of the robotic field, including robot manipulation (Rovida et al. 2018), unmanned aerial vehicles (Lan et al. 2018), mobile robots (Banerjee 2018) and so on.

*Corresponding author

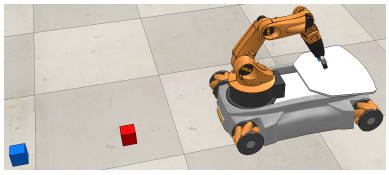


Figure 1: An example of blended reactive planning and acting of a BT-based robot which aims to grasp the blue cargo. If a red obstacle blocks the way, the BT can be expanded at runtime to clear the way first. If someone reverses the action by putting the obstacle back again, the robot will clear the way again without replanning. If someone helps to clear the way, the robot will skip the clear action without replanning.

Although BTs were originally created to facilitate the manual design of agent behaviors, its properties are beneficial for both manual and automated synthesis (Iovino et al. 2020). Besides the intrinsic advantages of automated synthesis, e.g. reducing human workload and exploiting behaviors beyond the ability of human design in complex tasks (Neupane and Goodrich 2019), BTs also bring the advantage of *blended reactive planning and acting* for intelligent robots (Colledanchise, Almeida, and Ögren 2019), which are robust to external disturbances, as illustrated in Fig. 1. However, due to the gap between the BT representation and formal analysis, existing methods either massively modify the definition of BTs with the loss of BT's advantages and compatability, or do empirical demonstrations without theoretical guarantees, which may fail in actually solvable tasks.

In this paper, we propose BT expansion, a sound and complete algorithm for behavior planning of intelligent robots with BTs. The behavior planning is theoretically studied based on the STRIPS-style planning (Fikes and Nilsson 1971) and the state-space formulation of BTs (Colledanchise and Ögren 2017). The algorithm starts with a primary BT and expands this initial BT to find a solution, which terminates in finite time and a solution is guaranteed as long as the problem is solvable. The advantages of *blended reactive planning and acting* are formally discussed though BT's region of attraction, by which robots are robust to external disturbances. With the aid of completeness, BT expansion yields a stronger property that robots are guaranteed to successfully perform tasks under any resolvable disturbances.

Experiments simulated with a mobile manipulator and test sets are conducted to verify the effectiveness and efficiency of the algorithm and its superiority to the baseline.

The contribution of this paper is summarized as follows.

1. We propose a behavior planning algorithm of intelligent robots with BTs and prove the soundness as well as the completeness of the algorithm. We believe it is the first time to introduce the state-space formulation of BTs to synthesize BTs with a complete theoretical basis.
2. We formally analyze the advantages of behavior planning with BTs from the perspective of region of attraction, by which robots with BT expansion are robust to any resolvable external disturbances.
3. We conduct experiments with a simulated mobile manipulator and test sets, where our algorithm surpasses the latest approach to BT planning. The empirical study demonstrates the effectiveness and efficiency of our approach.

Background

Behavior tree is a directed rooted tree consisting of control flow nodes, execution nodes and a root (Colledanchise and Ögren 2017). The control flow nodes are internal nodes with triggering logic of their children. The execution nodes are leaf nodes which check task conditions or perform actions.

The execution of a BT starts from the root, which ticks its children and gets returned status. The child nodes return *success*, *running* or *failure* to the parents. The return value of a control flow node is determined by the return value of its children. And for execution (leaf) nodes, the return value is defined according to the task states and actions. We describe the two mostly used control flow nodes as follows:

Sequence: The sequence node ticks its children from left to right, as illustrated in Fig. 2a. Only if one child returns success does it ticks the next child. If a child returns failure, the sequence node stops ticking and immediately returns failure. It returns success only when all children return success. For other cases it returns running. The sequence node is usually represented by a $\rightarrow$ in a box.

Fallback: The fallback node (also known as selector) ticks its children from left to right, as illustrated in Fig. 2b. If a child returns running or success, it also returns the same status. Only if a child returns failure does it ticks the next child. If all children fail, the fallback node returns failure. The fallback node is usually represented by a ? in a box.

Besides the internal control flow nodes, the leaves of BTs are composed of two types of execution nodes:

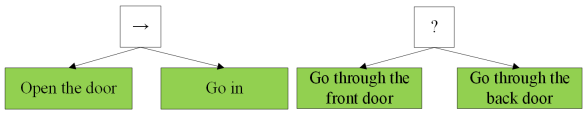


Figure 2: (a) The sequence node for passing a door executes its children from left to right and returns success only when both children succeed. (b) The fallback node for entering a room succeeds when any one of its children returns success.

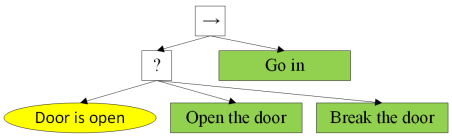


Figure 3: An example of a BT for entering a room. The root is always the parent of the top node and is usually omitted.

