Building Modular, Human-Interpretable AI Systems with Behavior Trees

Dianmu Zhang

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2020

Reading Committee:

Blake Hannaford, Chair

Dieter Fox

Hannaneh Hajishirzi

Program Authorized to Offer Degree:
Electrical and Computer Engineering

University of Washington

Abstract

Building Modular, Human-Interpretable AI Systems with Behavior Trees

Dianmu Zhang

Chair of the Supervisory Committee:
Professor Blake Hannaford
Electrical and Computer Engineering

Behavior Trees provide a structure to control the execution and switching of tasks and actions for autonomous agents, such as game characters or robots. Complex behaviors or tasks can be compartmentalized into sub-problems that correspond to actions. These actions become leaves of behavior tree, under the control of high level composite nodes. This modular arrangement makes behavior tree an ideal candidate for constructing solutions for many AI applications, advantages includes readily changeable and human-interpretable. Two main topics covered in this thesis are:

- IKBT: solving inverse kinematics with behavior trees. IKBT demonstrates how manually designed behavior trees with domain-specific knowledge are capable to solve problems that were usually handled by human experts before.

- Behavior tree for efficient hierarchical reinforcement learning. Behavior tree provides a modular problem formulation that facilitates searching within feasible actions and reapplying acquired knowledge. Behavior tree augmented reinforcement learning agents are more efficient in learning long horizon, sparse reward problems. Such problems are challenging for reinforcement learning.

TABLE OF CONTENTS

LIST OF FIGURES

DEDICATION

to my aunt, ZHANG Wei

you gave me the courage to purse my childhood dream

you believed in me before I believed in myself

ZHANG Wei, 1958-2006

Chapter 1

A BRIEF INTRODUCTION TO BEHAVIOR TREES

Behavior Trees provide a structure to control the execution and switching of tasks and actions for autonomous agents. Behavior trees first came to the center of attention in video game development for building game AIs, and just starts gaining more popularity in robotics research. It models intelligent agent behavior by incorporating specific tasks into action leaves[37] [41] [7] [10]. Behavior Trees have the advantages of composability and scalability compared to finite state machines.

Like many powerful tools, behavior trees are built from a small set of components - composite nodes and leaf nodes. The root of the behavior tree generates *ticks* and sends to its children. When a child node receives a tick, the node get executed. The child node returns one of the three possible values: *success*, if the goal is achieved; *failure*, if it fails to achieve the goal but the execution is complete; *running*, if the node is still in the process of execution.

Composite nodes control the flow of execution, including *selector*, *sequence*, and *decorators*. Leaf nodes carry out specific operations, include *action nodes* and *condition nodes*.

1.1 Node Description

1.1.1 Composite Nodes

Selector Node

Selector node (represented as "?") ticks its children from left to right. It returns success if any of the children returns success, and stop sending ticks for the rest of children. It returns failure if all children returns failure.

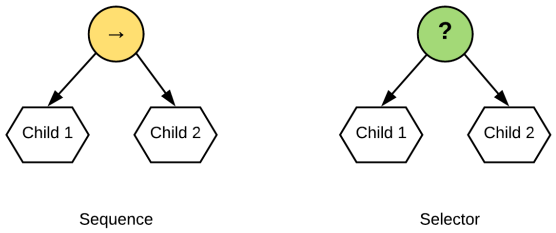


Figure 1.1: Behavior Trees - Composite Nodes

Sequence Node

Sequence node (represented as "→") ticks its children from left to right. It returns success if *all* the children returns success, otherwise, returns failure. As soon as one child returns failure, it does not tick the next child.

Decorators

Decorators are used to repeat a node or a subtree, include "Repeat till success", "Repeat till failure", "Repeat X times".

1.1.2 Leaf Nodes

Leaf nodes, by definition, are the deepest level of behavior tree.

Action Node

An action node performs a specific action. It returns success if the goal is completed, otherwise, returns failure (or running).

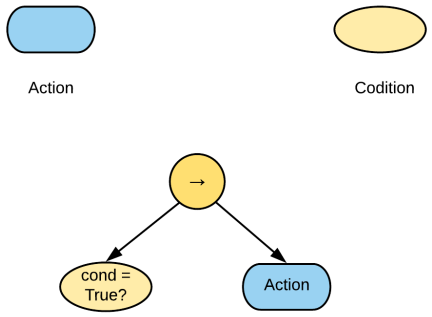


Figure 1.2: **Behavior Trees - Leaf Nodes.** (Bottom) Example use of condition node. If condition is satisfied, proceeds to the action.

Condition Node

A condition node contains a proposition or condition, usually about the environment. If it's true, it returns success, otherwise, returns failure.

1.2 Behavior Tree in Robotics

The use of behavior tress in robotics became more popular in the recent years. [40] proposed an unified behavior tree framework to control a humanoid robot NAO to perform grasping. Another example is the navigation problem in unmanned vehicles. Most people use finite state machine based architecture [59]. [48] used behavior tree to navigate a UAV in fuel-constraint environment, which is teleo-reactive. As for collaborative AI, CoSTAR is a behavior tree-based collaborative system for controlling KUKA, an industrial robotic arm. CoSTAR allows non-expert end users to instruct the robot to perform various tasks.

Chapter 2

IKBT: SOLVING CLOSED-FORM INVERSE KINEMATICS WITH BEHAVIOR TREE

Inverse kinematics solves the problem of how to control robot arm joints to achieve desired end effector positions, which is critical to any robot arm design and implementations of control algorithms. It is a common misunderstanding that closed-form inverse kinematics analysis is solved. Popular software and algorithms, such as gradient descent or any multi-variant equations solving algorithm, claims solving inverse kinematics but only on the numerical level. While the numerical inverse kinematics solutions are relatively straightforward to obtain, these methods often fail, due to dependency on specific numerical values, even when the inverse kinematics solutions exist. Therefore, closed-form inverse kinematics analysis is superior, but there is no generalized automated algorithm. Up till now, the high-level logical reasoning involved in solving closed-form inverse kinematics made it hard to automate, so it's handled by human experts. We developed IKBT, a knowledge-based intelligent system that can mimic human experts' behaviors in solving closed-from inverse kinematics using Behavior Tree. Knowledge and rules used by engineers when solving closed-from inverse kinematics are encoded as actions in Behavior Tree. The order of applying these rules is governed by higher level composite nodes, which resembles the logical reasoning process of engineers. It is also the first time that the dependency of joint variables, an important issue in inverse kinematics analysis, is automatically tracked in graph form. Besides generating closed-form solutions, IKBT also explains its solving strategies in human (engineers) interpretable form. This is a proof-of-concept of using Behavior Trees to solve high-cognitive problems [1].

---

[1]This paper has been published on *Journal of Artificial Intelligence Research*, [link here]

This work is published in JAIR [62].

2.1 Introduction

Symbolic inverse kinematics analysis is a non-trivial task critical for operation and design of robot manipulators as well as animated characters. For example, in a simple serial robot (wrist robot) of three degrees of freedom (joint variables A, B, and C), the inverse kinematics computation takes the desired end-effector pose as input (typically as a homogeneous transform, left hand side of the equation), and solves for joint angles or joint displacements from the forward kinematic equations (right hand side of the equation).

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & Px \\ r_{21} & r_{22} & r_{23} & Py \\ r_{31} & r_{32} & r_{33} & Pz \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta_1\cos\theta_2 & -\sin\theta_1\cos\theta_3 + \sin\theta_2\sin\theta_3\cos\theta_1 & \sin\theta_1\sin\theta_3 + \sin\theta_2\cos\theta_1\cos\theta_3 & 0 \\ \sin\theta_1\cos\theta_2 & \sin\theta_1\sin\theta_2\sin\theta_3 + \cos\theta_1\cos\theta_3 & \sin\theta_1\sin\theta_2\cos\theta_3 - \sin\theta_3\cos\theta_1 & 0 \\ -\sin\theta_2 & \sin\theta_3\cos\theta_2 & \cos\theta_2\cos\theta_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

The numerical solutions are often substituted for closed form symbolic solutions, where all elements in the desired end effector matrix (left hand side) are real numbers and solutions for joint variables are numerical values, using two major methods: A) gradient descent searches for a set of joint angles/length that minimize the cost function. Convergence of gradient descent can depend on the starting value and only generates one solution. Most existing software packages [13, 31] use a version of gradient descent as their core algorithm. The shared limitations include finding only one of the multiple solutions, convergence depending on the starting value, and problems with convergence near singular configurations. B) Solving multi-variant polynomial equations [39, 44]. Though this method generates multiple possible

solutions, it fails when the augmented transformation matrix is ill-conditioned, which is unavoidable in practice. And this method is often DOF-specific.

Comparatively, closed-form inverse kinematics analysis overcomes all these shortcomings of the numerical methods, but it's difficult to automate conceptually, because of the high-level mathematical reasoning needed. For example, in (2.1), above, closed-form inverse kinematics analysis for joint θ_1 begins by looking through all the equations and choosing two:

$$r_{21} = \sin\theta_1 \cos\theta_2$$
$$r_{11} = \cos\theta_1 \cos\theta_2$$

Though θ_2 is unsolved at this step, $cos(\theta_2)$ can be canceled out by dividing these two equations while calculating $atan2$. Depending on the value of $cos(\theta_2)$, θ_2 can have two solutions:

$$\begin{aligned}
\theta_{1s1} &= \text{atan}_2\left(r_{21}, r_{11}\right), && \cos(\theta_2) > 0 \\
\theta_{1s2} &= \text{atan}_2\left(-r_{21}, -r_{11}\right), && \cos(\theta_2) < 0 \\
\theta_1 &\quad \text{undefined}, && \cos(\theta_2) == 0
\end{aligned}$$

Several groups have attempted to automate symbolic inverse kinematics analysis starting in the 1990's [27, 21], which laid the foundation for our work. Their work is reviewed and compared with ours in detail in the next section.

Though the sufficient conditions for the existence of closed-formed IK solutions is well-established [51], the necessary condition remains unknown. For instance, a robot with three-intersecting axes has analytical IK solution; but it's not necessary for a robot to have three-intersecting axes to have closed-form IK solutions. In terms of degrees of freedom (DOF), any mechanism having greater than 6-DOF has unlimited number of analytical solutions, due to kinematic redundancy. IKBT solves cases up to 6-DOF.

In this work we develop an AI agent that solves closed-form inverse kinematics with the following goals: The agent

- Should solve closed-form inverse kinematics with a generalized algorithm applicable to most serial chain robot arms, without assumptions of configuration or degree-of-freedom

- Should explicitly use common knowledge that engineers use when solving the inverse kinematics problems, such as trig identities or the method of determinants, rather than relying on tricks for specific kinematic configurations

- Should have search and apply suitable knowledge or rules to equations containing unsolved joint variables as human experts do.

- Should be able to explain its solving strategy in an easy to interpret format

- Should be extensible and modifiable

To address these goals, we adapt Behavior Trees to construct an expert system, "IKBT", having the logical reasoning power to solve inverse kinematics symbolically without human supervision. A Behavior Tree - initially popular in video game AI, models intelligent agent behavior by incorporating specific tasks into action leaves [37, 41, 7, 10]. Behavior Trees have the advantages of composability and scalability compared to finite state machines.

The main contributions of this work are:

- We compactly encode the inverse kinematics logic and solution strategy in a Behavior Tree (see Work Flow and Architecture section).

- We code each knowledge-based solver into a modular leaf, forming a "tool box" which is organized and applied to equations and intermediate results by the Behavior Tree (see Transformations and Solvers section). The structure is readily extensible.

- IKBT generates a dependency *graph* of joint variables in the solutions, which specifies all possible poses. Tracking these dependencies facilitates grouping variables into distinct solutions, essential to downstream control softwares for robots (see Solution Graph section).

