2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

Performance Analysis of Stochastic Behavior Trees

Michele Colledanchise, Alejandro Marzinotto and Petter Ögren

***Abstract*— This paper presents a mathematical framework for performance analysis of Behavior Trees (BTs). BTs are a recent alternative to Finite State Machines (FSMs), for doing modular task switching in robot control architectures. By encoding the switching logic in a tree structure, instead of distributing it in the states of a FSM, modularity and reusability are improved.**

In this paper, we compute performance measures, such as success/failure probabilities and execution times, for plans encoded and executed by BTs. To do this, we first introduce Stochastic Behavior Trees (SBT), where we assume that the probabilistic performance measures of the basic action controllers are given. We then show how Discrete Time Markov Chains (DTMC) can be used to aggregate these measures from one level of the tree to the next. The recursive structure of the tree then enables us to step by step propagate such estimates from the leaves (basic action controllers) to the root (complete task execution). Finally, we verify our analytical results using massive Monte Carlo simulations, and provide an illustrative example of the results for a complex robotic task.

I. Introduction

Behavior Trees (BTs) were developed within the computer gaming industry [1], [2] as a more modular and flexible alternative to Finite State Machines (FSMs). Their recursive structure and usability have made them very popular in industry, which in turn has created a growing amount of attention in academia [3]–[9]. The main advantage of BTs as compared to FSMs can be seen by the following programming language analogy. In FSMs, the state transitions are encoded in the states themselves, and switching from one state to the other leaves no memory of where the transition was made from. This is very general and flexible, but actually very similar to the now obsolete *GOTO statement*, that was an important part of many early programming languages, e.g., BASIC. In BTs the equivalents of state transitions are governed by function calls and return values being passed up and down the tree structure. This is also flexible, but more similar to the calls of FUNCTIONS that has replaced GOTO in almost all modern programming languages. Thus, BTs exhibit many of the advantages in terms of readability, modularity and reusability that was gained when going from GOTO to FUNCTION calls in the 1980s.

BTs were first described in [1], [2], [10], as powerful tools to provide intelligent behaviors for non-player characters in high profile computer games, such as the HALO series.

The authors are with the Computer Vision and Active Perception Lab., Centre for Autonomous Systems, School of Computer Science and Communication, Royal Institute of Technology (KTH), SE-100 44 Stockholm, Sweden. e-mail: {miccol|almc|petter}@kth.se
This work has been supported by the Swedish Research Council and the European Union FP7 project Reconfig (FP7-ICT-600825). The authors gratefully acknowledge the support.

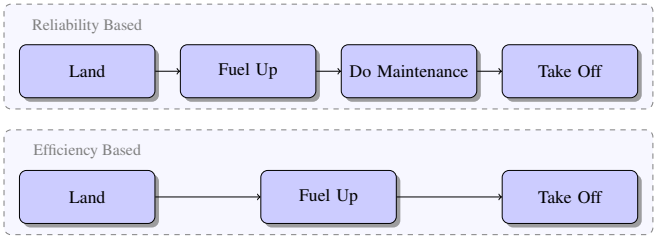


Fig. 1. Difference between a plan that optimizes expected success rate (top) and one that optimizes expected time to completion (bottom).

Later work proposed ways of combining BTs with machine learning techniques [3], [4], and making them more flexible in terms of parameter passing [5]. The advantage of BTs as compared to FSMs was also the reason for extending the JADE agent Behavior Model with BTs in [6], and the benefits of using BTs to control complex multi mission Unmanned Aircraft Vehicle (UAVs) was described in [7]. The modularity and structure of BTs enabled a step towards the formal verification of mission plans in [9]. Finally, in [8], BTs were used to perform autonomous robotic grasping. In particular, it was shown how BTs enabled the easy composition of primitive actions into complex and robust manipulation programs.

In this paper, we estimate performance measures such as success/failure probabilities and execution times, for plans that are encoded and executed using BTs. Imagine we have an airborne micro UAV that needs to land, refuel and then take off again to do some important surveillance. While landed, some sensor maintenance can be done to improve the sensor performance, but it can also be skipped, to reduce the time on ground. Thus, the two options are the trivial sequential plans depicted in Fig. 1, one maximizing success rate, and the other minimizing time to completion. Assuming we are given the performance of the individual actions, e.g., *Do Maintenance*, we show how to compute overall estimates for arbitrary BT compositions, including both sequences such as the one in Fig. 1 and so-called Selectors providing *fall back functionality* such as the one in Fig. 2 below.

The analysis is done by defining Stochastic Behavior Trees (SBTs) and describing the interaction of a BT node with its children in terms of a Discrete Time Markov Chain (DTMC). This enables us to propagate performance estimates from one level in the tree to the next. Applying the scheme in a recursive fashion then makes it possible to compute the properties of an arbitrarily complex BT composition of actions.

The contribution of this paper is that we provide estimates of how reliable (in terms of success probability) and efficient

(in terms of time to success) a BT-composition of primitive actions is. To the best of our knowledge, such a probabilistic analysis of BTs has not been done before.

The outline of this paper is as follows. In Section II we review details on BTs and DTMCs. Then, in Section III we state the main problem of this paper. A theoretic analysis of the problem is given in Section IV and a numerical verification of the results is presented in Section V, together with an illustrative example. Finally, the paper is concluded in Section VI.

II. Background: BTs and DTMCs

In this section we give a brief review of BTs and DTMCs. The later is used to analyze the performance of the former.

A. Behavior Trees

Here, we give the reader a brief overview of BTs, while a more detailed description can be found in [7].

A BT is a directed rooted tree in which the nodes are classified as root, control flow nodes, or execution nodes. For each pair of connected nodes we call the outgoing node *parent* and the incoming node *child*. The root has no parents and exactly one child, the control flow nodes have one parent and at least one child, and the execution nodes have one parent and no children. Graphically, the children of a control flow node are placed below it, ordered from its left to its right, as shown in Figures 2-4.