Condition: The condition node checks whether a condition is satisfied, and returns success or failure accordingly.

Action: The action node perform an action of the agent, and returns success, running or failure accordingly.

Fig. 3 illustrates a BT for reactive robots entering a room. Behavior trees intrinsically bring modularity and reactivity to agent behavior modeling. Each subtree is also a BT that can execute independently. The reactivity is mainly brought by the fallback node and the frequent tick of condition nodes during acting. The BT in Fig. 3 can react and adapt to different run-time situations, e.g. when the robot fails to open the door it tries to break the door, and when an external agent opens the door, then the next tick of the BT will directly make the robot go in, skipping the action *Break the door*.

Related Work

The planning community has developed solid methods for robotics, e.g. searching in state or plan space (Ghallab, Nau, and Traverso 2016), and planning for temporal logic tasks (Guo, Tumova, and Dimarogonas 2016). However, the integration of planning and BT synthesis has not been studied until recently(Colledanchise, Almeida, and Ögren 2019). Existing approaches to synthesizing BTs in the literature include evolutionary methods, learning methods and classic analytic methods (Iovino et al. 2020).

The evolutionary methods initialize a group of simple BTs and evolve them to produce better BTs with genetic-like algorithms (Neupane and Goodrich 2019; Lim, Baumgarten, and Colton 2010). However these methods are inefficient because simulations are needed to evaluate each BT, and they have no guarantee of feasible results in limited time.

Learning methods are mostly based on reinforcement leaning (RL) (de Pontes Pereira and Engel 2015) and imitation learning (French et al. 2019). The RL approaches parameterize some learning nodes with learned policy. A learning node itself is a complex policy which breaks the concise definition of BTs and interpretability. The imitation learning methods learn BTs from human demonstration (French et al. 2019), which relies on human ability and the result may not meet expectations (Iovino et al. 2020).

The classic analytic methods utilize classic algorithms to synthesize BTs, like greedy algorithms (Colledanchise and Ögren 2018), the hierarchical task network (Rovida, Großmann, and Krüger 2017), automated planning (Colledanchise, Almeida, and Ögren 2019), etc. One major challenge is that although BTs are quite readable for humans, their representation is not suitable for formal analysis.

Our work is closely related to the recent work (Colledanchise, Almeida, and Ögren 2019) synthesizing BTs through

automated planning, which exploits BT's reactivity and modularity in robot planing, highlighting advantages in terms of *blended reactive planning and acting* with continually hierarchical monitoring. Apart from the pioneering work, they focus on empirical studies with few theoretical insights. As shown by our experiments, their method may fail even if a solution actually exists. On the contrast, our algorithm is proven to be sound and complete. We formally discuss the advantages and verifies our approach with intensive theoretical and empirical studies.

Reactive Behavior Planning through BT Expansion

In this section, we propose BT expansion, an algorithm expanding a BT from scratch to solve the behavior planning problem. We first formulate the problem with STRIPS-style planning and the state-space formulation of BTs as the basis of theoretical study, then propose the one-step expansion of a BT for condition satisfaction. Finally we propose the entire algorithm and prove its soundness and completeness.

Problem Formulation

To conduct formal analysis, we begin with the state-space formulation of BTs (Colledanchise and Ögren 2017). Without loss of generality, an environment state at a point in time is represented by a set of positive literals, and those that are not included are assumed to be $False$, as is in STRIPS-style planning (Fikes and Nilsson 1971). In the literature there are different methods for checking whether a condition c holds in a state s, e.g. checking whether $s \cup \neg c$ is contradictory. This paper utilizes $c \subseteq s$ for clarity, i.e. all literals of the condition c is included in s.

Definition 1 (Behavior Tree)**.** *A BT is a three tuple $\mathcal{T}_i = \langle f_i, r_i, \Delta t\rangle$. i is the label of the referring BT, $f_i : 2^n \to 2^n$ is its effect in the system states, Δt is the time step, $r_i : 2^n \to \{\mathcal{S}, \mathcal{R}, \mathcal{F}\}$ is the return status, where $\mathcal{S}$ stands for success, $\mathcal{R}$ stands for running and $\mathcal{F}$ stands for failure. r_i defines a partition of the system states S into three sets, the success region, the running region and the failure region.*

The environment transition is defined by the following equations with a time step to count the frequent tick of BTs:

$$s_{k+1} = f_i(s_k), t_{k+1} = t_k + \Delta t \tag{1}$$

where s is the state of the environment and t denotes the time. To evaluate a BT, the property *finite time successful* is utilized (Colledanchise and Ögren 2017).