- IKBT successfully solves complicated robots, such as the 6-DOF commercial robot manipulator PUMA 560 and successfully solved 18 out of 19 test robots (95% success rate) (see Results section).

- On average, IKBT generates symbolic solutions and source code in a few minutes on a normal PC. The same work often takes a human expert hours to complete.

- IKBT generates a report of its results and solution method in LaTeX, and generates code in Python and C++, creating functions which implement the derived solutions including domain (reachability) checking of numerical inputs (see Pose Validation section).

- Inverse kinematics solutions from IKBT are verifiable with numerical computations (facilitated by the IKBT code generator) (see Result Verification subsection under Results).

- Implementation in a modern open-source, cross-platform, programming language (Python). IKBT requires few dependencies outside of the standard Python distribution (mainly the symbolic manipulation package `sympy` and the unit testing framework `unittest`).

2.1.1 Related Work and Comparison

It is a common misconception that automated closed-form inverse kinematics is a solved problem. A few research groups introduced a generalized method to perform inverse kinematics analysis by solving multipolynomial multivariate equations [39, 44]. However, the

original research [39], as well as authors of various textbooks, pointed out that this method involves a combination of numerical and symbolic manipulation (symbolic reduction at early steps, only auxiliary), and is thus not a closed-form solution. The multipolynomial method depends on numerical values in critical steps, such as eigenvalue computation. The resulting values for joint variables are only numerical. Other drawbacks of this method, as pointed out by Manocha and Canny (1994), include frequent poor conditioning of the matrices requiring inversion. Another explicit limitation of this method is that all joint variables need to be rotary - robots with one or more prismatic joints can't be solved by this method.

Previous research on closed-form inverse kinematics lays a good foundation for IKBT. One such example is a rule-based pattern matching approach (implemented as an expert system in LISP) [27] from which IKBT adapted the sin or cos solver, tangent solver, and simultaneous equation solver. Similar to IKBT, that system scanned a list of equations, found the ones matching patterns, and then fetched the respective solutions. Their method solved several commercial robots, including the more complicated PUMA 560. Limitations of this work include a hard coded framework for solution sequencing, and dependence on obsolescent software. In comparison, IKBT uses only 6 rule-based solvers, indicative of more efficient combinatorial logical reasoning.

A similar approach implemented rule-based solvers in LISP [60], although neither the detailed rules or the source code were made public. In this implementation, the system solved equations sequentially and stopped working on a variable as soon as it is solved. In contrast, IKBT's assigner node picks a variable first, and tries the entire toolbox for the chosen variable. We designed IKBT this way to get the optimal solution, in the situation where more than one solver applies to the same variable (choosing from multiple equations). IKBT ranks the solutions obtained and chooses the best solution according to specified criteria (described below). Unlike IKBT, previous research [27, 60] did not show the ability to find all possible solutions or tracking dependencies among variables .

A different method used elimination techniques to convert the set of kinematic equations into a univariate polynomial [21]. It is effective in solving specific robots. However, whether

their methods could be applied to other robots was not systematically tested. Also, because it uses an approach unlike what human experts do, it is harder to check the correctness of the solution or strategy: IKBT's toolbox contains only well known rules frequently used by human experts. Each rule is provided with a straightforward unit test.

An additional solver used a product-of-exponentials formula, which doesn't require D-H parameters, and is robust in dealing with kinematics singularities [6]. However, this solver only showed the capability of handling numerical inputs and rendering numerical solutions, and very limited solving capability for complex robots (6-DOF robots $\approx 50\%$) . Compared to this D-H parameters-free solver [6], IKBT handles symbolic input, generates closed-form solutions, and achieved better success rate with complex 5-DOF (100 %) and 6-DOF (80%) robots. Another solver used evolutionary algorithms to get an approximate inverse solution [5]. By contrast, IKBT computes exact symbolic closed-form solutions.

IKFast performs inverse kinematics analysis as part of the OpenRAVE package [17]. Instead of general solving techniques, IKFast adapts a case-specific hybrid approach.

We performed an in-depth source code analysis of IKFast which revealed the following major differences between IKFast and IKBT:

- Numerous lines of the IKFast source code specify the desired end effector position (the inverse kinematic input) as numerical values. Therefore, by definition, these results are not symbolic.

- Many numbers are presented in output equations and intermediate results of IKFast, where there should be only symbols in a closed form solution. The use of "iterations" in the IKFast solving code signals the underlying gradient descent/ascent method, which is prevalent in numerical IK process.

- Unlike IKBT, IKFast's results are complicated C++ code and not human understandable. Human interpretability is a major feature of IKBT.

- The output of IKFast is a `.cpp` file, which is not a symbolic solution. Even if C++

source code is claimed to be a symbolic solution, IKFast's code is too complex to manually verify. In addition to generating python and `.cpp` files, IKBT also generates LaTeXequations, obeying all standard conventions of a symbolic solution, that are compiled into human readable form with standard LaTeXtools.

Additional insights from IKFast code inspection reveal that IKFast categorizes robots by their number of DOFs, and uses hard-coded algorithms for arms with different DOFs. IKFast does generates a "dependency tree" like IKBT. However, a tree cannot represent the multiple independent dependencies we found for some variables in some robots. The graph-based representation generated by IKBT solves this problem.

As described in detail in Section 2.4.4, a dependency graph is essential to keep track of the joint poses, and make sure the solutions sets are complete and necessary. Complete means IKBT can find all possible joint poses if the solution exists. Necessary means that all solutions are unique, not duplicates of each other. Dependency tracking has previously been done manually. To the best of our knowledge, IKBT is the first to automate this process.

2.2 Work Flow and Architecture

2.2.1 Work Flow

As shown in Fig. 2.1, the system input takes symbolic Denavit—Hartenberg (DH) parameters and calculates symbolic forward kinematic equations in the form of a 4x4 homogeneous transformation, T_0^6, is computed symbolically as

$$T_s = T_6^0 = T_1^0 T_2^1 T_3^2 T_4^3 T_5^4 T_6^5 \tag{2.2}$$

By convention, each transformation matrix takes a coordinate from the subscript frame and transforms it to the superscript frame (T_6^0 transforms from frame 6 to frame 0).

We denote the desired robot end effector pose as T_d (see left hand side of (2.1)). Then the inverse kinematics problem can be stated as solving

$$T_d = T_6^0(q_1, \ldots, q_6) \tag{2.3}$$

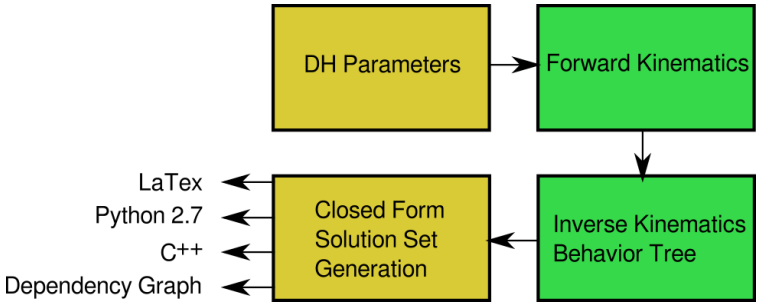


Figure 2.1: **Work Flow.** A Forward kinematics module computes symbolic kinematic equations to be solved ($T_d = T_s$) given the input DH parameters. Then the equations are evaluated for closed-form inverse kinematics solutions to each joint variable. Upon solving a robot, along with the solutions, a dependency graph, Latex report, and Python/C++ code are generated as convenience features.

(where q_i are the unknown joint variables) for all sets of joint variables ($q_i = \theta_i$ or d_i) which satisfy 2.3. Related equations which can be used to find soluble equations include:

$$[T_1^0]^{-1} T_d = [T_1^0]^{-1} T_s \tag{2.4}$$

$$[T_2^1]^{-1} [T_1^0]^{-1} T_d = [T_2^1]^{-1} [T_1^0]^{-1} T_s \tag{2.5}$$

$$[T_{n-1}^{n-2}]^{-1} ... [T_1^0]^{-1} T_d = [T_{n-1}^{n-2}]^{-1} ... [T_1^0]^{-1} T_s \tag{2.6}$$

IKBT first symbolically calculates and simplifies these intermediate results (2.4 - 2.6) to augment (2.2). These intermediate results are stored in a buffer which is used as a lookup (joint variable - equations) table for solvers. Each of these matrix equations creates 12 scalar equations (one for each element in the first three rows) which can be searched for solvable equations. The individual equations are sorted into three lists according to the number of unsolved variables in each equation (1, 2, and 3-or-more unknowns). As each variable is solved, this scan is repeated.

The Behavior Tree's leaf nodes transform, or identify and solve, a particular kind of equation (see the Section 2.3). For example, one pair of nodes identifies and solves scalar equations of one-unknown of the form $A = \sin(B\theta_j + C)$ or $A = \cos(B\theta_j + C)$ where A, B, C are known expressions, and θ_j is unknown.

After all joint variables are solved, the solution graph and the solution vectors (2^n joint vectors in symbolic form correctly associating the multiplicity of each variable) are constructed. A LaTeXreport, C++ and Python code, containing symbolic solutions for all possible poses, is also generated.

2.2.2 Architecture

Behavior Trees have been explored in the context of humanoid robot control [41, 11, 3], collaborative robotics [20, 10], and as a modeling language for intelligent robotic surgical procedures [28, 23].

The work reported here is the first to our knowledge to use Behavior Trees to encode algorithms for reasoning about and solving mathematical equations symbolically. When implementing intelligent behavior with Behavior Trees, the designer of a robotic control system breaks the task down into modules (Behavior Tree leaves) which return either "success" or "failure" when called by parent nodes. Higher level nodes define composition rules to combine the leaves including: Sequence, Selector, and Parallel node types which also return "success" or "failure". A Sequence node defines the order of execution of leaves and returns success if all leaves succeed in order. A Selector node (called "Priority" by some authors) tries leaf behaviors in a fixed order, returns success when a node succeeds, and returns failure if all leaves fail. We also implemented a "Parallel" node (represented as "OR" in Fig. 2.2), which executes all leaves regardless of their return status, and returns success if any one of the leaves succeeds. The IKBT structure used for our current results is shown in Fig.2.2.

Before solving, IKBT looks through 1- and 2- unknown equation lists, and applies sum-of-angle and substitution transformations which may reduce number of unknown variables. The "Assigner" node assigns the current variable to all solvers. For each joint variable, it tries out

all solvers in the toolbox until it is solved (or we reach the maximum trial number). When a joint variable is solved, the solver marks it as solved, and reduces the number of unsolved variables by one, for all equations involved this variable. When multiple solvers can solve a joint variable, the solutions are compared using a "Ranker" node which selects preferred solution forms over others. For example, $\theta_4 = \text{atan2}(y, x)$ is preferred over $\theta_4 = \arcsin(y/r)$ because it has only one solution. IKBT repeats this until all variables are solved. Finally the solutions, report, code generation, and dependency graph are generated.

2.3 Transformations and Solvers

In the following subsections, θ_i and d_i represent rotary and prismatic joint variables (q_i). a, b, c, etc., stand for known constant DH parameters.

2.3.1 Transformations

Transformation nodes make equations easier to solve by reducing the number of unknown variables.

1. Sum of angle transform

$$\sin(\theta_x \pm \theta_y) \rightarrow \sin(\theta_{xy})$$

$$\cos(\theta_x \pm \theta_y) \rightarrow \cos(\theta_{xy})$$

   Although the sum-of-angle simplification is done by sympy's `simplify` operation, creation of a new variable (θ_{xy}) is done by this node. This transformation also works for the sum of 3 angles (which allows solution of robot arms having 3 parallel sequential axes).