The execution of a BT starts from the root which sends *ticks* [1] with a certain frequency to its child. When a given node in the BT receives a tick from its parent, its execution is allowed and it returns to the parent a status *running* if its execution has not finished yet, *success* if it has achieved its goal, or *failure* otherwise.

There are four types of control flow nodes (selector, sequence, parallel, and decorator) and two execution nodes (action and condition). Their execution is explained below.

Selector: When the selector node receives a tick from its parent, it ticks its children in succession, returning success (running) as soon as it finds a child that returns success (running). It returns failure only if all the children return failure. The purpose of the selector node is to robustly carry out a task that can be performed using several different approaches (e.g. an object manipulation that can be performed using either a single-hand or a multiple-hand approach) by trying each of them until one that succeeds is found. The selector node is graphically represented by a box with a "?", as in Fig. 2.

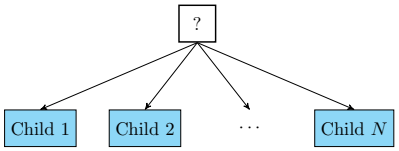


Fig. 2. Graphical representation of a selector node with N children.

[1] A tick is a signal that enables the execution of a child

Sequence: When the sequence node receives a tick from its parent it ticks its children in succession, returning failure (running) as soon as if finds a child that returns failure (running). It returns success only if all the children return success. The purpose of the sequence node is to carry out the tasks that are defined by a strict sequence of sub-tasks, in which all have to succeed to achieve a desired goal (e.g. a mobile robot that has to collect an object, move to a desired position and then unload the collected object). The sequence node is graphically represented by a box with a "→", as in Fig. 3.

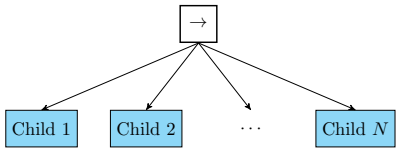


Fig. 3. Graphical representation of a sequence node with N children.

Parallel: When the parallel node receives a tick from its parent, it ticks its children in parallel and returns success if a given number of children return success, it returns failure if the remaining running children are not enough to reach the given number, even if they are all going to succeed, and it returns running otherwise. The purpose of the parallel node is to model those tasks separable in dependent sub-tasks performing similar actions (e.g. a fault diagnosis system that has to monitor in parallel several hardware). The parallel node is graphically represented by a box with two "→", as in Fig. 4.

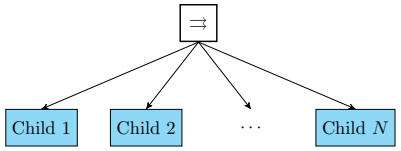


Fig. 4. Graphical representation of a parallel node with N children.

Decorator: The decorator is a special node that has only one child and it changes its child outcome according to an arbitrarily policy defined by the user (e.g. it returns failure if the child returns running after a given time). The decorator is graphically represented in Fig. 5(a).

Action: When the action node receives a tick from its parent it returns success if the action is completed and failure if the action cannot be completed. Otherwise it returns running. The action node is represented in Fig. 5(b)

Condition: The condition node verifies if a condition is satisfied or not, returning success or failure accordingly. The

(a) Decorator node. The label describes the user defined policy.

(b) Action node. The label describes the action performed

(c) Condition node. The label describes the condition verified

Fig. 5. Graphical representation of a decorator, action, and condition node.

condition node never returns running. The condition node is represented in Fig. 5(c)

Root: The root node is the node that generates ticks. It is graphically represented as a white box labeled with "$\varnothing$"

B. Discrete Time Markov Chains

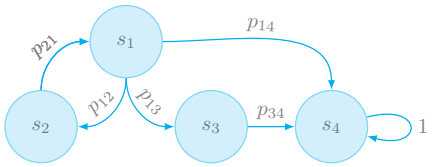


Fig. 6. Example of a Markov chain with 4 states and 6 transitions.

Markov theory [11] deals with memory-less processes. If a plan is given by a sequence of actions that changes the system's state disregarding its history, a Discrete Time Markov Chain (DTMC) is suitable to model a plan execution. A DTMC is given by a collection of states $\mathcal{S} = \{s_1, s_2, \ldots, s_d\}$. The process modeled by the DTMC moves from a state s_i to a state s_j at time step k with the probability $p_{ij}(k)$.

Definition 1: The *one-step transition matrix* $P(k)$ is a $|\mathcal{S}| \times |\mathcal{S}|$ matrix in which the entries are the transition probabilities $p_{ij}(k)$.
Let $\pi(k) = (\pi_1(k), \ldots, \pi_{|\mathcal{S}|}(k))$, where $\pi_i(k)$ is the probability of being in state i at time k, then the Markov process can be described as a discrete time system with the following time evolution:

$$\pi(k+1) = P^\top(k)\pi(k) \tag{1}$$

Definition 2: The *mean sojourn time* $\mathcal{T}_i$ is the mean time spent by the DTMC in a given state s_i.
If a state s_i in a MC has no outgoing arcs to any other state (i.e. with $p_{ii} = 1$) it is called *absorbing*, otherwise it is called *transient*. The sojourn time is computed for transient states only, the time spent in a absorbing state is trivially infinity. In Fig. 6 the state s_4 is an absorbing state.