Definition 2 (Finite Time Successful)**.** *A BT is finite time successful with region of attraction R', if for any starting states $s_0 \in R'$, there is a finite time τ such that for any $t < \tau$, $r_i(s_t) = \mathcal{R}$ and for any $t \geq \tau$, $r_i(s_t) = \mathcal{S}$.*

We further define the actions of the BT:

Definition 3 (Action)**.** *An action a is a three tuple $\langle pre(a), add(a), del(a)\rangle$, consisting of the precondition, add effects and delete effects of the action.*

The precondition is a set of literals that must be satisfied to perform an action. An action a is applicable in state s_t if $pre(a) \subseteq s_t$. After successfully taking the action, literals in $add(a)$ are added and literals in $del(a)$ are removed from the state. The following property holds for a well defined action:

$$add(a) \cap del(a) = \emptyset \tag{2}$$

$$add(a) \cap pre(a) = \emptyset \tag{3}$$

According to the effects of the action a, for a BT $\mathcal{T}_a$ that executes a from s_t, we have:

$$s_{t+k} = f_a(s_t) = s_t \cup add(a) \backslash del(a) \tag{4}$$

where k denotes the execution time of the action. This paper does not consider the details during the execution of an action and assumes that the action is indivisible. For states where the action a is running, we assume that $pre(a)$ always holds to keep execution, and only when the action succeeds do its effects take place. We also assume that an action always finishes in finite time. Given these preliminary, the behavior planning problem can be formally defined:

Problem 1 (Behavior Planning)**.** *The problem is a tuple $\langle S, A, \mathcal{T}, s_0, g\rangle$, where S is the finite set of environment states, A is the finite set of actions, $\mathcal{T} = \langle f, r, \Delta t\rangle$ is the behavior tree of the robot, s_0 is the initial state, g is the goal condition. The problem is to build a BT $\mathcal{T}_s$ with actions from A as a solution, such that $\mathcal{T}_s$ is finite time successful with region of attraction $R' \ni s_0$, and any s_s in its success region satisfies the goal condition $g \subseteq s_s$.*

One-Step BT Expansion for Condition Satisfaction

To return success in some condition c, a primary BT can be built which only contains a corresponding condition node. If c is satisfied in the current state, the BT immediately returns success. Otherwise the BT returns failure. For such a BT, the success region is exactly the state set satisfying the condition $S_c = \{s \in S | c \subseteq s\}$, and $S \backslash S_c$ is its failure region.

Although the states in the failure region do not satisfy the condition, an active robot may run actions from these states and finally reach the success region. Therefore we can expand the condition node with a behavior tree containing actions to expand more states that can result in success.

The algorithm of the one-step expansion of a BT $\mathcal{T}$ on a condition node c is given in Algorithm 1 and graphically shown in Fig. 4. First, the expansion preserves the condition node c (line 2). Then all the actions which can produce literals in c and do not delete literals in c are selected (line 4). Next the sequence structures for the selected actions are built (lines 5-6), which links each action a with a condition node $c^a_{attr} = pre(a) \cup c \backslash add(a)$. Finally the sequence structures are linked to the top fallback node of the subtree $\mathcal{T}_{sub}$ in line 7 (Note that constructing fallback nodes multiple times is equivalent to linking more nodes to one fallback node, i.e. $FallbackNode(a, FallbackNode(b, c)) = FallbackNode(a, b, c)$). The one-step expansion of the BT holds the following properties.

Proposition 1. *Given a success region defined by condition c, the sequence structure $\mathcal{T}_a$ expanded from action a for condition c is finite time successful with region of attraction $s^a_{attr} = \{s \in S | c^a_{attr} \subseteq s\}$ (cf. lines 5-6 in Algorithm 1).*

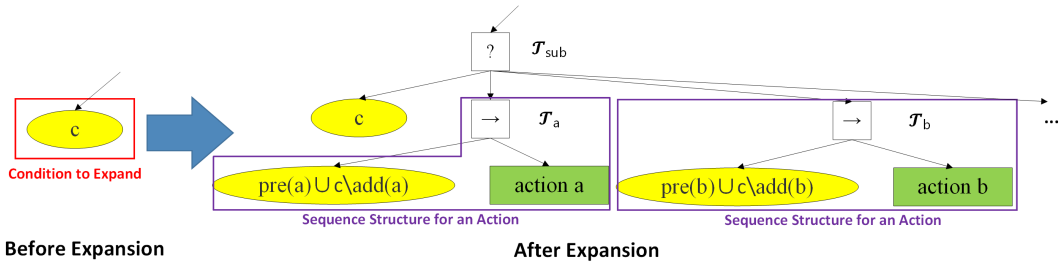


Figure 4: One-Step BT expansion of one condition node.