2. Substitution transform: looks for two equations such that one contains the other, and replaces the partial expression with an unknown value. For example, in the following pair of equations:

$$\sin(\theta_x) + a \cdot \cos(\theta_y) = b$$

$$a \cdot \cos(\theta_y) = c$$

The first equation can be transformed to:

$$\sin(\theta_x) + c = b$$

Eliminating one unknowns so that θ_x can be solved.

2.3.2 Rule-based Solvers

The IKBT contains a set of solvers that identifies an expression that fits a rule set and returns the respective solutions. These rules are used by human experts when solving inverse kinematics problems, and not are specific to any DOF or robot configuration. Solvers 1 - 4 are derived from straightforward algebra and trigonometry. 5-6 are adapted from respective literature [27, 14].

1. algebraic solver

   Identifies pattern

   $$a + b\theta = c$$

   where $b \neq 0$. Solves for

   $$\theta = \frac{c - a}{b}$$

   as well as

   $$a + bd_x = c$$

   giving

   $$d_x = \frac{c - a}{b}$$

2. sine or cosine solver

   Identifies pattern

$$\sin(\theta) = a, \qquad \cos(\theta) = b$$

   Solves for two solutions in each case:

$$\theta = \arcsin(a), \qquad \theta = \arccos(b)$$

   and

$$\theta = \pi - \arcsin(a), \qquad \theta = -\arccos(b)$$

3. tangent solver

   identifies a pattern in two equations containing

$$\sin(\theta) = aC_1 \quad and \quad \cos(\theta) = bC_2$$

   If neither C_1 or C_2 contain unsolved variables:

$$\theta = \text{atan2}(aC_1, bC_2)$$

   Sometimes C_1 and C_2 contain common unsolved variables, which can be canceled out by division. In this case we use a new coefficient C:

$$C = \frac{C_1}{C_2}$$

$$\theta = \text{atan2}(aC, b) \quad C > 0$$

$$\theta = \text{atan2}(-aC, -b) \quad C < 0$$

   Terms which are solvable by tangent solver are often also solvable by sine or cosine solver. As shown in Fig 2.2, IKBT takes this into consideration by comparing the

solutions from the above-mentioned solvers, and determines the optimal solution. This selection is done by the Ranking node.

4. Sine and cosine solver

   Identifies

   $$a \cdot \sin(\theta) + b \cdot \cos(\theta) = 0$$

   giving two solutions:

   $$\theta = \text{atan2}(-b, a), \quad and \quad \theta = \text{atan2}(-b, a) + \pi$$

   as well as

   $$a \cdot \sin(\theta) + b \cdot \cos(\theta) = c$$

   giving

   $$\theta = \text{atan2}(a, b) + \text{atan2}(\pm\sqrt{a^2 + b^2 - c^2}, c)$$

5. Simultaneous equation solver [27] Identifies two equations:

   $$a\sin(\theta) + b\cos(\theta) = c \qquad a\cos(\theta) - b\sin(\theta) = d$$

   Giving the solution:

   $$\theta = \text{atan2}(ac - bd, ad + bc)$$

6. x^2y^2 solver

   Identifies two equations that contain P_x, P_y, and/or P_z, that can be squared and added together to cancel out unsolved variables (other than the intended variable), and get a new equation with pattern [14] :

   $$-\sin(\theta)P_x + \cos(\theta)P_y = d$$

Giving solutions:

$$\theta = \text{atan2}(P_y, P_x) - \text{atan2}(d, \pm\sqrt{{P_x}^2 + {P_y}^2 - d^2})$$

2.4 Solution Graph

IKBT provides solution graph to track the dependency among joint variables. IKBT traces these dependencies in a mathematical sense, based on their symbolic solution equations. Intuitively, in the physical world, it can be interpreted as when one joint is set to a new value, the dependent joints values changes with it, in order to achieve certain end effector position and orientation.

2.4.1 Origins of Dependency

The solutions produced by inverse kinematics are typically interdependent in that results obtained early in the process are used to solve later results. For example, one may have

$$\theta_4 = \text{asin}\left(\frac{1}{l_4}\left(Pz - l_3 + l_5 \cos\left(\theta_{45}\right)\right)\right)$$

in which θ_4 depends on $\cos(\theta_{45}) = \cos(\theta_4 + \theta_5)$ as well as some constants. In principle, it is possible to substitute these dependencies until there are no joint variables on the right hand side, but this makes the solutions difficult to compare with previously published hand solutions. In the above example, there are two solutions to the asin() operator, and additional multiplicity could come from the solution method for θ_{45}.

Thus the two sources of multiple solutions are: A) each joint variable may have multiple solutions due to its solver's characteristic; and B) dependence of the solution on other solved joint variables.

To further illustrate the sources of multiplicity and dependency, we use the following example. The goal is to solve variables θ_1 and θ_2 from these equations:

$$\cos(\theta_1) = a$$

$$\sin(\theta_2) + \sin(\theta_1) = b$$

From the first equation, θ_1 is solved as:

$$\theta_1 = \begin{cases} \theta_{1s1} = & \arccos(a) \\ \theta_{1s2} = & -\arccos(a) \end{cases}$$

where we have used the subscript s to separate joint numbers from solution numbers. Here, θ_{1s2} means the second solution of θ_1. θ_1 has 2 solutions due to the nature of solver arccos(). Now that θ_1 is solved, we can solve θ_2 as:

$$\theta_2 = \begin{cases} \theta_{2s1} = & \arcsin(b - \sin(\theta_{1s1})) \\ \theta_{2s2} = & \pi - \arcsin(b - \sin(\theta_{1s1})) \\ \theta_{2s3} = & \arcsin(b - \sin(\theta_{1s2})) \\ \theta_{2s4} = & \pi - \arcsin(b - \sin(\theta_{1s2})) \end{cases}$$

θ_2 has 4 solutions, because its solver () generates 2 solutions, and on top of that, it θ_2 depends on the two solutions of θ_1. Example solution dependency is illustrated in 2.3.

In the resulting graph (shown in Fig. 2.3 a), each joint solution (e.g. θ_{2s1}, θ_{2s2}, etc.) is a node. A parent node is the node that appears in another node's solution expressions, in this example, θ_{1s1} is the parent of θ_{2s2} . A node and its parent/child node are connected with an edge.

2.4.2 Redundancy Detection and Dependency Tracking

When building a dependency graph, we implemented redundancy elimination to ensure the correct relations between joint variables. Redundancy is defined as a dependency that traced back to a higher level parent can be mediated by a lower level and direct parent. If a joint variable θ_5 has the following solutions:

$$\theta_{5s1} = \arccos(l + \cos(\theta_{4s1}))$$

$$\theta_{5s2} = -\arccos(l + \cos(\theta_{4s1}))$$

And θ_6 has solutions:

$$\theta_{6s1} = atan2(a + \cos(\theta_{4s1}), b + \sin(\theta_{5s1}))$$

$$\theta_{6s2} = atan2(a + \cos(\theta_{4s2}), b + \sin(\theta_{5s2}))$$

Though the solution of θ_6 involves both θ_4 and θ_5, its dependency to θ_4 is redundant. Given that θ_5 is also dependent on θ_4, the effects of choosing different θ_4 values (if applicable) on θ_6 are conveyed through θ_5. Therefore, when building a graph, only the edges between direct child-parents are added, in this case, Edge (θ_6, θ_5) and Edge(θ_5, θ_4). Shown in Fig. 2.3 b).

Classic search algorithms (breadth-first search and depth-first search) are used to traverse the graph and find correct ancestor nodes, where the current variable is the start point and the ancestors are the goals.

2.4.3 Grouping Variables

As required by many planning and control algorithms, IKBT is capable of grouping variables into solution sets that have all possible joint configurations for the given end-effector configuration. To generate correct sets of solutions, the following steps are carried out to match the variables: First, all parents nodes are extracted from each solution expression, forming subsets of variables. Secondly, the subsets are sorted by size of their content. Search starts from the largest subsets, and looks for the variables that are a part of the joint space, but not in the set, till all variables are found. A scoring system is applied on all subsets (other than the starting set) to focus the search on the more likely candidate first.

Using the Fig. 2.3 a) as an example, the solutions can be grouped into: [θ_{1s1}, θ_{2s1}], [θ_{1s1}, θ_{2s2}], [θ_{1s2}, θ_{2s3}], and [θ_{1s2}, θ_{2s4}].

2.4.4 Graph Representation

The multiple dependencies can be linked by a common dependency further up, or they can be independent. Although traditionally this structure is represented as a tree, we discovered cases in which variables have multiple independent "parents" and thus a graph is required instead.

For example, θ_1 and θ_2 are independent to each other:

$$\begin{aligned}
\theta_{1s1} &= \arcsin(a) \\
\theta_{1s2} &= -\arcsin(a) + \pi \\
\theta_{2s1} &= \arccos(b) \\
\theta_{2s2} &= -\arccos(b)
\end{aligned}$$

And θ_3 depends on both θ_1 and θ_2:

$$\begin{aligned}
\theta_{3s1} &= \arccos(a + cos(\theta_{1s1}) + atan2(b, sin(\theta_{2s1})c) \\
\theta_{3s2} &= \arccos(a + cos(\theta_{1s1}) + atan2(b, sin(\theta_{2s2})c) \\
\theta_{3s3} &= \arccos(a + cos(\theta_{1s2}) + atan2(b, sin(\theta_{2s1})c) \\
\theta_{3s4} &= \arccos(a + cos(\theta_{1s2}) + atan2(b, sin(\theta_{2s2})c)
\end{aligned}$$

The dependency graph is shown in Fig. 2.3 c).

One of the example robot solutions in the Results section shows the necessity of using graph representation.

2.5 Verification & Validation

This section answers two important questions: 1) How do we know if the IK solutions from IKBT are correct?, and 2) How to make sure that a requested end effector pose is valid, that

IK solutions exist?

2.5.1 Solution Verification

To prove that the inverse kinematics solution equations from IKBT are correct, we conducted the following verification process, as shown in Fig. 2.4 . First, we constructed a valid numerical transformation matrix from a reachable joint-space pose (note that joint limits are completed only after an IK solution is evaluated). Then we used numbers from the transformation matrix and the inverse kinematics solutions equations to get the numerical values for each pose. If the inverse kinematics solutions are correct, one of the poses should match the starting pose value. Next, we computed the forward kinematics using each numerical pose. If the resulted transformation matrix is the same (within a stringent range) as the starting matrix, then we can safely draw the conclusion that this pose has correct inverse kinematics solution. The completeness of solutions is verified by comparing IKBT results to existing literature, and analyzed theoretically by tracing through joint dependency.

2.5.2 Pose Validation

If a pose is not reachable by the robot (for example due to distance of a point extending beyond the length of the arm, but not considering joint limits), at least in generated code output, the solution must have a means to detect this case. In inverse kinematic solution equations, unreachable poses generate intermediate values outside the domain of transcendental functions, for example:

$$\theta_2 = \arcsin(x) \qquad x = 1.2$$

or would require complex joint angles:

$$d_3 = \sqrt{x} \qquad x = -5$$

Both the C++ and Python output modules of IKBT automatically generate code which checks numerical arguments of inverse trig functions and square roots for such cases and

returns a flag to indicate an unreachable pose. In practice, users can choose among all IK solutions that satisfies joint limits.

2.6 Experimental Results

2.6.1 General Performance

We tested IKBT on many sets of DH parameters, representing serial arm robot designs (including commercial robots, and solved design examples from student homework), the successful solving rate is listed in Table 3.1. As the DOF number increases, the problem becomes more complex and the success rate decreases. In general it solves most of the robots, up to 6 DOF. Note that IKBT can solve robots regardless of their configurations, e.g. IKBT does not require robots having three intersecting axes.

Source code can be found at: https://github.com/uw-biorobotics/IKBT. The DH parameters of all these robots are stored as part of the source code (in `ik_robots.py`), for purpose of testing and reproducing the results. Instructions are on the GitHub page.