Definition 3: The *mean time to absorbtion* (MTTA) is the mean time needed to reach any absorbing state starting from the initial state.
To compute a closed form expression of the MTTA, we reorder the states such that the initial state comes first, and all the absorbing states come last. Let $\mathcal{S}_\mathcal{A} \in \mathcal{S}$ be the set of absorbing states

$$P^\top(k) = \begin{bmatrix} T(k) & \mathbb{0} \\ R(k) & \mathbb{I} \end{bmatrix} \tag{2}$$

where T is a $(|\mathcal{S}|-|\mathcal{S}_\mathcal{A}|)\times(|\mathcal{S}|-|\mathcal{S}_\mathcal{A}|)$ matrix describing the one-step transition from a transient state to another, R is a $|\mathcal{S}_\mathcal{A}| \times (|\mathcal{S}|-|\mathcal{S}_\mathcal{A}|)$ matrix describing the one-step transition from a transient state to an absorbing state, the matrices $\mathbb{0}$ and $\mathbb{I}$ express the characteristic of the absorbing states, in particular they have no transitions to any other state. The MTTA is the computed as

$$MTTA = \sum_i g_{1,i}\mathcal{T}_i \tag{3}$$

where $\mathcal{T}_i$ is the mean sojourn time and $g_{1,i}$ is the i-th entry of the first column of the matrix G defined as follows:

$$G = \prod_{k=1}^{\infty} T(k), \tag{4}$$

where $T(k)$ is given by Equation (2). We consider only the first column since our calculation concerns the initial state. The series (4) converges according to the main theorem of [12] since $p_{ij}(k) \in [0,1] \forall i,j,k$.

Infinitesimal generators matrix: The infinitesimal generator matrix Q describes the continuous time behavior of a Markov process, its entries are defined as follows:

$$q_{i,j} = \begin{cases} \frac{1}{\mathcal{T}_j} p_{j,i} & \text{if } i \neq j \\ -\sum_{k \neq i} q_{i,k} & \text{otherwise} \end{cases} \tag{5}$$

The continuous time behavior of the Markov process is described by the following ordinary differential equation, known as the Cauchy problem:

$$\begin{cases} \dot{\pi}(t) = Q(t)\pi(t) \\ \pi(0) = \pi_0 \end{cases} \tag{6}$$

where the initial probability vector π_0 is assumed to be known a priori.

III. Problem Formulation

In this section, we first make some definitions and assumptions, then state the main problem of this paper, and finally illustrate it with an example. We begin with a formal definition of the BTs, first introduced in Section II.

Definition 4: A BT is a rooted tree defined as a two-tuple, BT $= (\mathcal{V}, \mathcal{E})$ where:

- $\mathcal{V} = \mathcal{A} \cup \mathcal{C} \cup \mathcal{N} \cup \tau$ a finite set of nodes, composed by execution nodes (actions $\mathcal{A}$ and conditions $\mathcal{C}$) control flow nodes $\mathcal{N}$ (e.g. selectors or sequences) and a root τ. The nodes are numbered, $\mathcal{V} \subset \mathbb{N}$.
- $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ is a finite set of edges.

As noted above, we are interested in describing the execution properties of a BT in terms of succees and fail probabilities, and time to success and failure respectively. In order to facilitate such an analysis, we make the following assumptions.

Assumption 1: For each action $\mathcal{A}$ in the BT, the following holds:

- It first returns running, for an amount of time that might be zero or non-zero, then consistently returns either success or failure for the rest of the execution of its parent node[2].
- The probability to succeed at any given time $p_s(t)$ and the probability to fail at any given time $p_f(t)$ are known a priori.
- The time to *succeed* and the time to *fail* are random variables with exponential distribution with rate μ and ν known a priori.

[2]The execution of the parent node starts when it receives a tick and finishes when it returns either success/failure to its parent.

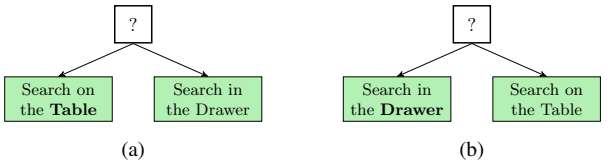


Fig. 7. BT modeling of two plan options. In (a), the robot searches on the table first, and in the drawer only if the table search fails. In (b), the table is searched only if nothing is found in the drawer.

Assumption 2: For each condition $\mathcal{C}$ in the BT, the following holds

- It consistently returns the same value (success or failure) throughout the execution of its parent node.
- The probability to succeed at any given time $p_s(t)$ and the probability to fail at any given time $p_f(t)$ are known a priori.

We are now ready to define a SBT.

Definition 5: A Stochastic Behavior Tree (SBT) is a BT satisfying Assumptions 1 and 2.

Given a SBT, we want to use the probabilistic descriptions of its actions and conditions, $p_s(t)$, $p_f(t)$, μ and ν, to recursively compute analogous descriptions for every sub-trees and finally the whole tree. Formally the problem is as follows:

Problem 1: Given a SBT as defined above. Compute the probabilistic measures $p_s(t)$, $p_f(t)$, μ and ν for each sub-tree, and ultimately the root of the SBT.

To illustrate the problem and SBTs we take a look at the following example.

Example 1: Imagine a robot that is to search for a set of keys on a table and in a drawer. The robot knows that the keys are often located in the drawer, so that location is more likely than the table. However, searching the table takes less time, since the drawer must be opened first. Two possible plans are conceivable: searching the table first, and then the drawer, as in Fig. 7(a), or the other way around as in Fig. 7(b). These two plans can be formulated as SBTs and analyzed through the scope of Problem 1, using the results of Section IV below. Depending on the user requirements in terms of available time or desired reliability at a given time, etc. the proper plan/SBT can be chosen.

Remark 1: Note that Assumption 1 corresponds to the return status of the search actions behaving in a reasonable way, e.g., not switching between success and failure.

To address Problem 1 for SBTs that are more complex than the one in Example 1, we need to preform the following analysis.

IV. Proposed Approach

In this section we first transform Problem 1 into a DTMC problem (see Section II above) and then compute the desired probabilistic measures of it using the proposed approach.

A. Transforming a SBT into a DTMC

The first step of our approach is to define, for each control flow node in $\mathcal{V}$, a vector representation of the children's outcomes and a description of its execution policy, then we map the execution into a DTMC with a direct representation of the one-step transition matrix, and finally we compute the probability of success or failure over time for each node and its success/failure rates.