Algorithm 1: One-step expansion from a condition

1 **Function** `Expand`$(\mathcal{T}, c)$**:**
2 $\quad\mathcal{T}_{sub} \leftarrow c$
3 $\quad$**foreach** $a \in A$ **do**
4 $\quad\quad$**if** $c \cap (pre(a) \cup add(a) \backslash del(a)) \neq \emptyset$ *and* $c \backslash del(a) = c$ **then**
5 $\quad\quad\quad c^a_{attr} \leftarrow pre(a) \cup c \backslash add(a)$
6 $\quad\quad\quad \mathcal{T}_a \leftarrow$ SequenceNode(c^a_{attr}, a)
7 $\quad\quad\quad \mathcal{T}_{sub} \leftarrow$ FallbackNode$(\mathcal{T}_{sub}, \mathcal{T}_a)$
8 $\quad\quad$**end**
9 $\quad$**end**
10 $\quad$Replace c with $\mathcal{T}_{sub}$ in $\mathcal{T}$
11 $\quad$**return** $\mathcal{T}$

Proof. Starting from any $s_t \in s^a_{attr}$, $s_t \supseteq c^a_{attr} = pre(a) \cup c \backslash add(a)$, the sequence structure $\mathcal{T}_a$ starts running action a in this state since $pre(a) \subseteq pre(a) \cup c \backslash add(a) \subseteq s_t$ (cf. eqn. 3). According to eqn. 4, there exists a finite k such that the action returns success and $s_{t+k} = s_t \cup add(a) \backslash del(a)$. With $s_t \supseteq pre(a) \cup c \backslash add(a)$, we can further deduce that:

$$\begin{aligned} s_{t+k} &= s_t \cup add(a) \backslash del(a) \\ &\supseteq (pre(a) \cup c \backslash add(a)) \cup add(a) \backslash del(a) \\ &\supseteq pre(a) \cup c \backslash del(a) \end{aligned} \tag{5}$$

The action selection (line 4 in Algorithm 1) ensures that $c \backslash del(a) = \emptyset$, then from eqn. 5 we have $s_{t+k} \supseteq c$, indicating s_{t+k} satisfies the condition c. Therefore the sequence structure $\mathcal{T}_a$ is finite time successful with region of attraction s^a_{attr} for the success region of condition c. □

Proposition 2. *After the expansion of condition c, the subtree $\mathcal{T}_{sub}$ whose fallback node links all sequence structure of selected actions (cf. line 7 in Algorithm 1 and Fig. 4) is finite time successful for the success region of condition c, and the union $\bigcup s^a_{attr}$ for all selected actions is its region of attraction.*

Proof. For any states in $\bigcup s^a_{attr}$, the corresponding sequence structure is finite time successful of condition c according to Proposition 1. The fallback node of $\mathcal{T}_{sub}$ returns success as long as any child (the sequence structure of an action) returns success and the number of children is finite because the total number of actions are finite. Therefore $\mathcal{T}_{sub}$ is also finite time successful of condition c. □

Proposition 3. *Algorithm 1 terminates in finite time.*

Proof. This is because the total number of actions is finite, so the for-iteration iterates for finite times and each iteration performs constant operations. □

Proposition 2 indicates that the one-step expansion of a condition node c is to generate a subtree that is finite time successful to reach that condition. Before the expansion, the condition node only returns success in states s_c satisfying c and failure otherwise. After the expansion, the subtree returns success immediately in s_c or in finite time from the expanded region of attraction $\bigcup s^a_{attr}$. An interesting point is that each added condition c^a_{attr} may further be expanded.

The BT Expansion Algorithm

The one-step BT expansion provides an intuitive idea to iteratively expand the region of attraction until the initial state s_0 is included, so that the BT can execute from s_0 and succeeds. The expansion starts with a primary BT which only contains a condition node for the goal as in Fig. 5. Ticking the BT from the initial state s_0, the goal condition of the BT can be expanded by a subtree. If the added condition of one sequence structure is satisfied in the initial state, the BT can reach the goal after execution, and if not, the unsatisfied conditions of sequence structures are further expanded.

The pseudo code of the BT expansion algorithm is shown in Algorithm 2. The expansion of the tree performs iteratively until some expanded subtree can run in the initial state (the while-loop in Algorithm 2). Line 5 in Algorithm 2 selects the next condition node to expand with any tree traversal method, e.g. breadth first search, where the conditions already expanded are skipped. Besides, to avoid dead loop, after each expansion those sequence structures whose condition is already expanded are pruned (line 9-10 in Algorithm 2), since it makes no sense to expand a condition with an action sequence that starts from that condition.

Proposition 4. *Algorithm 2 terminates in finite time.*

Proof. The total number of states/conditions is finite, therefore the condition nodes of the BT is also finite with the prune of redundant conditions (line 10). So the while-loop runs for finite times where the algorithm either breaks the loop earlier or returns failure after all conditions are traversed. Inside the loop, the one-step expansion finishes in finite time with Proposition 3. The traversal and the prune finish in finite time due to finite condition nodes of the BT. □

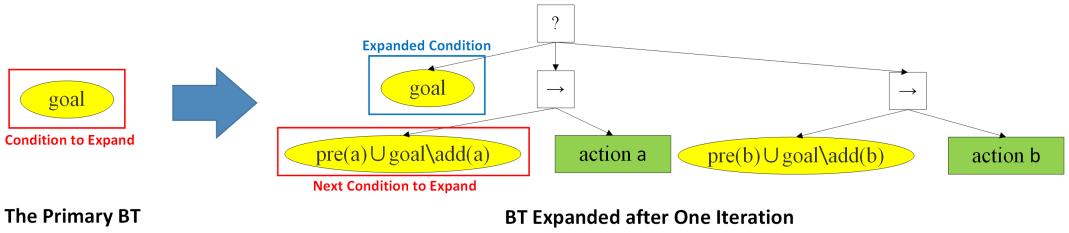


Figure 5: BT Expansion by iteratively expanding unsatisfied condition nodes.