2.6.2 Example Solutions PUMA 560

Here PUMA560 is used as an example to illustrate how IKBT solves inverse kinematics problems. PUMA 560 was a commercial robot with six rotary joints and four joint offsets, well-known for its challenging inverse kinematics properties. The PUMA 560 has three axes

number of DOF	Test examples	Solved
4	4	4
5	10	10
6	5	4

Table 2.1: IKBT test results

Table 2.2: PUMA560 DH parameters

Link	α_{N-1}	a_{N-1}	d_N	θ_N
1	0	0	0	θ_1
2	$-\pi/2$	0	0	θ_2
3	0	a_2	d_3	θ_3
4	$-\pi/2$	a_3	d_4	θ_4
5	$\pi/2$	0	0	θ_5
6	$-\pi/2$	0	0	θ_6

intersecting at its wrist. The known variables are: a_2, a_3, d_3, and d_4. In all the solution results below, the equations are produced directly from the IKBT LaTeXoutput. A few edits have been made only to improve formatting at this column width.

Based on input of the Puma DH parameters, first, forward kinematics was calculated:

$$T_d = T_s$$

$$T_0^6 = T_0^1 T_1^2 T_2^3 T_3^4 T_4^5 T_5^6$$

(where T_d is the "desired position", T_s symbolic expressions, T_0^6 transformation matrix from frame 0 to 6)

$$T_0^6 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & P_x \\ r_{21} & r_{22} & r_{23} & P_y \\ r_{31} & r_{32} & r_{33} & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} v1 & v2 & v3 & v4 \end{bmatrix}$$

$v_1 =$

$$\begin{bmatrix} c_6(-c_1 s_{23} s_5 + c_5(c_1 c_{23} c_4 + s_1 s_4)) - s_6(c_1 c_{23} s_4 - c_4 s_1) \\ c_6(c_5(-c_1 s_4 + c_{23} c_4 s_1) - s_1 s_{23} s_5) - s_6(c_1 c_4 + c_{23} s_1 s_4) \\ -c_6(c_{23} s_5 + c_4 c_5 s_{23}) + s_{23} s_4 s_6 \\ 0 \end{bmatrix}$$

$v_2 =$

$$\begin{bmatrix} -c_6(c_1 c_{23} s_4 - c_4 s_1) - s_6(-c_1 s_{23} s_5 + c_5(c_1 c_{23} c_4 + s_1 s_4)) \\ -c_6(c_1 c_4 + c_{23} s_1 s_4) - s_6(c_5(-c_1 s_4 + c_{23} c_4 s_1) - s_1 s_{23} s_5) \\ c_6 s_{23} s_4 + s_6(c_{23} s_5 + c_4 c_5 s_{23}) \\ 0 \end{bmatrix}$$

$$v_3 = \begin{bmatrix} -c_1 c_5 s_{23} - s_5(c_1 c_{23} c_4 + s_1 s_4) \\ -c_5 s_1 s_{23} - s_5(-c_1 s_4 + c_{23} c_4 s_1) \\ -c_{23} c_5 + c_4 s_{23} s_5 \\ 0 \end{bmatrix}$$

$$v_4 = \begin{bmatrix} a_2 c_1 c_2 + a_3 c_1 c_{23} - c_1 d_4 s_{23} - l_3 s_1 \\ a_2 c_2 s_1 + a_3 c_{23} s_1 + c_1 l_3 - d_4 s_1 s_{23} \\ -a_2 s_2 - a_3 s_{23} - c_{23} d_4 \\ 1 \end{bmatrix}$$

where $c_1 = cos(\theta_1)$, $s_{23} = sin(\theta_2 + \theta_3)$ etc.

IKBT solved the six joint variables, $\theta_1 \ldots \theta_6$ in the following order:

1. θ_1, chosen solver: sinANDcos

$$\theta_{1s1} = \text{atan2}\,(Px, -Py) + \text{atan2}\,(\sqrt{Px^2 + Py^2 - d_3^2}, -d_3)$$
$$\theta_{1s2} = \text{atan2}\,(Px, -Py) + \text{atan2}\,(-\sqrt{Px^2 + Py^2 - d_3^2}, -d_3)$$

2. θ_3, chosen solver: x^2y^2

$$\begin{aligned}
\theta_{3s1} &= \operatorname{atan2}\left(-2a_2d_4, 2a_2a_3\right) + \operatorname{atan2}(\sqrt{s-(t+(Px\cos\left(\theta_{1s1}\right)+Py\sin\left(\theta_{1s1}\right))^2)^2}, \\
&\quad t+(Px\cos\left(\theta_{1s1}\right)+Py\sin\left(\theta_{1s1}\right))^2) \\
\theta_{3s2} &= \operatorname{atan2}\left(-2a_2d_4, 2a_2a_3\right) + \operatorname{atan2}(-\sqrt{s-(t+(Px\cos\left(\theta_{1s1}\right)+Py\sin\left(\theta_{1s1}\right))^2)^2}, \\
&\quad t+(Px\cos\left(\theta_{1s1}\right)+Py\sin\left(\theta_{1s1}\right))^2) \\
\theta_{3s3} &= \operatorname{atan2}\left(-2a_2d_4, 2a_2a_3\right) + \operatorname{atan2}(\sqrt{s-(t+(Px\cos\left(\theta_{1s2}\right)+Py\sin\left(\theta_{1s2}\right))^2)^2}, \\
&\quad t+(Px\cos\left(\theta_{1s2}\right)+Py\sin\left(\theta_{1s2}\right))^2) \\
\theta_{3s4} &= \operatorname{atan2}\left(-2a_2d_4, 2a_2a_3\right) + \operatorname{atan2}(-\sqrt{s-(t+(Px\cos\left(\theta_{1s2}\right)+Py\sin\left(\theta_{1s2}\right))^2)^2}, \\
&\quad t+(Px\cos\left(\theta_{1s2}\right)+Py\sin\left(\theta_{1s2}\right))^2)
\end{aligned}$$

where,

$$\begin{aligned}
s =& 4a_2^2a_3^2 + 4a_2^2d_4^2 \\
t =& Pz^2 - a_2^2 - a_3^2 - d_4^2
\end{aligned}$$

3. θ_{23}, chosen solver: simultaneous equation

$$\begin{aligned}
\theta_{23s1} &= \operatorname{atan2}(Pz(-a_2\cos\left(\theta_{3s3}\right)-a_3) - (-Px\cos\left(\theta_{1s2}\right)-Py\sin\left(\theta_{1s2}\right))(a_2\sin\left(\theta_{3s3}\right)-d_4), \\
&\quad Pz(a_2\sin\left(\theta_{3s3}\right)-d_4) + (-Px\cos\left(\theta_{1s2}\right)-Py\sin\left(\theta_{1s2}\right))(-a_2\cos\left(\theta_{3s3}\right)-a_3)) \\
\theta_{23s2} &= \operatorname{atan2}(Pz(-a_2\cos\left(\theta_{3s2}\right)-a_3) - (-Px\cos\left(\theta_{1s1}\right)-Py\sin\left(\theta_{1s1}\right))(a_2\sin\left(\theta_{3s2}\right)-d_4), \\
&\quad Pz(a_2\sin\left(\theta_{3s2}\right)-d_4) + (-Px\cos\left(\theta_{1s1}\right)-Py\sin\left(\theta_{1s1}\right))(-a_2\cos\left(\theta_{3s2}\right)-a_3)) \\
\theta_{23s3} &= \operatorname{atan2}(Pz(-a_2\cos\left(\theta_{3s1}\right)-a_3) - (-Px\cos\left(\theta_{1s1}\right)-Py\sin\left(\theta_{1s1}\right))(a_2\sin\left(\theta_{3s1}\right)-d_4), \\
&\quad Pz(a_2\sin\left(\theta_{3s1}\right)-d_4) + (-Px\cos\left(\theta_{1s1}\right)-Py\sin\left(\theta_{1s1}\right))(-a_2\cos\left(\theta_{3s1}\right)-a_3)) \\
\theta_{23s4} &= \operatorname{atan2}(Pz(-a_2\cos\left(\theta_{3s4}\right)-a_3) - (-Px\cos\left(\theta_{1s2}\right)-Py\sin\left(\theta_{1s2}\right))(a_2\sin\left(\theta_{3s4}\right)-d_4), \\
&\quad Pz(a_2\sin\left(\theta_{3s4}\right)-d_4) + (-Px\cos\left(\theta_{1s2}\right)-Py\sin\left(\theta_{1s2}\right))(-a_2\cos\left(\theta_{3s4}\right)-a_3))
\end{aligned}$$

4. θ_2, chosen solver: algebraic solver

$$\theta_{2s1} = \theta_{23s2} - \theta_{3s2}$$

$$\theta_{2s2} = \theta_{23s4} - \theta_{3s4}$$

$$\theta_{2s3} = \theta_{23s1} - \theta_{3s3}$$

$$\theta_{2s4} = \theta_{23s3} - \theta_{3s1}$$

5. θ_4, chosen solver: tangent

$$\begin{aligned}
\theta_{4s1} &= \text{atan2}(r_{13}\sin(\theta_{1s1}) - r_{23}\cos(\theta_{1s1}), \\
&\quad r_{13}\cos(\theta_{1s1})\cos(\theta_{23s2}) + r_{23}\sin(\theta_{1s1})\cos(\theta_{23s2}) - r_{33}\sin(\theta_{23s2})) \\
\theta_{4s2} &= \text{atan2}(-r_{13}\sin(\theta_{1s1}) + r_{23}\cos(\theta_{1s1}), \\
&\quad - r_{13}\cos(\theta_{1s1})\cos(\theta_{23s2}) - r_{23}\sin(\theta_{1s1})\cos(\theta_{23s2}) + r_{33}\sin(\theta_{23s2})) \\
\theta_{4s3} &= \text{atan2}(r_{13}\sin(\theta_{1s2}) - r_{23}\cos(\theta_{1s2}), \\
&\quad r_{13}\cos(\theta_{1s2})\cos(\theta_{23s4}) + r_{23}\sin(\theta_{1s2})\cos(\theta_{23s4}) - r_{33}\sin(\theta_{23s4})) \\
\theta_{4s4} &= \text{atan2}(-r_{13}\sin(\theta_{1s2}) + r_{23}\cos(\theta_{1s2}), \\
&\quad - r_{13}\cos(\theta_{1s2})\cos(\theta_{23s4}) - r_{23}\sin(\theta_{1s2})\cos(\theta_{23s4}) + r_{33}\sin(\theta_{23s4})) \\
\theta_{4s5} &= \text{atan2}(r_{13}\sin(\theta_{1s2}) - r_{23}\cos(\theta_{1s2}), \\
&\quad r_{13}\cos(\theta_{1s2})\cos(\theta_{23s1}) + r_{23}\sin(\theta_{1s2})\cos(\theta_{23s1}) - r_{33}\sin(\theta_{23s1})) \\
\theta_{4s6} &= \text{atan2}(-r_{13}\sin(\theta_{1s2}) + r_{23}\cos(\theta_{1s2}), \\
&\quad - r_{13}\cos(\theta_{1s2})\cos(\theta_{23s1}) - r_{23}\sin(\theta_{1s2})\cos(\theta_{23s1}) + r_{33}\sin(\theta_{23s1})) \\
\theta_{4s7} &= \text{atan2}(r_{13}\sin(\theta_{1s1}) - r_{23}\cos(\theta_{1s1}), \\
&\quad r_{13}\cos(\theta_{1s1})\cos(\theta_{23s3}) + r_{23}\sin(\theta_{1s1})\cos(\theta_{23s3}) - r_{33}\sin(\theta_{23s3})) \\
\theta_{4s8} &= \text{atan2}(-r_{13}\sin(\theta_{1s1}) + r_{23}\cos(\theta_{1s1}), \\
&\quad - r_{13}\cos(\theta_{1s1})\cos(\theta_{23s3}) - r_{23}\sin(\theta_{1s1})\cos(\theta_{23s3}) + r_{33}\sin(\theta_{23s3}))
\end{aligned}$$