The modularity of BTs comes from the recursive tree structure, any BT can be inserted as sub-tree (child of a flow control node) in another BT. This modularity allows us to do the analysis in a recursive fashion, beginning first with those sub-trees that have only execution nodes as children (i.e. only actions and conditions as children, which have known probabilistic parameters given Assumptions 1 and 2) and then using the newly derived parameters to recursively calculate the parameters of their parents node, all the way to the root.

The children outcomes of a given flow control node are collected in a vector state called the *marking* of the node, and the transitions between markings are defined according to the execution policy of the node. Let $\mathbf{m}(k) = [m_1(k), m_2(k), \dots, m_N(k)]$ be a marking of a given BT node with N children at time step k with

$$m_i = \begin{cases} -1 & \text{if child } i \text{ has failed} \\ 1 & \text{if child } i \text{ has succeded} \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

We define an *event* related to a BT node when one of its children returns either success or failure. Defining $\mathbf{e}_i(k)$ the vector associated to the event of the i-th running child, all zeros except the i-th entry which is equal to $e_i \in \{-1, 1\}$:

$$e_i = \begin{cases} -1 & \text{if child } i \text{ returs failure} \\ 1 & \text{if child } i \text{ returns success.} \end{cases} \tag{8}$$

An event is *feasible* only if the related children has been ticked (e.g. in a sequence node, a child i can not be ticked if the previous one has failed, in which case there is a feasible event associated to the child i), hence a transition between two markings exists if there is an associated feasible event. Let $\mathcal{F}(\mathbf{m}(k))$ be the set of feasible events at marking $\mathbf{m}(k)$, the marking sequence is given by $\mathbf{m}(k+1) = \mathbf{m}(k) + \mathbf{e}_i(k)$ with $\mathbf{e}_i \in \mathcal{F}(\mathbf{m}(k))$.

The reachability graph (RG) of a BT node can now be computed starting from the initial marking $\mathbf{m}(0) = \mathbf{0}$, taking into account all the possible event combinations that satisfy the feasibility condition.

Feasibility condition in the Selector node: An event $\mathbf{e}_i(k)$ in a selector node is feasible if the corresponding child has not returned success or failure and the node has not returned success/failure yet:

$$\begin{aligned} \mathcal{F}(\mathbf{m}(k)) = \{&\mathbf{e}_1(k) : m_1(k) = 0\ , \\ &\mathbf{e}_i(k) : m_i(k) = 0 \wedge m_{i-1}(k) = -1\} \end{aligned} \tag{9}$$

Feasibility condition in the Sequence node: An event $\mathbf{e}_i(k)$ in a sequence node is feasible if the corresponding child has not returned success or failure and the node has

not returned success/failure yet:

$$\mathcal{F}(\mathbf{m}(k)) = \{\mathbf{e}_1(k) : m_1(k) = 0\ , \\ \mathbf{e}_i(k) : m_i(k) = 0 \wedge m_{i-1}(k) = 1\} \tag{10}$$

Feasibility condition in the parallel node: An event $\mathbf{e}_i(k)$ in a parallel node with N children, in which M of them have to succeed, is feasible if the corresponding child has not returned success or failure, if less than M children have succeeded and if less than $N - M + 1$ have failed:

$$\mathcal{F}(\mathbf{m}(k)) = \{\mathbf{e}_i(k) : m_i(k) = 0 \ \wedge \sum_{j:m_j(k)>0} j < M \\ \wedge \sum_{j:m_j(k)<0} j < N - M + 1\} \tag{11}$$

Definition 6: A marking $\mathbf{m}_i$ is *reachable* from a marking $\mathbf{m}_j$ if there exists a sequence of feasible events $\boldsymbol{\sigma} = [\sigma_1, \sigma_2, \ldots, \sigma_g]$ such that $\mathbf{m}_i(k+g) = \mathbf{m}_j(k) + \sum_{h=1}^{g} \sigma_h$.

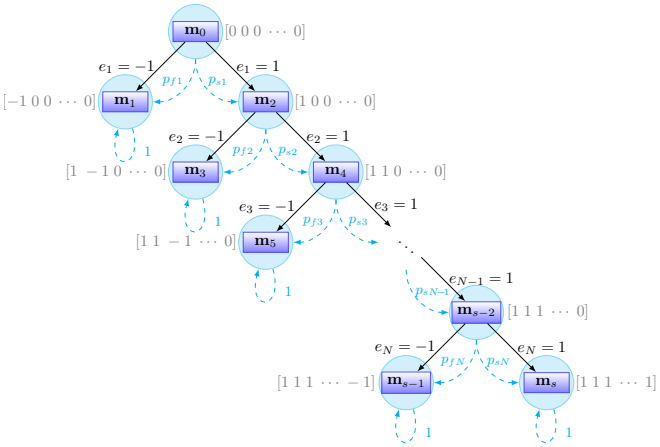


Fig. 8. Reachability graph of the sequence node (blue rectangles) with N children and its DTMC representation (cyan circles). When the node returns success/failure the related DTMC is in an absorbing state.

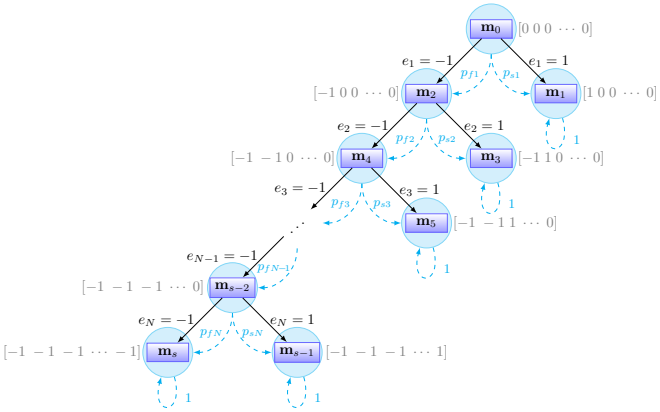


Fig. 9. Reachability graph of the selector node (blue rectangles) with N children and its DTMC representation (cyan circles). When the node returns success/failure the related DTMC is in an absorbing state.