Algorithm 2: BT expansion for behavior planning

```
1  T ← S_g
2  Expanded ← ∅
3  r ← Tick(T)
4  while r = F do
5  |   c ← TraverseToNextCondition(T, Expanded)
6  |   if c = Failure then
7  |   |   return Failure
8  |   end
9  |   T ← Expand(T, c)
10 |   T ← Prune(T, Expanded)
11 |   Expanded ← Expanded ∪ {c}
12 |   r ← Tick(T)
13 end
14 return T
```

We analyze the soundness and completeness of Algorithm 2 as follows.

Lemma 1. *In the expansion loop of Algorithm 2, any added condition node in finite steps is in the region of attraction of the BT with the goal condition as success region.*

Proof. This lemma can be proved by strong induction. In the basis step (the first expansion), the only condition node to expand in the primary BT is the goal condition. According to Proposition 2, Lemma 1 holds for the basis step.

For the inductive step, consider an expansion of an existing condition node c for which Lemma 1 holds as the inductive premise. We need to prove that it also holds in the added condition nodes of this expansion. According to Proposition 2, the BT will reach states under c in finite time if executing from the newly added sequence structure. With the inductive premise that c is in the region of attraction of the goal condition, the agent can then reach the goal condition in finite time. According to the above induction, any added condition node in finite steps is in the region of attraction of the BT with the goal condition as success region. □

Proposition 5. *Algorithm 2 is sound, i.e. if it returns a result $\mathcal{T}$ other than $Failure$, $\mathcal{T}$ is a solution to Problem 1.*

Proof. If a BT $\mathcal{T}$ is returned, the algorithm breaks the while-loop in finite steps with a BT tick resulting other than failure. It indicates that some condition node holds in the initial state. According to Lemma 1, the initial state is in the region of attraction of the BT with the goal condition as its success region. Therefore the returned BT solves Problem 1. □

Proposition 6. *Algorithm 2 is complete, i.e. if Problem 1 is solvable, the algorithm returns a BT $\mathcal{T}$ which is a solution.*

Proof. This proposition can be proved by contradiction. We assume that there exists a BT that is the solution of Problem 1 but our algorithm fails, i.e. it cannot break the while-loop after expanding all condition nodes in the building BT. The solution is finite time successful so we can utilize a finite sequence to represent the state transition of the solution BT: $(s_0, a_1, s_1, a_2, ..., s_{n-1}, a_n, s_n)$ where the goal condition $g \subseteq s_n$. We first prove that any state in this sequence must satisfy some condition node traversed in the expansion while-loop by mathmatical induction from the right to the left of this sequence.

For the basis step, we consider the state s_{n-k} with $k = 0$, i.e. $s_n \supseteq g$. Obviously g is a condition node of the BT from the beginning primary BT.

For the inductive step, we consider a transition $(s_{n-k-1}, a_{n-k}, s_{n-k})$ and there exists a condition node $c \subseteq s_{n-k}$ in the BT as the inductive premise. We need to prove that there also exists a condition node satisfied by s_{n-k-1}. The inductive premise and the transition gives:

$$s_{n-k-1} \cup add(a_{n-k}) \backslash del(a_{n-k}) = s_{n-k} \supseteq c \tag{6}$$

which can be deduced to:

$$s_{n-k-1} \supseteq c \backslash add(a_{n-k}) \tag{7}$$

Without loss of generality, we analyze in two cases, whether $c \cap (pre(a_{n-k}) \cup add(a_{n-k}) \backslash del(a_{n-k})) = \emptyset$.

(1) $c \cap (pre(a_{n-k}) \cup add(a_{n-k}) \backslash del(a_{n-k})) = \emptyset$: From the case premise, $c \cap add(a_{n-k}) = \emptyset$ holds because the add effects and delete effects has no common member (cf. eqn. 2). Therefore eqn. 7 can further be deduced to:

$$s_{n-k-1} \supseteq c \tag{8}$$

Therefore s_{n-k-1} satisfies the condition node c of the BT.