6. θ_5, chosen solver: tangent

$$\theta_{5s1} = \text{atan2}(\frac{1}{\sin(\theta_{4s3})}(-r_{13}\sin(\theta_{1s2}) + r_{23}\cos(\theta_{1s2})),$$
$$- r_{13}\sin(\theta_{23s4})\cos(\theta_{1s2}) - r_{23}\sin(\theta_{1s2})\sin(\theta_{23s4}) - r_{33}\cos(\theta_{23s4})$$

$$\theta_{5s2} = \text{atan2}(\frac{1}{\sin(\theta_{4s7})}(-r_{13}\sin(\theta_{1s1}) + r_{23}\cos(\theta_{1s1})),$$
$$- r_{13}\sin(\theta_{23s3})\cos(\theta_{1s1}) - r_{23}\sin(\theta_{1s1})\sin(\theta_{23s3}) - r_{33}\cos(\theta_{23s3}))$$

$$\theta_{5s3} = \text{atan2}(\frac{1}{\sin(\theta_{4s2})}(-r_{13}\sin(\theta_{1s1}) + r_{23}\cos(\theta_{1s1})),$$
$$- r_{13}\sin(\theta_{23s2})\cos(\theta_{1s1}) - r_{23}\sin(\theta_{1s1})\sin(\theta_{23s2}) - r_{33}\cos(\theta_{23s2}))$$

$$\theta_{5s4} = \text{atan2}(\frac{1}{\sin(\theta_{4s6})}(-r_{13}\sin(\theta_{1s2}) + r_{23}\cos(\theta_{1s2})),$$
$$- r_{13}\sin(\theta_{23s1})\cos(\theta_{1s2}) - r_{23}\sin(\theta_{1s2})\sin(\theta_{23s1}) - r_{33}\cos(\theta_{23s1}))$$

$$\theta_{5s5} = \text{atan2}(\frac{1}{\sin(\theta_{4s4})}(-r_{13}\sin(\theta_{1s2}) + r_{23}\cos(\theta_{1s2})),$$
$$- r_{13}\sin(\theta_{23s4})\cos(\theta_{1s2}) - r_{23}\sin(\theta_{1s2})\sin(\theta_{23s4}) - r_{33}\cos(\theta_{23s4}))$$

$$\theta_{5s6} = \text{atan2}(\frac{1}{\sin(\theta_{4s1})}(-r_{13}\sin(\theta_{1s1}) + r_{23}\cos(\theta_{1s1})),$$
$$- r_{13}\sin(\theta_{23s2})\cos(\theta_{1s1}) - r_{23}\sin(\theta_{1s1})\sin(\theta_{23s2}) - r_{33}\cos(\theta_{23s2}))$$

$$\theta_{5s7} = \text{atan2}(\frac{1}{\sin(\theta_{4s8})}(-r_{13}\sin(\theta_{1s1}) + r_{23}\cos(\theta_{1s1})),$$
$$-r_{13}\sin(\theta_{23s3})\cos(\theta_{1s1}) - r_{23}\sin(\theta_{1s1})\sin(\theta_{23s3}) - r_{33}\cos(\theta_{23s3}))$$

$$\theta_{5s8} = \text{atan2}(\frac{1}{\sin(\theta_{4s5})}(-r_{13}\sin(\theta_{1s2}) + r_{23}\cos(\theta_{1s2})),$$
$$-r_{13}\sin(\theta_{23s1})\cos(\theta_{1s2}) - r_{23}\sin(\theta_{1s2})\sin(\theta_{23s1}) - r_{33}\cos(\theta_{23s1}))$$

7. θ_6, chosen solver: tangent

$$\theta_{6s1} = \text{atan2}(-\frac{-r_{12}\sin(\theta_{23s1})\cos(\theta_{1s2}) - r_{22}\sin(\theta_{1s2})\sin(\theta_{23s1}) - r_{32}\cos(\theta_{23s1})}{\sin(\theta_{5s4})},$$
$$\frac{-r_{11}\sin(\theta_{23s1})\cos(\theta_{1s2}) - r_{21}\sin(\theta_{1s2})\sin(\theta_{23s1}) - r_{31}\cos(\theta_{23s1})}{\sin(\theta_{5s4})})$$
$$\theta_{6s2} = \text{atan2}(-\frac{-r_{12}\sin(\theta_{23s1})\cos(\theta_{1s2}) - r_{22}\sin(\theta_{1s2})\sin(\theta_{23s1}) - r_{32}\cos(\theta_{23s1})}{\sin(\theta_{5s8})},$$
$$\frac{-r_{11}\sin(\theta_{23s1})\cos(\theta_{1s2}) - r_{21}\sin(\theta_{1s2})\sin(\theta_{23s1}) - r_{31}\cos(\theta_{23s1})}{\sin(\theta_{5s8})})$$
$$\theta_{6s3} = \text{atan2}(-\frac{-r_{12}\sin(\theta_{23s4})\cos(\theta_{1s2}) - r_{22}\sin(\theta_{1s2})\sin(\theta_{23s4}) - r_{32}\cos(\theta_{23s4})}{\sin(\theta_{5s1})},$$
$$\frac{-r_{11}\sin(\theta_{23s4})\cos(\theta_{1s2}) - r_{21}\sin(\theta_{1s2})\sin(\theta_{23s4}) - r_{31}\cos(\theta_{23s4})}{\sin(\theta_{5s1})})$$
$$\theta_{6s4} = \text{atan2}(-\frac{-r_{12}\sin(\theta_{23s3})\cos(\theta_{1s1}) - r_{22}\sin(\theta_{1s1})\sin(\theta_{23s3}) - r_{32}\cos(\theta_{23s3})}{\sin(\theta_{5s2})},$$
$$\frac{-r_{11}\sin(\theta_{23s3})\cos(\theta_{1s1}) - r_{21}\sin(\theta_{1s1})\sin(\theta_{23s3}) - r_{31}\cos(\theta_{23s3})}{\sin(\theta_{5s2})})$$

$$\theta_{6s5} = \text{atan2}(- \frac{-r_{12}\sin(\theta_{23s2})\cos(\theta_{1s1}) - r_{22}\sin(\theta_{1s1})\sin(\theta_{23s2}) - r_{32}\cos(\theta_{23s2})}{\sin(\theta_{5s6})}, \frac{-r_{11}\sin(\theta_{23s2})\cos(\theta_{1s1}) - r_{21}\sin(\theta_{1s1})\sin(\theta_{23s2}) - r_{31}\cos(\theta_{23s2})}{\sin(\theta_{5s6})})$$

$$\theta_{6s6} = \text{atan2}(- \frac{-r_{12}\sin(\theta_{23s2})\cos(\theta_{1s1}) - r_{22}\sin(\theta_{1s1})\sin(\theta_{23s2}) - r_{32}\cos(\theta_{23s2})}{\sin(\theta_{5s3})}, \frac{-r_{11}\sin(\theta_{23s2})\cos(\theta_{1s1}) - r_{21}\sin(\theta_{1s1})\sin(\theta_{23s2}) - r_{31}\cos(\theta_{23s2})}{\sin(\theta_{5s3})})$$

$$\theta_{6s7} = \text{atan2}(- \frac{-r_{12}\sin(\theta_{23s3})\cos(\theta_{1s1}) - r_{22}\sin(\theta_{1s1})\sin(\theta_{23s3}) - r_{32}\cos(\theta_{23s3})}{\sin(\theta_{5s7})}, \frac{-r_{11}\sin(\theta_{23s3})\cos(\theta_{1s1}) - r_{21}\sin(\theta_{1s1})\sin(\theta_{23s3}) - r_{31}\cos(\theta_{23s3})}{\sin(\theta_{5s7})})$$

$$\theta_{6s8} = \text{atan2}(- \frac{-r_{12}\sin(\theta_{23s4})\cos(\theta_{1s2}) - r_{22}\sin(\theta_{1s2})\sin(\theta_{23s4}) - r_{32}\cos(\theta_{23s4})}{\sin(\theta_{5s5})}, \frac{-r_{11}\sin(\theta_{23s4})\cos(\theta_{1s2}) - r_{21}\sin(\theta_{1s2})\sin(\theta_{23s4}) - r_{31}\cos(\theta_{23s4})}{\sin(\theta_{5s5})})$$

IKBT can find all 8 positions of PUMA 560, and the solution graph shows the dependency among variables in Fig. 2.5. These joint solutions are then grouped into sets corresponding to correct poses:

$$Pose \quad 1 : [\theta_{1s1}, \theta_{2s4}, \theta_{3s1}, \theta_{4s8}, \theta_{5s7}, \theta_{6s7}]$$

$$Pose \quad 2 : [\theta_{1s2}, \theta_{2s2}, \theta_{3s4}, \theta_{4s4}, \theta_{5s5}, \theta_{6s8}]$$

$$Pose \quad 3 : [\theta_{1s1}, \theta_{2s1}, \theta_{3s2}, \theta_{4s2}, \theta_{5s3}, \theta_{6s6}]$$

$$Pose \quad 4 : [\theta_{1s2}, \theta_{2s3}, \theta_{3s3}, \theta_{4s6}, \theta_{5s4}, \theta_{6s1}]$$

$$Pose \quad 5 : [\theta_{1s2}, \theta_{2s3}, \theta_{3s3}, \theta_{4s5}, \theta_{5s8}, \theta_{6s2}]$$

$$Pose \quad 6 : [\theta_{1s1}, \theta_{2s1}, \theta_{3s2}, \theta_{4s1}, \theta_{5s6}, \theta_{6s5}]$$

$$Pose \quad 7 : [\theta_{1s1}, \theta_{2s4}, \theta_{3s1}, \theta_{4s7}, \theta_{5s2}, \theta_{6s4}]$$

$$Pose \quad 8 : [\theta_{1s2}, \theta_{2s2}, \theta_{3s4}, \theta_{4s3}, \theta_{5s1}, \theta_{6s3}]$$

Result Verification

To verify the solution, we followed the process stated in section 2.5.1. The starting pose used is:

$$\theta_1 = 30^\circ, \theta_2 = 50^\circ, \theta_3 = 40^\circ,$$

$$\theta_4 = 45^\circ, \theta_5 = 120^\circ, \theta_6 = 60^\circ$$

as well as the parameters:

$$a_2 = 5, a_3 = 1, d_3 = 2, d_4 = 4$$

From the DH parameters, the Forward kinematics code generated by IKBT generated the numerical T matrix:

$$T_d = \begin{bmatrix} -0.15720 & 0.97938 & 0.12682 & -1.68074 \\ -0.59374 & -0.19635 & 0.78032 & 1.33902 \\ 0.78914 & 0.04737 & 0.61237 & -4.83022 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We then plugged T_d into symbolic solutions obtained from the last section, and got the joint poses listed in Table 2.3. Note that Pose 7 (Table 2.3) is the same as initial input pose, within 10^{-4}.

With all the numerical poses, we computed forward kinematics. The T matrix computed from Pose 1 (T_{p1} is selected as an example, since it showed the largest variation compared to the original T matrix:

$$T_{p1} = \begin{bmatrix} -0.15720 & 0.97939 & 0.12682 & -1.68075 \\ -0.59374 & -0.19634 & 0.78033 & 1.33902 \\ 0.78915 & 0.04737 & 0.61237 & -4.83025 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We got the same values compared to the original T matrix for all solution poses, with differences $\approx 10^{-5}$. This result unequivocally proves that IKBT's symbolic inverse kinematics analysis is correct.