B. Computing Properties of The DTMC

The RG of a BT node comprises all the reachable markings, the transitions between them describe events which have a certain success/failure probability. We can then map the node execution to a DTMC where the states are the markings in the RG and the one-step transition matrix $P^{\top}(k)$ is given by the probability of jump between markings, with off diagonal entries defined as follows:

$$p_{i,j}(k) = \begin{cases} p_{sj}(k) & \text{if } m_j(k) - m_i(k) = 1 \wedge \mathbf{e}_i \in \mathcal{F}(\mathbf{m}_i(k)) \\ p_{fj}(k) & \text{if } m_i(k) - m_j(k) = 1 \wedge \mathbf{e}_i \in \mathcal{F}(\mathbf{m}_i(k)) \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

and diagonal entries defined as:

$$p_{i,i}(k) = 1 - \sum_j p_{i,j}(k) \tag{13}$$

In Figs. 8 and 9 the mapping from RG to a DTCM related to a sequence node and a selector node are shown, we choose not to depict the mapping of a parallel node, due to its large amount of states and possible transition between them.

Lemma 1: The mean sojourn time is:

$$\mathcal{T}_i = \sum_{j:\mathbf{e}_j \in \mathcal{F}(\mathbf{m}_i)} \frac{G^{(j)}_{2,1}}{\mu_j} + \frac{G^{(j)}_{3,1}}{\nu_j} \tag{14}$$

where $G^{(i)}$ is the matrix define as follows:

$$G^{(i)} = \begin{bmatrix} 1 & 0 & 0 \\ \text{avg}(p_{si}(t)) & 1 & 0 \\ \text{avg}(p_{fi}(t)) & 0 & 1 \end{bmatrix} \tag{15}$$

where $\text{avg}(\cdot)$ is the average function over time.

Proof: The DTMC moves from a state to another with a given probability disregarding the time spent in such states. To take into account both probabilities and time rates, that influence the mean sojourn time, we describe the child execution with a DTMC where we model as *absorbing* those states in which a child returns either success or failure. This model is graphically represented in Fig. 10. Then the MTTA of

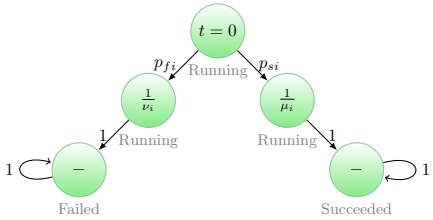


Fig. 10. DTMC of a child's execution. The label inside the state is the mean time spent in that state. The absorbing states are labeled as "−".

the DTMC is the mean time elapsed before a child returns either success or failure and is equal to the mean sojourn time described above. The MTTA is computed as the sum of steady state probabilities of reaching an absorbing state multiplied by the average time spent on each path to it, since the probabilities could be time varying, we consider their average. The one step transition matrix between transient states is given by

$$C^{(i)}(t) = \begin{bmatrix} 0 & 0 & 0 \\ p_{si}(t) & 0 & 0 \\ p_{fi}(t) & 0 & 0 \end{bmatrix}. \tag{16}$$

To retrieve the steady state probabilities we define the matrix $G^{(i)}$ as:

$$G^{(i)} \triangleq (\mathbb{I} - \text{avg}(C^{(i)}))^{-1}, \tag{17}$$

then (17) represents the mean probability of moving in a state, of a MC in Fig. 10, before being absorbed [12]. Hence its MTTA is computed as $\frac{G^{(j)}_{2,1}}{\mu_j} + \frac{G^{(j)}_{3,1}}{\nu_j}$. The (i, j) entry of the matrix (17) gives the probability of moving from state j to state i in the DTMC, before an event occurs. Therefore, taking into account all the feasible events, the mean sojourn time of the state i is (14).

C. Probability Distribution Over Time

Given the infinitesimal generator matrix $Q(t)$, we compute the probability distribution over time of the node according to (6) with the initial condition $\pi_0 = [1\ \mathbf{0}]^\top$ that represents the state in which none of the children have returned success/failure yet.

1) Time To Fail and Time To Succeed: Since we are interested in differentiating between time to succeed/fail we make a computation similar to the MTTA made in Section II. To derive a closed form of the mean time to fail (MTTF) and mean time to succeed (MTTS) we rearrange the state space of the DTMC so that the initial state is first, the other transient states are second, the failure states are second last and the success states are last:

$$\tilde{P}^\top(k) = \begin{bmatrix} T(k) & \mathbb{0} & \mathbb{0} \\ R_F(k) & \mathbb{I} & \mathbb{0} \\ R_S(k) & \mathbb{0} & \mathbb{I} \end{bmatrix} \tag{18}$$

where T is a $(|\mathcal{S}|-|\mathcal{S}_A|)\times(|\mathcal{S}|-|\mathcal{S}_A|)$ matrix describing the one-step transition from a transit state to another one, R_F is a $|\mathcal{S}_F| \times |\mathcal{S}_F|$ matrix describing the one-step transition from a transit state to a failure state, R_S is a $|\mathcal{S}_S| \times |\mathcal{S}_S|$ matrix describing the one-step transition from a transit state to a success state. We call this rearrangement *canonization* of the state space. MTTF and MTTS are meant to be calculated at steady state, hence we focus our attention on the probability of leaving a transition state:

$$\prod_{k=1}^{\infty} T(k) \triangleq U. \tag{19}$$

Considering i as the initial transient state, the entries u_{ij} is the mean number of visits of j starting from i above being absorbed, we have to distinguish the case in which the absorbing state is a failure state from the case in which it is a success state:

$$U^F(k) \triangleq R_F(k)U \tag{20}$$

$$U^S(k) \triangleq R_S(k)U. \tag{21}$$

Equations (20) and (21) represent the mean number of visits before being absorbed in a failure or success state respectively. Now let A be a matrix with the ij-th entry defined as $\exp(t_{ij})$ where t_{ij} is the time needed to transit from a state j to a state i if they are neighbors in the DTMC, 0 otherwise. Since the state space is arranged by the canonization described above, the matrix A has the following form:

$$A = \begin{bmatrix} A_T(k) & \mathbb{0} & \mathbb{0} \\ A_F(k) & \mathbb{0} & \mathbb{0} \\ A_S(k) & \mathbb{0} & \mathbb{0} \end{bmatrix}. \tag{22}$$

To derive MTTF (MTTS) we take into account the mean time needed to reach every single failure (success) state with its probability, normalized over the probability of reaching any failure (success) state, starting from the initial state. Hence we sum the probabilities of reaching a state starting from the initial one, taking into account only the first column of the matrices.