(2) $c \cap (pre(a_{n-k}) \cup add(a_{n-k}) \backslash del(a_{n-k})) \neq \emptyset$: The transition shown by eqn. 6 indicates $s_{n-k} \backslash del(a_{n-k}) = s_{n-k}$. Therefore when expanding c, action a_{n-k} will be selected because the selective condition holds (cf. line 4 in Algorithm 1), which creates a condition node $c_{attr}^{a_{n-k}} = pre(a_{n-k}) \cup c \backslash add(a_{n-k})$. The state transition also provides that $s_{n-k-1} \supseteq pre(a_{n-k})$, then with eqn. 7 we can deduce:

$$s_{n-k-1} \supseteq pre(a) \cup c \backslash add(a_{n-k}) \tag{9}$$

Therefore s_{n-k-1} satisfies the condition node c_{attr} added to the BT. The last step is to consider whether the condition node c_{attr} will be pruned. If it is not pruned, c_{attr} is exactly the required condition node. If it is pruned, the prune indicates that there already exists some condition node $c_{exist} \subseteq c_{attr} \subseteq s_{n-k-1}$ in the BT, c_{exist} is the condition node satisfied by s_{n-k-1}.

The two cases complete the proof of the inductive step. The mathmatical induction proves that any state in the sequence must satisfy some condition node generated in the expansion while-loop. Therefore s_0 will satisfy some condition node during the BT expansion loop, then the tick will return $\mathcal{R}$ to break the while-loop and return a BT, which contradicts the initial assumption that a solution BT exists but our algorithm fails. With the soundness of the algorithm, the returned BT is a solution. Therefore if Problem 1 is solvable, the algorithm returns a BT $\mathcal{T}$ which is a solution. □

Evaluation

Formal Discussion of Behavior Planning with BTs

Despite the advantages of automated planning like assisting human design and ensuring correctness, the behavior planning with BTs has more advantages in terms of *blended planning and acting*. The pioneering work (Colledanchise, Almeida, and Ögren 2019) states the advantages in four points, which we can formally discuss in our formulation.

(1) *if the action executed by the robot is reversed, the robot takes the action again without the need to replan*. (2) *If an external agent accomplishes an action the robot plans to do, the robot skips the action without the need to replan.*

Consider a solution BT whose action sequence is $(s_0, a_1, s_1, a_2, ..., s_n)$. Reversing any action means a back step to a previous state and external help of any action means a forward step to a subsequent state. Since all states are in the success region of the BT, there is no need to replan and the BT is finite time successful from the disturbed states.

(3) *If there are several actions reaching the same post condition, the BT should include them so that if one fails the others can be tried.*

As in Fig. 4, when expanding some condition, our algorithm utilizes multiple actions to build sequence structures linking to the top fallback node. Intuitively any one fails the others can be tried. From the view of region of attraction, this is because the region expanded through different actions may have intersecting states, in which if one action fails, the other actions can be tried.

(4) *The BT should be able to be expanded during runtime.*

Our method comes to an even stronger property. Assuming that the environment suddenly changes from s_a to s_b for any reason, as long as s_b is in the region of attraction of the BT, there is no need to expand. If s_b is outside the region of attraction, the BT can continue the BT expansion algorithm to further expand its region of attraction until s_b is included. The soundness and completeness guarantees that if a solution exists, the BT will always expands to a valid one.

Robustness. The BT is more robust to external disturbances with larger region of attraction. Algorithm 2 can be modified to tradeoff between the robustness and efficiency.

Actions	Pre	Add	Del
Move(b,a_b)	Free(a_b), WayClear	At(b,a_b)	Free(a_b), At(b,p_b)
Move(s,a_b)	Free(a_b)	At(s,a_b), WayClear	Free(a_b), At(s,p_s)
Move(s,a_s)	Free(a_s)	At(s,a_s), WayClear	Free(a_s), At(s,p_s)

Table 1: Actions in the mobile manipulator simulation.

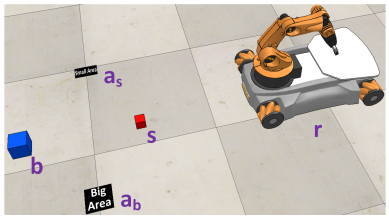


Figure 6: The simulation scenario.

In the most robust case, the algorithm breaks the loop after all condition nodes are expanded, regardless of the satisfaction in s_0. The resulting BT's region of attraction can be pretty large, as well as the size of the tree. It includes all solvable states according to the completeness of the algorithm. In the most efficient case, the algorithm prunes all nodes that are redundant and degenerates into classical planning sequence $(s_0, a_1, s_1, a_2, ..., s_n)$. The BT's region of attraction is small but valid, as well as the size of the tree.

Complexity. The time complexity of BT expansion is polynomial to the system size $(|A| + |S|)$ for state space traversal. It may vary from $O(b|S|)$ to $O(|A||S|log|S|)$ with problem and implementation settings, where b is the max-branching factor. Details are in the supplementary material[1].

Experiments

We first demonstrate the superior of our method to the baseline (Colledanchise, Almeida, and Ögren 2019) in a case study simulated with a mobile manipulator. Then we evaluate our method with numerical computation tests.