Solution Poses						
Pose	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6
1	-287.08771	130.00008	-191.92745	-6.78054	-66.24462	4.13687
2	29.99995	49.99992	39.99994	-135.00007	-119.99981	-120.00010
3	29.99995	148.48625	-191.92745	142.01103	95.78709	-151.06756
4	-287.08771	31.51375	39.99994	18.00641	159.53838	18.33206
5	-287.08771	130.00008	-191.92745	173.21946	66.24462	-175.86313
6	29.99995	148.48625	-191.92745	-37.98897	-95.78709	28.93244
7	29.99995	49.99992	39.99994	44.99993	119.99981	59.99990
8	-287.08771	31.51375	39.99994	-161.99359	-159.53838	-161.66794

Table 2.3: **Puma 560 Numerical Solutions**

Note: these tests shown here is only a proof-of-principle. Extensive tests were carried out with numerous test values. Validity of end effector poses can be screened by pose validation method built-in IKBT, details see section Verification and Validation.

2.6.3 Example Solutions - Robot without 3 Intersecting Axes

Previous software packages which perform inverse kinematics analysis usually require the robot to have three intersecting axes (such as the popular ROS package). To demonstrate IKBT's flexibility in handling robots with different configurations. We select the example of "Chair Helper", a 5 DOF robot without three intersecting axes (Table 2.4)[2].

Forward kinematics:

[2]Thanks to Prof. Melanie Shoemaker Plett.

Table 2.4: Chair Helper DH parameters

Link	α_{N-1}	a_{N-1}	d_N	θ_N
1	0	0	d_1	0
2	0	l_1	0	θ_2
3	$\pi/2$	0	l_2	θ_3
4	$\pi/2$	0	0	θ_4
5	- $\pi/2$	0	l_4	θ_5

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & Px \\ r_{21} & r_{22} & r_{23} & Py \\ r_{31} & r_{32} & r_{33} & Pz \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} v_1 & v_2 & v_3 & v_4 \end{bmatrix}$$

$$v_1 = \begin{bmatrix} -c_2 s_3 s_5 + c_5(c_2 c_3 c_4 + s_2 s_4) \\ c_5(-c_2 s_4 + c_3 c_4 s_2) - s_2 s_3 s_5 \\ c_3 s_5 + c_4 c_5 s_3 \\ 0 \end{bmatrix} \qquad v_2 = \begin{bmatrix} -c_2 c_5 s_3 - s_5(c_2 c_3 c_4 + s_2 s_4) \\ -c_5 s_2 s_3 - s_5(-c_2 s_4 + c_3 c_4 s_2) \\ c_3 c_5 - c_4 s_3 s_5 \\ 0 \end{bmatrix}$$

$$v_3 = \begin{bmatrix} -c_2 c_3 s_4 + c_4 s_2 \\ -c_2 c_4 - c_3 s_2 s_4 \\ -s_3 s_4 \\ 0 \end{bmatrix} \qquad v_4 = \begin{bmatrix} l_1 + l_2 s_2 + l_4(-c_2 c_3 s_4 + c_4 s_2) \\ -c_2 c_4 l_4 - c_2 l_2 - c_3 l_4 s_2 s_4 \\ d_1 - l_4 s_3 s_4 \\ 1 \end{bmatrix}$$

IKBT gave the inverse kinematics solutions:

1. d_1, chosen solver: algebra

$$d_1 = Pz - l_4 r_{33}$$

2. θ_2, chosen solver: sine or cosine

$$\theta_2 = \text{atan2}\,(\frac{1}{l_2}(Px - l_1 - l_4 r_{13}), -\frac{1}{l_2}(Py - l_4 r_{23}))$$

3. θ_3, chosen solver: tangent

$$\theta_{3s1} = \text{atan2}\,(r_{33}, r_{13}\cos(\theta_2) + r_{23}\sin(\theta_2))$$
$$\theta_{3s2} = \text{atan2}\,(-r_{33}, -r_{13}\cos(\theta_2) - r_{23}\sin(\theta_2))$$

4. θ_4, chosen solver: tangent

$$\theta_{4s1} = \text{atan2}\,(-\frac{r_{33}}{\sin(\theta_{3s2})}, r_{13}\sin(\theta_2) - r_{23}\cos(\theta_2))$$
$$\theta_{4s2} = \text{atan2}\,(-\frac{r_{33}}{\sin(\theta_{3s1})}, r_{13}\sin(\theta_2) - r_{23}\cos(\theta_2))$$

5. θ_5, chosen solver: tangent

$$\theta_{5s1} = \text{atan2}\,(\frac{1}{\sin(\theta_{4s1})}(-r_{12}\sin(\theta_2) + r_{22}\cos(\theta_2)), \frac{1}{\sin(\theta_{4s1})}(r_{11}\sin(\theta_2) - r_{21}\cos(\theta_2)))$$
$$\theta_{5s2} = \text{atan2}\,(\frac{1}{\sin(\theta_{4s2})}(-r_{12}\sin(\theta_2) + r_{22}\cos(\theta_2)), \frac{1}{\sin(\theta_{4s2})}(r_{11}\sin(\theta_2) - r_{21}\cos(\theta_2)))$$

Solutions sets:

$$[d_1, \theta_2, \theta_{3s1}, \theta_{4s2}, \theta_{5s2}]$$
$$[d_1, \theta_2, \theta_{3s2}, \theta_{4s1}, \theta_{5s1}]$$

Numerical verification confirmed that the inverse kinematics solutions are correct.

2.6.4 Example Solutions - Robot with Strictly Solution Graph

The following example (Olson13) illustrates the necessity of a graph when tracking dependency, where variables have two independent parent variables, as shown in 2.6. DH parameters are listed in Table 2.5. Olson13 is a 6-DOF robot. The unknown variables are: $[d_1 \quad d_2 \quad \theta_3 \quad \theta_4 \quad \theta_5 \quad \theta_6]$. The known parameters are: $[l_3 \quad l_4 \quad l_5]$

Table 2.5: Olson13 DH parameters

Link	α_{N-1}	a_{N-1}	d_N	θ_N
1	$-\pi/2$	0	d_1	$\pi/2$
2	$\pi/2$	0	d_2	$-\pi/2$
3	$\pi/2$	0	l_3	θ_3
4	$\pi/2$	0	0	θ_4
5	0	l_4	0	θ_5
6	$\pi/2$	0	l_5	θ_6

Forward kinematics:

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & Px \\ r_{21} & r_{22} & r_{23} & Py \\ r_{31} & r_{32} & r_{33} & Pz \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} v_1 & v_2 & v_3 & v_4 \end{bmatrix}$$

$$\begin{bmatrix} v_1 & v_2 \end{bmatrix} = \begin{bmatrix} -c_3 s_6 + c_{45} c_6 s_3 & -c_3 c_6 - c_{45} s_3 s_6 \\ -c_3 c_{45} c_6 - s_3 s_6 & c_3 c_{45} s_6 - c_6 s_3 \\ c_6 s_{45} & -s_{45} s_6 \\ 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} v_3 & v_4 \end{bmatrix} = \begin{bmatrix} s_3 s_{45} & c_4 l_4 s_3 + d_2 + l_5 s_3 s_{45} \\ -c_3 s_{45} & -c_3 c_4 l_4 - c_3 l_5 s_{45} + d_1 \\ -c_{45} & -c_{45} l_5 + l_3 + l_4 s_4 \\ 0 & 1 \end{bmatrix}$$

The variables are solved in the following order:

1. θ_3, chosen solver: tangent

$$\theta_{3s1} = \text{atan2}\left(-r_{13}, r_{23}\right)$$
$$\theta_{3s2} = \text{atan2}\left(r_{13}, -r_{23}\right)$$

2. θ_4, chosen solver: sine or cosine

$$\theta_{4s1} = \text{asin}\left(\frac{1}{l_4}\left(Pz - l_3 - l_5 r_{33}\right)\right)$$
$$\theta_{4s2} = -\text{asin}\left(\frac{1}{l_4}\left(Pz - l_3 - l_5 r_{33}\right)\right) + \pi$$

3. θ_5, chosen solver: tangent

$$\begin{aligned}
\theta_{5s1} = \text{atan2}(&r_{13}\sin\left(\theta_{3s2}\right)\cos\left(\theta_{4s1}\right) - r_{23}\cos\left(\theta_{3s2}\right)\cos\left(\theta_{4s1}\right) + r_{33}\sin\left(\theta_{4s1}\right),\\
&r_{13}\sin\left(\theta_{3s2}\right)\sin\left(\theta_{4s1}\right) - r_{23}\sin\left(\theta_{4s1}\right)\cos\left(\theta_{3s2}\right) - r_{33}\cos\left(\theta_{4s1}\right))\\
\theta_{5s2} = \text{atan2}(&r_{13}\sin\left(\theta_{3s1}\right)\cos\left(\theta_{4s1}\right) - r_{23}\cos\left(\theta_{3s1}\right)\cos\left(\theta_{4s1}\right) + r_{33}\sin\left(\theta_{4s1}\right),\\
&r_{13}\sin\left(\theta_{3s1}\right)\sin\left(\theta_{4s1}\right) - r_{23}\sin\left(\theta_{4s1}\right)\cos\left(\theta_{3s1}\right) - r_{33}\cos\left(\theta_{4s1}\right))\\
\theta_{5s3} = \text{atan2}(&r_{13}\sin\left(\theta_{3s2}\right)\cos\left(\theta_{4s2}\right) - r_{23}\cos\left(\theta_{3s2}\right)\cos\left(\theta_{4s2}\right) + r_{33}\sin\left(\theta_{4s2}\right),\\
&r_{13}\sin\left(\theta_{3s2}\right)\sin\left(\theta_{4s2}\right) - r_{23}\sin\left(\theta_{4s2}\right)\cos\left(\theta_{3s2}\right) - r_{33}\cos\left(\theta_{4s2}\right))\\
\theta_{5s4} = \text{atan2}(&r_{13}\sin\left(\theta_{3s1}\right)\cos\left(\theta_{4s2}\right) - r_{23}\cos\left(\theta_{3s1}\right)\cos\left(\theta_{4s2}\right) + r_{33}\sin\left(\theta_{4s2}\right),\\
&r_{13}\sin\left(\theta_{3s1}\right)\sin\left(\theta_{4s2}\right) - r_{23}\sin\left(\theta_{4s2}\right)\cos\left(\theta_{3s1}\right) - r_{33}\cos\left(\theta_{4s2}\right))
\end{aligned}$$

4. θ_6, chosen solver: tangent

$$\theta_{6s1} = \text{atan2}(-r_{11}\cos\left(\theta_{3s2}\right) - r_{21}\sin\left(\theta_{3s2}\right), -r_{12}\cos\left(\theta_{3s2}\right) - r_{22}\sin\left(\theta_{3s2}\right))$$
$$\theta_{6s2} = \text{atan2}(-r_{11}\cos\left(\theta_{3s1}\right) - r_{21}\sin\left(\theta_{3s1}\right), -r_{12}\cos\left(\theta_{3s1}\right) - r_{22}\sin\left(\theta_{3s1}\right))$$

5. d_1, chosen solver: algebra

$$d_{1s1} = Py + l_4 \cos(\theta_{3s2}) \cos(\theta_{4s1}) - l_5 r_{23}$$
$$d_{1s2} = Py + l_4 \cos(\theta_{3s1}) \cos(\theta_{4s1}) - l_5 r_{23}$$
$$d_{1s3} = Py + l_4 \cos(\theta_{3s2}) \cos(\theta_{4s2}) - l_5 r_{23}$$
$$d_{1s4} = Py + l_4 \cos(\theta_{3s1}) \cos(\theta_{4s2}) - l_5 r_{23}$$

6. d_2, chosen solver: algebra

$$d_{2s1} = Px - l_4 \sin(\theta_{3s2}) \cos(\theta_{4s1}) - l_5 r_{13}$$
$$d_{2s2} = Px - l_4 \sin(\theta_{3s1}) \cos(\theta_{4s1}) - l_5 r_{13}$$
$$d_{2s3} = Px - l_4 \sin(\theta_{3s2}) \cos(\theta_{4s2}) - l_5 r_{13}$$
$$d_{2s4} = Px - l_4 \sin(\theta_{3s1}) \cos(\theta_{4s2}) - l_5 r_{13}$$

The following are the sets of joint solutions (poses) for this manipulator:

$$[d_{1s3}, d_{2s3}, \theta_{3s2}, \theta_{4s2}, \theta_{5s3}, \theta_{6s1}]$$
$$[d_{1s4}, d_{2s4}, \theta_{3s1}, \theta_{4s2}, \theta_{5s4}, \theta_{6s2}]$$
$$[d_{1s1}, d_{2s1}, \theta_{3s2}, \theta_{4s1}, \theta_{5s1}, \theta_{6s1}]$$
$$[d_{1s2}, d_{2s2}, \theta_{3s1}, \theta_{4s1}, \theta_{5s2}, \theta_{6s2}]$$

The solution graph is shown in Fig. 2.6. d_1, d_2, and θ_5 all share two independently solved parent variables: θ_3 and θ_4. Thus, it results in a dependency graph.