Due to the considerations above, the MTTF is computed as follows

$$MTTF = \text{avg}\left(\frac{\sum_{i=1}^{|\mathcal{S}_F|} u^F_{i1}(k)\log(h^F_{i1}(k))}{\sum_{i=1}^{|\mathcal{S}_F|} u^F_{i1}(k)}\right) \tag{23}$$

where:

$$H^F(k) \triangleq A_F(k) \prod_{k=1}^{\infty} A_T(k). \tag{24}$$

In a similar way, the MTTS is computed as follows:

$$MTTS = \text{avg}\left(\frac{\sum_{i=1}^{|\mathcal{S}_S|} u^S_{i1}(k)\log(h^S_{i1}(k))}{\sum_{i=1}^{|\mathcal{S}_S|} u^S_{i1}(k)}\right) \tag{25}$$

where:

$$H^S(k) \triangleq A_S(k) \prod_{k=1}^{\infty} A_T(k). \tag{26}$$

D. Main result

We are now ready to state the solution to Problem 1.

Proposition 1: Given a SBT, with known probabilistic parameters for actions and conditions, we can compute probabilistic measures for the rest of the tree as follows: For each node whose children have known probabilistic measures we compute the related DTMC. Now the probability of a node to return success $p_s(t)$ (failure $p_f(t)$) is given by the sum of the probabilities of the DTMC of being in a success (failure) state. Let $\mathcal{S}_S \subset \mathcal{S}_A$, and $\mathcal{S}_F \subset \mathcal{S}_A$ be the set of the success and failure states respectively of a DTMC related to a node, i.e. those states representing a marking in which the node returns success or failure, with $\mathcal{S}_F \cup \mathcal{S}_S = \mathcal{S}_A$ and $\mathcal{S}_F \cap \mathcal{S}_S = \emptyset$.

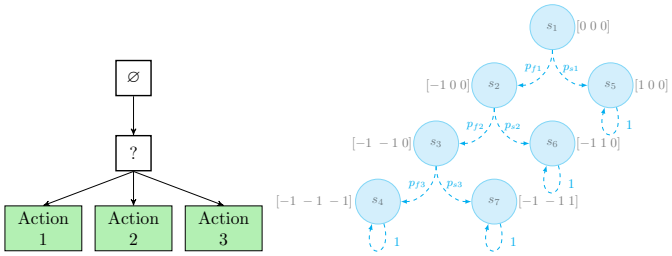
$$p_s(t) = \sum_{i:s_i \in \mathcal{S}_S} \pi_i(t) \tag{27}$$

$$p_f(t) = \sum_{i:s_i \in \mathcal{S}_F} \pi_i(t) \tag{28}$$

where $\pi(t)$ is the probability vector of the DTMC related to the node (i.e. the solution of (6)). Similarly to the MTTA, the MTTS (MTTF) for a node as we said before is given by a random variable with exponential distribution and rate given by the inverse of the MTTS (MTTF) since for such random variable the mean time is given by the inverse of the rate.

$$\mu = MTTS^{-1} \tag{29}$$

$$\nu = MTTF^{-1} \tag{30}$$



(a) BT of example. (b) Markov Chain.

Fig. 11. BT and related MC modeling the plan of Example 2.

Two examples, one small that is solved in detail, and a more complex one, see Section V, are now presented.

Example 2: We will now show the computation of probabilistic parameters for an example SBT. Considering the tree showed in Fig. 11(a), its probabilistic parameters are given by evaluating the selector node, since it is the child of the root node. The given probabilistic parameters related to the i-th action are:

- p_{f_i} probability of failure
- p_{s_i} probability of success
- ν_i failure rate
- μ_i success rate

The DTMC related as shown in Fig. 11(b) has $\mathcal{S} = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7\}$, $\mathcal{S}_F = \{s_4\}$ and $\mathcal{S}_S = \{s_5, s_6, s_7\}$. According to Equation (14) the mean sojourn times are collected in the following vector

$$\mathcal{T} = \left[\begin{array}{ccc} \frac{p_{s_1}}{\mu_1} + \frac{p_{f_1}}{\nu_1}, & \frac{p_{s_2}}{\mu_2} + \frac{p_{f_2}}{\nu_2}, & \frac{p_{s_3}}{\mu_3} + \frac{p_{f_3}}{\nu_3} \end{array} \right] \tag{31}$$

The infinitesimal generator matrix is defined, according to (5), as follows:

$$Q = \left[\begin{array}{ccccccc} \frac{-\mu_1\nu_1}{p_{s_1}\nu_1+p_{f_1}\mu_1} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{\mu_1\nu_1 p_{f_1}}{p_{s_1}\nu_1+p_{f_1}\mu_1} & \frac{-\mu_2\nu_2}{p_{s_2}\nu_2+p_{f_2}\mu_2} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{\mu_2\nu_2 p_{f_2}}{p_{s_2}\nu_2+p_{f_2}\mu_2} & \frac{-\mu_3\nu_3}{p_{s_3}\nu_3+p_{f_3}\mu_3} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{\mu_3\nu_3 p_{f_3}}{p_{s_3}\nu_3+p_{f_3}\mu_3} & 0 & 0 & 0 & 0 \\ \frac{\mu_1\nu_1 p_{s_1}}{p_{s_1}\nu_1+p_{f_1}\mu_1} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{\mu_2\nu_2 p_{s_2}}{p_{s_2}\nu_2+p_{f_2}\mu_2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{\mu_3\nu_3 p_{s_3}}{p_{s_3}\nu_3+p_{f_3}\mu_3} & 0 & 0 & 0 & 0 \end{array} \right]. \tag{32}$$