Simulation The simulation is in the CoppeliaSim simulator (cf. Fig. 6) where there are a Youbot mobile manipulator r, a big cargo b, a small cargo s, and two free appointed areas a_b and a_s. The big cargo is only allowed to place in the big area a_b and the small cargo can be placed in both areas.

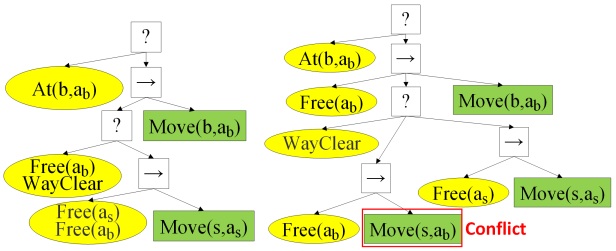
The goal for the robot is to move the cargo b from the initial position p_b to the appointed area a_b. However the cargo s at p_s stands in its way, so the robot has to first move s to somewhere and then moves b. The actions are listed in Table. 1, where we define three actions for the case study.

The solution BT of our algorithm is shown in Fig. 7a, by which the robot first moves the blocking cargo s to the area a_s, then moves the target cargo b to the area a_b and accomplishes the goal. The advantages discussed in the previous

[1]https://github.com/HPCL-micros/bt-expansion

Case	Test Set Generation			Problem		Tree Size (BT expansion)		Tree Size (baseline)	
	Literals	Distance	Iterations	States	Actions	Avg.	Std.	Avg.	Std.
0	10	10	10	20.6	20	**35.3**	29.7	85.7	**19.3**
1	10	10	100	103.9	110	**80.6**	**64.6**	457.7	95.1
2	10	10	1000	607.5	1010	**395.6**	**265.4**	4135.0	908.7
3	100	10	10	21	20	**41.0**	**0.1**	605.2	21.5
4	100	10	1000	1011	1010	**41.5**	**1.5**	30840.8	192.8
5	10	50	10	58.8	60	**62.7**	**54.6**	247.1	59.3
6	10	50	100	138.1	150	**99.7**	**79.0**	616.9	146.6
7	10	50	1000	621.0	1050	**430.0**	**290.8**	4248.7	977.4
8	100	50	10	61	60	**201.2**	**0.9**	1829.8	36.2
9	100	50	1000	1051	1050	**203.9**	**7.1**	32074.7	184.4

Table 2: Evaluation of efficiency in terms of tree size with different test sets.



(a) Right Solution (b) Wrong Solution

Figure 7: Results of (a) our approach and (b) the baseline.

subsection hold, e.g if someone else moves back the blocking cargo after the robot clears its way, the robot will move it again to clear the way, and if someone moves away the blocking cargo, the robot directly moves the target cargo, without the need to replan.

Although the solution is obvious in such a simple case, the baseline approach (Colledanchise, Almeida, and Ögren 2019) may fail and expand a wrong BT in Fig. 7b. The BT moves s to a_b to clear the way, but also occupies the only area for cargo b, breaking the precondition Free(a_b) to move b. Although the authors intuitively proposed a conflict resolution method, this error cannot be corrected since an action selection/deletion mechanism is missed in their method, but the wrong action in this case should never be selected.

The case study illustrates the baseline's defects in a simple way for clarity, and other problems such as an infinite loop may also make the baseline fail. Basically this is because the algorithm is not sound and complete, it may return a BT that is not a solution and fails even if a solution exists. On the contrast, our algorithm is proven to be sound and complete, so it is superior due to its complete theoretical basis. Besides, our method tends to achieve better efficiency, in terms of the size of the generated BT. In the case study, even if the baseline's conflicting subtree is pruned, its number of nodes is still more than ours.

Computation This section we empirically evaluate the efficiency through computation. A vast number of tasks with different number of literals, states, actions, and solution distance are randomly generated, each case for 1000 tests, as listed in Table. 2. Test sets are generated by firstly generating a path from the start to the goal with given distance and then randomly expanding states by iterations to produce a mission graph. More details of the test set can also be found in the aforementioned supplementary materials.

We mainly compare the generated BT size as the indicator of efficiency, where less tree size means not only efficiency for the expansion of BT generation, but also the tick of BT execution, i.e. the execution overhead. Since our algorithm utilizes compact condition nodes, i.e. one condition node may check multiple literals, the BT size does not necessarily increase with the number of literals, while the baseline (Colledanchise, Almeida, and Ögren 2019) generates significantly large BTs with more literals (cases 0/2 vs 3/4, 5/7 vs 8/9). The tree size increases with the distance from the start to the goal (cases 0/1/2/3/4 vs 5/6/7/8/9). It may not increase much with the total states number in the test set, but with the actions per state. If one state contains exactly one action (except the end), the solution BT is rather small (cases 3/4/8/9).