2.7 Discussion

Building on previous research, IKBT has several advantages including applicability to any robot (up to 6-DOF), generalized solving scheme, extensible toolbox and solving logic, mod-

ern and easy to implement language (Python), and dependencies limited to only a few libraries. We expect these characteristics will spur wide adoption of IKBT into the robotics research and education communities.

We reiterate the point of analytical inverse kinematics solutions is superior than numerical ones from introduction : numerical solvers often fails (unable to converge) due to dependency on specific numerical values, even when the inverse kinematics solutions exist - analytical solutions circumvent such dependency. To clarify, there are no numerical methods (with their attending difficulties) in our system, and purely symbolic reasoning is used. Historically, such reasoning has been proven difficult even when handled by human experts - to an extent that many commercial robots were designed with special configurations (such as three intersecting axes) to facilitate the solving process of closed-form inverse kinematics. IKBT, the automatic closed-form inverse kinematic solver, brings the possibility of broadening robot designs.

The rule-based solvers included in IKBT's toolbox are commonly employed by human experts when solving inverse kinematics problems. This is advantageous because IKBT solution methods are relateable to manual methods, and its approach is not limited by robot configuration. Specifically, it doesn't require three orthogonal axes in order to solve a robot. One noteworthy characteristic about IKBT is that there is no upper limit for the number of solutions. As long as the solutions are valid and non-repetitive, IKBT can find all possible symbolic solutions for a robot. This is also due to the nature of IKBT - it is rule-based and not dependent on specific robot configuration.

IKBT's Behavior Tree represents an interpretable strategy - vital for judging many AI applications. This makes it easier to examine the correctness of the solution and the strategy formulating process. Although IKBT's approach costs more computing time than DOF-specific algorithms (4 ms, according to [17]), symbolic derivation only has to be done once per robot arm design. The Behavior Tree is easily modified and the solver toolbox is readily extensible. Although all the results presented here were generated by the BT of Fig. 2.2, it may be the case that a custom Behavior Tree could solve additional robots or solve robots more efficiently.

The Behavior Tree representation has gained success in game AI [45, 30], and showed substantial possibilities in robotics research [47, 41, 10, 28]. IKBT serves as a proof-of-concept of solving high-cognitive problems with Behavior Trees. IKBT mimics human experts' logical reasoning process, and constructs a generalized solving scheme applicable to an entire class of problems, using a small number of knowledge leaves. While most of current AI work focuses on recognizing and understanding scenarios, Behavior Trees emerge as a path to an equally vital component - combinatorial reasoning. Such logical reasoning enables AI agents to use limited knowledge base to solve problems of much larger magnitude. Intuitively, combinatorial reasoning is how human interact with the world: we decompose a unseen problem into solvable parts, then piece together modular knowledge to solve the larger problem. We don't get re-trained from scratch whenever we face new problems.

All knowledge-based solvers in IKBT are coded by us, in other words, we "teach" the system pre-existing rules and tricks people have used when solving inverse kinematics problems. Moving forward, the system can learn the knowledge by itself, by observing patterns and understanding their meaning. Learning in Behavior Trees has been explored in preliminary efforts[37, 15, 8, 23]. Combining forces of learning (especially unsupervised) and combinatorial reasoning, we might be on our way to unlock the next level of autonomy.

Acknowledgement

We thank Dr. Steve Tanimoto for the invaluable advice on identifying the right audience and publication venue. We gratefully acknowledge support from National Science Foundation grant IIS-1637444 and support for Blake Hannaford at Google-X / Google Life-Sciences / Verily in 2015. We thank Moshe Lutz for proofreading this article.

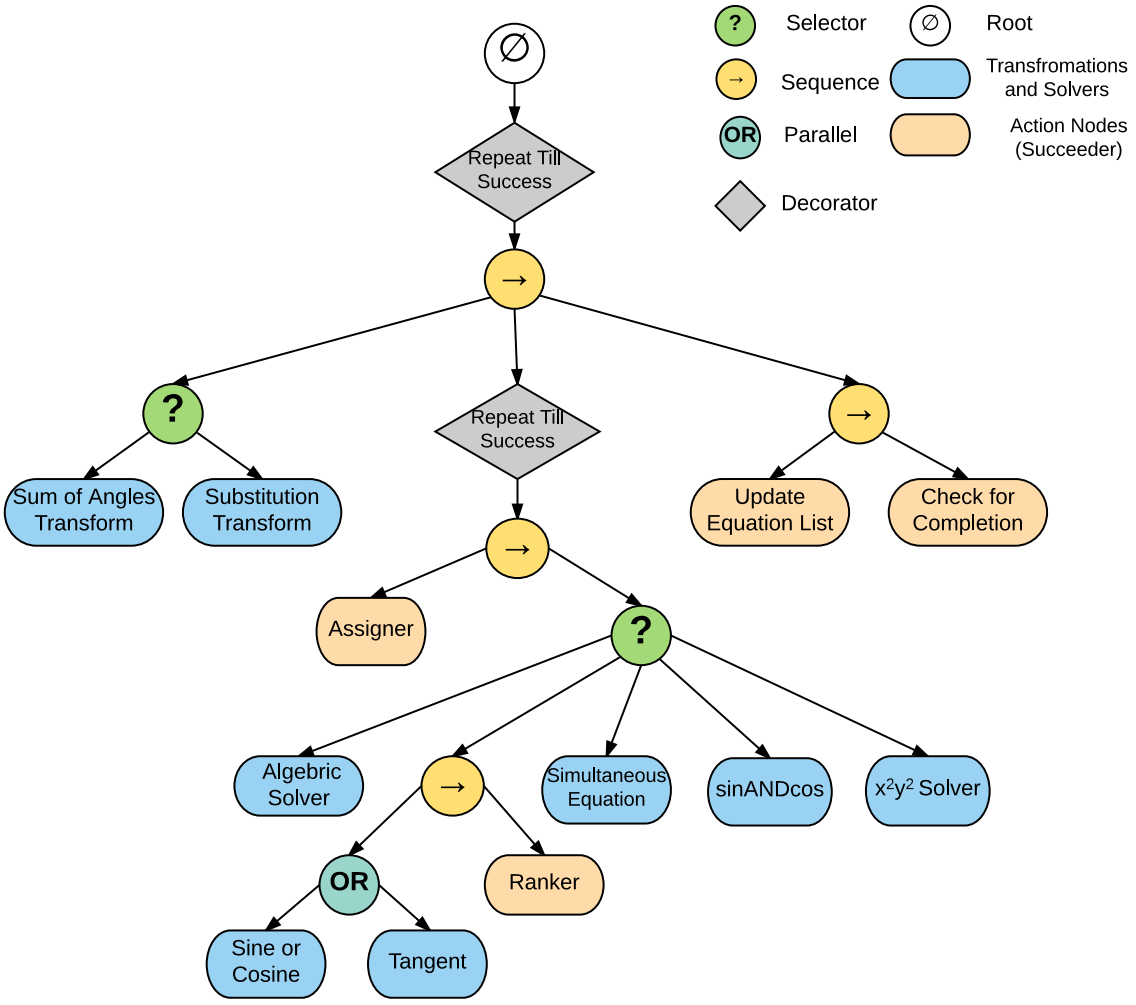


Figure 2.2: **IKBT Structure.** Node type explanation: Action nodes (leaves) carry out specific tasks, and returns SUCCESS or FAILURE. Succeeder is a special type of action nodes that only returns SUCCESS. Selector node ticks its children in turn, returns SUCCESS and stops if one of the children succeeds, otherwise returns FAILURE. Sequence node only returns SUCCESS if all its children succeed. Parallel node tries out all its children regardless of their return status, returns SUCCESS if any child succeeds.

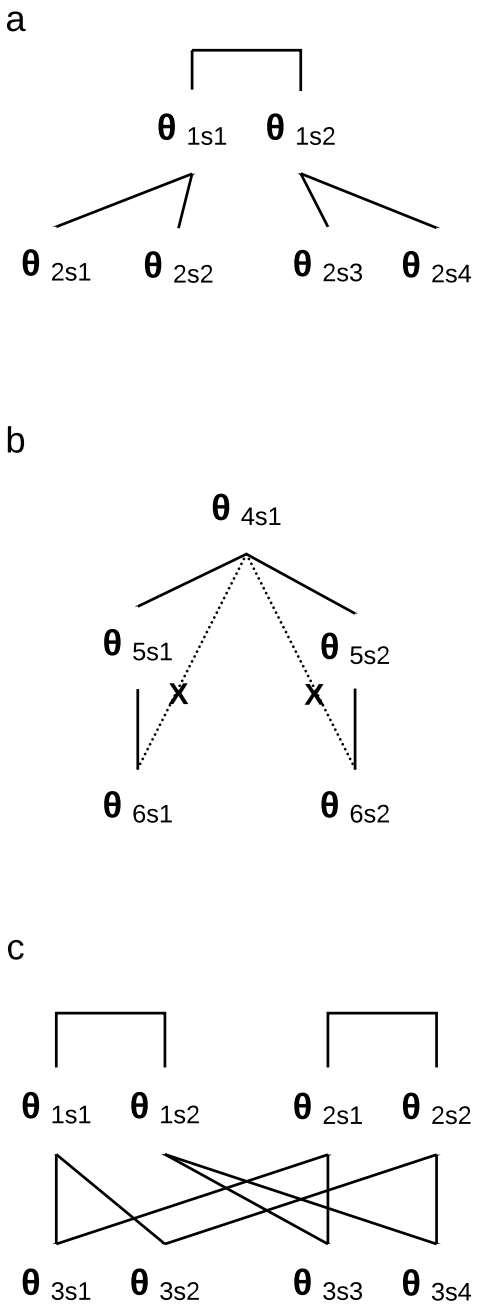


Figure 2.3: **Sample Solution Graph.** a) Simple example solution graph explains the origins of multiplicity. b) Redundancy pruning, 'x' marks the dependent relations that are not included in the graph. c) Example of a case of variables with multiple independent parents.

Figure 2.4: **Result Verification.** A numerical 4x4 homogeneous transformation matrix, T_d, is constructed from a reachable pose. Numerical joint space poses are computed from T_d using the closed-form solutions. For each solution pose, forward kinematics is calculated. The resulting transform matrices are compared against the original matrix, matching value is indicative of correct IKBT inverse kinematics analysis. 2^N indicates the number of solutions is always even.