The probability vector, according to (6), is given by:

$$\pi(t) = \left[\begin{array}{ccccccc} \pi_1(t) & \pi_2(t) & \pi_3(t) & \pi_4(t) & \pi_5(t) & \pi_6(t) & \pi_7(t) \end{array} \right]^\top \tag{33}$$

We can now derive closed form expression for MTTS and MTTF. Using the decomposition in (18), the matrices computed according Equations 21 and 20 are:

$$U^S = \left[\begin{array}{ccc} p_{s_1} & 0 & 0 \\ p_{f_1}p_{s_2} & p_{s_2} & 0 \\ p_{f_1}p_{f_2}p_{s_3} & p_{f_2}p_{s_3} & p_{s_3} \end{array} \right] \tag{34}$$

$$U^F = \left[\begin{array}{ccc} p_{f_1}p_{f_2}p_{f_3} & p_{f_2}p_{f_3} & p_{f_3} \end{array} \right] \tag{35}$$

Note that U^S is a 3×3 matrix and U^S is a 1×3 matrix since there are 3 transition states, 3 success state and 1 failure state. For action i we define $t_{f_i} = \nu_i^{-1}$ the time to fail and $t_{s_i} = \mu_i^{-1}$ the time to succeed. The non-zero entries of the matrix given by (22) are:

$$\begin{array}{lll} a_{2,1} = e^{t_{f_1}} & a_{3,2} = e^{t_{f_2}} & a_{4,3} = e^{t_{f_3}} \\ a_{5,1} = e^{t_{s_1}} & a_{6,2} = e^{t_{s_2}} & a_{7,3} = e^{t_{s_3}} \end{array} \tag{36}$$

from which we derive (24) and (26) as:

$$H^S = \left[\begin{array}{ccc} e^{t_{s_1}} & 0 & 0 \\ e^{t_{f_1}}e^{t_{s_2}} & e^{t_{s_2}} & 0 \\ e^{t_{f_1}}e^{t_{f_2}}e^{t_{s_3}} & e^{t_{f_2}}e^{t_{s_3}} & e^{t_{s_3}} \end{array} \right] \tag{37}$$

$$H^F = \left[\begin{array}{ccc} e^{t_{f_1}}e^{t_{f_2}}e^{t_{f_3}} & e^{t_{f_2}}e^{t_{f_3}} & e^{t_{f_3}} \end{array} \right] \tag{38}$$

Using Equations (23) and (25) we obtain the MTTS and MTTF. Finally, the probabilistic parameters of the tree are expressed in a closed form according Equations (27)-(30):

$$p_s(t) = \pi_5(t) + \pi_6(t) + \pi_7(t) \tag{39}$$

$$p_f(t) = \pi_4(t) \tag{40}$$

$$\mu = \frac{p_{s_1}+p_{f_1}p_{s_2}+p_{f_1}p_{f_2}p_{s_3}}{p_{s_1}t_{s_1}+p_{f_1}p_{s_2}(t_{f_1}+t_{s_2})+p_{f_1}p_{f_2}p_{s_3}(t_{f_1}+t_{f_2}+t_{s_3})} \tag{41}$$

$$\nu = \frac{1}{t_{f_1}+t_{f_2}+t_{f_3}} \tag{42}$$

V. Numerical Verification and Complex Example

In this section we present a more complex example, extending Example 2 above. We use this example for two purposes, first, to verify the correctness of the proposed approach using Monte Carlo simulations, and second, to illustrate how changes in the SBT lead to different performance metrics.

Example 3: The task given to a two armed robot is to find and collect objects which can be found either on the floor, in the drawers or in the closet. The time needed to search for a desired object on the floor is less than the time needed to search for it in the drawers, since the latter has to be reached and opened first. On the other hand, the object is more likely to be in the drawers than on the floor, or in the closet. Moreover, the available policies for picking up objects are the one-hand and the two-hands grasps. The one-hand grasp most likely fails, but it takes less time to check if it has failed or not. Given these options, the task can be achieved in different ways, each of them corresponding to a different performance measure. The plan chosen for this example is modeled by the SBT shown in Fig. 12.

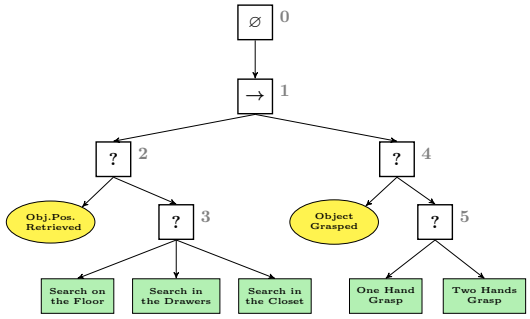


Fig. 12. BT modeling the search and grasp plan. The leaf nodes are labeled with a text, and the control flow nodes are labeled with a number, for easy reference.

The correctness of the analytical estimates can be seen from Table II. We compare the analytical solution derived using our approach with numerical results given by a massive Monte Carlo simulation carried out using a BT implementation in the Robot Operative System (ROS) [13] where actions and conditions are performed using ROS nodes with outcomes computed using the C++ random number generator with exponential distribution. The BT implementation in ROS was run approximately 80000 times to have enough samples to get numerical averages close to the true values. For each run we stored if the tree (and some sub-trees) succeeded or failed and how long it took, allowing us to estimate μ, ν, $p_s(t)$, $p_f(t)$ experimentally. The match is reported in Figs. 13-15 and in Table I. As can be seen, all estimates are within 0.0018 % of the analytical results.