Summing up all the experimental results, BT expansion is superior in effectiveness and efficiency. With the solid theoretical basis, it always generates a solution BT as long as a solution exists, providing reliability in robot applications. And the empirical results show that it tends to generate BTs that are much more efficient in terms of the tree size.

Conclusion

This paper proposes BT expansion, a sound and complete algorithm for behavior planning of intelligent robots. BTs are automatically synthesized with a complete theoretical basis through the combination of the STRIPS-style planning and the state-space formulation of BTs. Robots with BT expansion are guaranteed to work under any resolvable disturbances. Experiments are conducted with a simulated manipulator and a test set. The results demonstrate the effectiveness and efficiency of BT expansion, and its superiority to the baseline approach. Future work is to further improve the generated BT size and study planning under uncertainty. Another research direction is to improve the mathematical evaluation of BTs other than finite time successful property, such as temporal logic, so as to facilitate periodic tasks.

Acknowledgements

This work was partially supported by the National Key Research and Development Program of China (No. 2017YFB1001900), the Youth Talent Lifting Project (No. 17-JCJQ-QT-047), the National Natural Science Foundation of China (No. 91948303, 91648204, 61803375, 62032024 and 12002380), and the National Science and Technology Major Project.

References

Banerjee, B. 2018. Autonomous Acquisition of Behavior Trees for Robot Control. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain, October 1-5, 2018*, 3460–3467. doi:10.1109/IROS.2018.8594083.

Brooks, R. A. 1986. A robust layered control system for a mobile robot. *IEEE J. Robotics Autom.* 2(1): 14–23. doi:10.1109/JRA.1986.1087032.

Colledanchise, M.; Almeida, D.; and 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*, 8839–8845. doi:10.1109/ICRA.2019.8794128.

Colledanchise, M.; and Ögren, P. 2017. How Behavior Trees Modularize Hybrid Control Systems and Generalize Sequential Behavior Compositions, the Subsumption Architecture, and Decision Trees. *IEEE Trans. Robotics* 33(2): 372–389. doi:10.1109/TRO.2016.2633567.

Colledanchise, M.; and Ögren, P. 2018. *Behavior trees in robotics and AI: An introduction*. CRC Press.

de Pontes Pereira, R.; and Engel, P. M. 2015. A Framework for Constrained and Adaptive Behavior-Based Agents. *ArXiv* abs/1506.02312.

Fikes, R.; and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artif. Intell.* 2(3/4): 189–208. doi:10.1016/0004-3702(71)90010-5.

French, K.; Wu, S.; Pan, T.; Zhou, Z.; and Jenkins, O. C. 2019. Learning Behavior Trees From Demonstration. In *International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019*, 7791–7797. doi:10.1109/ICRA.2019.8794104.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2016. *Automated Planning and Acting*. Cambridge University Press. ISBN 978-1-107-03727-4.

Guo, M.; Tumova, J.; and Dimarogonas, D. V. 2016. Communication-Free Multi-Agent Control Under Local Temporal Tasks and Relative-Distance Constraints. *IEEE Trans. Autom. Control.* 61(12): 3948–3962. doi:10.1109/TAC.2016.2527731.

Iovino, M.; Scukins, E.; Styrud, J.; Ögren, P.; and Smith, C. 2020. A Survey of Behavior Trees in Robotics and AI. *ArXiv* abs/2005.05842.

Lan, M.; Xu, Y.; Lai, S.; and Chen, B. M. 2018. A modular mission management system for micro aerial vehicles. In *14th IEEE International Conference on Control and Automation, ICCA 2018, Anchorage, AK, USA, June 12-15, 2018*, 293–299. doi:10.1109/ICCA.2018.8444251.

Lim, C.; Baumgarten, R.; and Colton, S. 2010. Evolving Behaviour Trees for the Commercial Game DEFCON. In *Applications of Evolutionary Computation, EvoApplicatons 2010: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC, Istanbul, Turkey, April 7-9, 2010, Proceedings, Part I*, 100–110. doi:10.1007/978-3-642-12239-2_11.

Millington, I.; and Funge, J. 2009. *Artificial intelligence for games*. CRC Press.

Neupane, A.; and Goodrich, M. A. 2019. Learning Swarm Behaviors using Grammatical Evolution and Behavior Trees. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, 513–520. doi:10.24963/ijcai.2019/73.

Nilsson, N. J. 1994. Teleo-Reactive Programs for Agent Control. *J. Artif. Intell. Res.* 1: 139–158. doi:10.1613/jair.30.

Rovida, F.; Großmann, B.; and Krüger, V. 2017. Extended behavior trees for quick definition of flexible robotic tasks. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017*, 6793–6800. doi:10.1109/IROS.2017.8206598.

Rovida, F.; Wuthier, D.; Großmann, B.; Fumagalli, M.; and Krüger, V. 2018. Motion Generators Combined with Behavior Trees: A Novel Approach to Skill Modelling. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain, October 1-5, 2018*, 5964–5971. doi:10.1109/IROS.2018.8594319.