Measure	Analytical	Numerical	Relative Error
μ_{Root}	5.9039E−3	5.8958E−3	0.0012%
ν_{Root}	4.4832E−3	4.4908E−3	0.0017%
μ_3	6.2905E−3	6.2998E−3	0.0014%
ν_3	2.6415E−3	2.6460E−3	0.0017%
μ_5	9.6060E−2	9.5891E−2	0.0018%
ν_5	4.8780E−2	4.8701E−2	0.0016%

TABLE I. Table comparing numerical end experimental results of MTTF and MTTS.

Label	μ	ν	$p_s(t)$	$p_f(t)$
Obj. Pos. Retrieved	–	–	$p_{s5}(t)$	$p_{f5}(t)$
Object Grasped	–	–	$p_{s4}(t)$	$p_{f4}(t)$
Search on the Floor	0.01	0.0167	0.3	0.7
Search in the Drawer	0.01	0.01	0.8	0.2
Search in the Closet	0.005	0.0056	0.2	0.8
One Hand Grasp	0.1	20	0.1	0.9
Two Hands Grasp	0.1	0.05	0.5	0.5

TABLE II. Table collecting given parameters, the label of the control flow nodes are reported in Fig. 12.

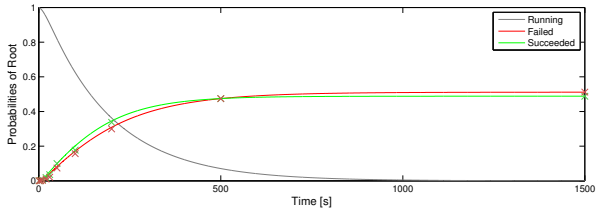


Fig. 13. Probability distribution over time for the Root node of the BT in Fig. 12. Solid lines are the analytical results, markers are the numerical results

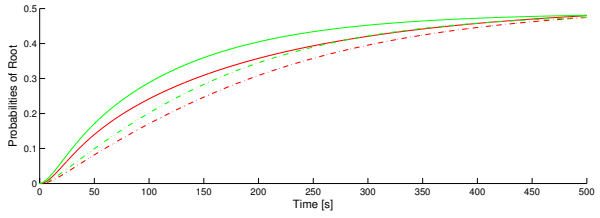
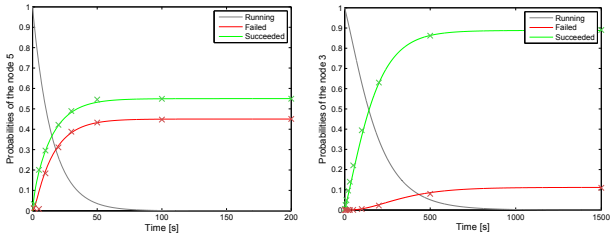


Fig. 14. Success/Failure probabilities in the case of searching on the floor first (dashed) and searching on the drawer first (solid).

We now show how the order of the execution nodes affects the plan performance. The results of swapping the order of



(a) (b)

Fig. 15. Matches of probability distribution over time related to Node 5 (a) and Node 3 (b)

"Search on the Floor" and "Search in the Drawers" are shown in Fig. 14. As can be seen, the success probability after 100s is about 30% when starting with the drawers, and about 20% when starting with the floor. The asymptotic probabilities are the same since the change was only a swap of order. We finish this section by noting that estimates like these can be used to optimize the execution plan for a given task.

VI. Conclusions

In this paper, we presented an analytical formalism that allowed us to do performance analysis of BTs. We showed why such estimates are important, and verified the approach by providing numerical results from a Monte Carlo simulation that agreed with the closed form expressions.

References

[1] A. Champandard, "Understanding Behavior Trees," *AiGameDev. com*, vol. 6, 2007.

[2] D. Isla, "Halo 3-building a Better Battle," in *Game Developers Conference*, 2008.

[3] C. Lim, R. Baumgarten, and S. Colton, "Evolving Behaviour Trees for the Commercial Game DEFCON," *Applications of Evolutionary Computation*, pp. 100–110, 2010.

[4] D. Perez, M. Nicolau, M. O'Neill, and A. Brabazon, "Evolving Behaviour Trees for the Mario AI Competition Using Grammatical Evolution," *Applications of Evolutionary Computation*, 2011.

[5] A. Shoulson, F. M. Garcia, M. Jones, R. Mead, and N. I. Badler, "Parameterizing Behavior Trees," in *Motion in Games*. Springer, 2011.

[6] I. Bojic, T. Lipic, M. Kusek, and G. Jezic, "Extending the JADE Agent Behaviour Model with JBehaviourtrees Framework," in *Agent and Multi-Agent Systems: Technologies and Applications*. Springer, 2011, pp. 159–168.

[7] P. Ögren, "Increasing Modularity of UAV Control Systems using Computer Game Behavior Trees," in *AIAA Guidance, Navigation and Control Conference, Minneapolis, MN*, 2012.

[8] J. A. D. Bagnell, F. Cavalcanti, L. Cui, T. Galluzzo, M. Hebert, M. Kazemi, M. Klingensmith, J. Libby, T. Y. Liu, N. Pollard, M. Pivtoraiko, J.-S. Valois, and R. Zhu, "An Integrated System for Autonomous Robotics Manipulation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 2012, pp. 2955–2962.

[9] A. Klökner, "Interfacing Behavior Trees with the World Using Description Logic," in *AIAA conference on Guidance, Navigation and Control, Boston*, 2013.

[10] D. Isla, "Handling Complexity in the Halo 2 AI," in *Game Developers Conference*, 2005.

[11] J. Norris, *Markov Chains*, ser. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1998, no. no. 2008.

[12] L. Elsner and S. Friedland, "Norm Conditions for Convergence of Infinite Products," vol. 250, no. 0, 1997, pp. 133 – 142.

[13] A. Marzinotto, M. Colledanchise, C. Smith, and P. Ögren, "Towards a Unified Behavior Trees Framework for Robot Control," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, June 2014.