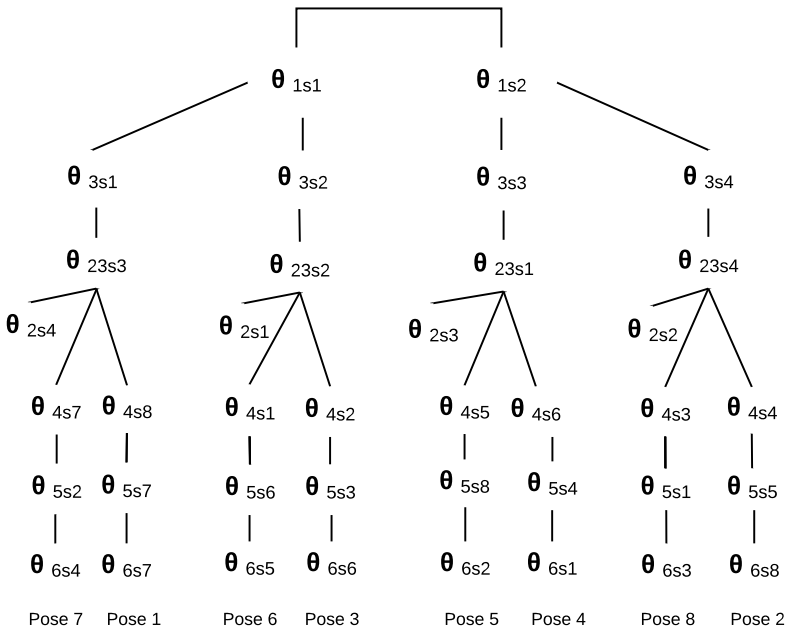


Figure 2.5: **PUMA 560 Solution Graph.**

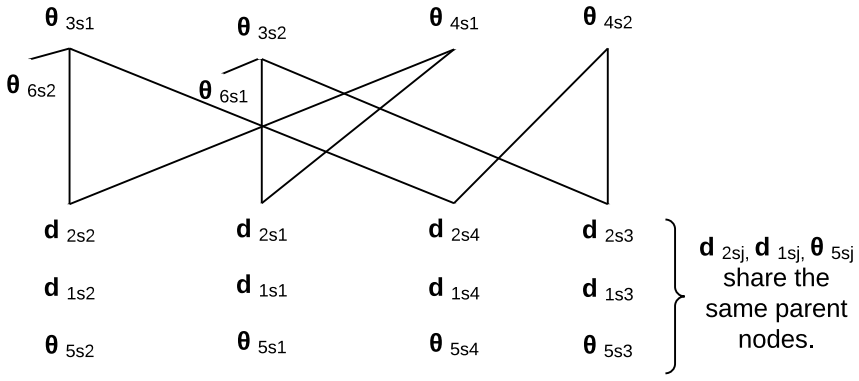


Figure 2.6: **Olson13 Solution Graph.**

Chapter 3

BEHAVIOR TREES FOR EFFICIENT HIERARCHICAL REINFORCEMENT LEARNING

In the previous chapter, an intricate behavior tree was designed and built manually to solve a complex problem, in a intuitive and human interpretable manner. The building process of the said behavior tree (IKBT) relies entirely on traditional programming - human experts carefully encode domain-specific knowledge or algorithms that requires no additional data to solve a problem instance. In this chapter, I explore the domain of combining the behavior tree and reinforcement learning, and demonstrate the advantage of such combination.

With the help of reinforcement learning, a behavior tree can have the learning capability. Similar to reinforcement learning, a learn-able behavior tree each action is defined, what can be learned is that how to apply different actions for different states.

The advantage of using behavior tree in learning is significant when solving hierarchical problems. In hierarchical problems, a high-level goal can be breaken up into subtasks and subproblems with independent sub-goals, as if they were primitive actions. This high-level goal and subproblems forms hierarchy. For one problem, it can have multi-levels of hierarchy. An example of hierarchical problem is cooking. The first level of hierarchy consists of cutting ingredients and cook ingredients. The second level of hierarchy for cutting ingredients is a) get a knife, and then b) cut ingredients to desired size and shape. The subproblems for cooking might be two equivalent choices, the chef can choose to a) saute, b) steam, or c) boil the ingredients.

3.1 Introduction

Long horizon, sequential tasks with sparse rewards have been challenging to solve. There are ample examples of such problems in daily life, for example fixing a bike. It involves getting the bike, get the necessary tools and parts, then fix the bike. The reward is for achieving the overall goal, and for us, we have the common sense to get the bike, tools and parts before the action of fixing. But for an AI agent, that learns from the scratch, such common sense knowledge is not a given. One of the characteristics that impede efficient learning in a non-hierarchical setting is that usually only a subset of actions are allowed at any given state, but the agent is not aware of such limitations. This causes inefficient exploring, where the agent attempts many actions that aren't feasible for current state, and categorize them as "bad choices". Even though those actions are critical to complete the overall goal at later stage, the agent already learned too much negatives about those actions, and unlikely to choose them. This inefficiency is even more obvious when there is a step limit for each episode - the agent might waste too many trials on the unfeasible actions, and long before the agent can learn about the preferable actions the episode ends.

Hierarchical reinforcement learning [4] as a field focuses on such problems. MAXQ method [18], options framework and related option algorithms[58, 35], and hDQN [36] share the common theme of temporal abstraction to achieve the goal of "divide and conquer". Instead of tackling the entire task all at once, these agents break the task into more manageable subtasks. Behavior tree's modular and composite structure is suitable for situations like this, where it can easily transfer knowledge among different sub-goals. Here we propose to leverage the modality of behavior trees to augment reinforcement learning (here double DQN, specifically) to be more efficient in solving long horizon, sparse rewards tasks. Although the behavior tree and reinforcement learning was studied by as few [16, 63], however, due to the problem studied was relatively simple (243 states), short term goal with abundant rewards signals, plus that the implementation is not leveraging modularity of the behavior trees, their results were inconclusive in proving the advantages of using behavior tree to

augment reinforcement learning. BT-RL provides a problem formulation that decomposes complex problem goal into sub-trees with respective sub-goals, and only permits feasible actions. This in turn encourages effective exploring. BT-RL leverages the modularity of behavior tree, so that the agent can easily re-use acquired knowledge in the same domain for different sub-goals.

Our contributions are:

1. Proposed a new learning paradigm with behavior tree augmented reinforcement learning (BT-RL), with trainable leaf nodes.

2. Demonstrated that BT-RL is much more efficient in handling complex tasks (long horizon, sequential, sparse rewards), compared to reinforcement learning alone, in varies difficulties of tasks.

3. Proved for simple tasks (not long horizon, with abundant rewards signal), such advantage is not obvious.

4. Implemented a new testing benchmark, Pandemic the board game.

3.2 Related Research

3.3 Previous Studies on Reinforcement Learning and Behavior Trees

There are very few studies on this subject. In [16],Dey and Child used Q-learning to improve behavior trees design (QL-BT), however, the Q-learning process is independent of the behavior tree. In [63], it modifies the behavior tree to be learnable. The issues with these researches are two-fold: 1) the problem studied in both project is too simple to demonstrate the advantage of using behavior tree to solve hierarchical reinforcement learning tasks, or to scale up to apply to most domains. Both studies used the same example of a prey/predictor game with a small state space of 243, with short term goals and abundant reward signals.In term complexity, this is a drastically different setting than what is explored in this paper.

2) Unlike our approach, the implementation in both papers are not utilizing modularity of behavior tree to efficiently learn in the hierarchical tasks. The results from both studies are inconclusive to prove that it's advantageous to combine behavior tree and reinforcement learning, compared to reinforcement alone.

3.3.1 Hierarchical Reinforcement Learning

Options framework [58] introduced the concept of temporal abstraction. It transforms a Markov Decision Processes (MDP) that consists of individual states to a collections of segments of states, semi-MDPs. An option can be several states or one single state. Later, many option-based methods were built upon this concept to increase the applicability of agent skills, for example, portable options [35] method build agent-oriented options instead of state-specific options, so that the knowledge obtained in one domain can be transferred to another problem. MAXQ algorithm [18] decomposes the value function in MDP, it uses smaller components or segments of MDPs. [36] uses DQN [43] to handle large state space and breaks the overall goal into sub-goals. The meta-controller to output current sub-goal, and uses controller to produce action that is suitable to current sub-goal.

We built upon these ideas, BT-RL formulates complex problem goals to sub-trees, each with respective goals and available actions, which encourages effective exploring by eliminating costly trials and errors on unfeasible actions. BT-RL leverages the modularity of behavior tree, the agent can easily re-use acquired knowledge in the same domain for different sub-goals. Similarily, [2] uses the notion of *modularity* to extract only relevant state information to make the optimal decision in abstract space and original space, ignoring the irrelevant part of state values. Difference in idea is we don't explicitly decompose value functions (into an additive function of subtasks). [2] also used an Lisp augumented programming language to achieve such goal. Though similar in idea, behavior tree as a modern graphical language, can capture the idea of state abstraction, but more human-interpretable and user friendly, and more compatible with other recent packages such as Tensorflow and Keras.

3.3.2 Genetic Programming and Behavior Trees

In [8], Colledanchise et al. applied Genetic Programming with greedy algorithm to behavior trees in solving a-priori unknown platform game *Super Mario*. Behavior trees here refer to generalized And-Or-Trees [9], whose composites nodes include Sequence/And node and Selector/Or node exclusively.

Genetic Programming (GP) treats individual policy as a gene, then evolves them using an evolutionary algorithm. Evolutionary algorithm exploits the natural processes - crossover, mutation, and selection, introduces small changes in individual policy and selects the onces that can successfully complete desired tasks.*Crossover* of two behavior trees is randomly swapping a subtree from tree A with a subtree of tree B. *Mutation* means replace a node in a behavior tree with a node of the same type, for example, replacing an action node with a different action node, or replacing a Sequence node with a Selector node (they are both composite nodes). *Selection* involves using a value function to evaluate newly formed behaviors trees and compare them against previous generation, and the ones that outperforms their parents have higher chance of being selected. To encourage convergence, *simulated annealing* is used, where large number of first generation nodes have mutations, and mutation frequency gradually decreases over generations.

Compared to Finite State Machine (FSM) based method stated in [19] and pure GP method stated in [52], GP with greedy algorithm achieves higher reward value and utilizes less resource (number of nodes).

3.3.3 Deep Q Learning

While tabular Q-learning might be sufficient in solving small state space problems, it is not for large discrete or continues state space. Therefore, it's important to use deep learning methods to estimate Q values. Deep Q networks (DQN) [43] introduced two important concepts: replay buffer and target network. The use of replay buffer increases sample efficiency and de-correlates samples. Target network makes the optimization goal more stable. However,

DQN serverly over-estimates state values, which in turn impact performance. Double DQN [26] decouples the action sampling and state value estimation to improve the over-estimation problem.

3.3.4 Other RL algorithms - Policy Gradient, Actor-Critic Algorithm

Policy gradient starts from algorithm REINFORCE [61]. It involves three main steps: 1) sample several trajectories from policy; 2) sum up their rewards, fit a model to estimate return; 3) improve the policy. Policy gradient is on-policy, in practice the gradient has high vairance, and requires a lot of samples. To reduce variance, generalized advantage estimation (GAE) [53] subtract value (V(s)) from Q value (Q(s,a)), and uses the advantage value to compute the gradient. To improve data efficiency, proximal policy optimization (PPO) [55] uses an objective function to compute gradient based on fixed-length trajectory segments, which enables multiple epochs of minibatch updates.

Most of reinforcement learning algorithms are either actor-only or critic-only. Actor-only methods changes the policy directly, the drawbacks are high variance and no accumulation and consolidation of previously acquired knowledge. Critic-only algorithms rely on some sort of value function approximation, the drawbacks are poor convergence (non-tabular fitted Q value functions don't converge in theory and rarely converge in practice) and slight change in value function can cause turbulent policy changes.

Actor-critic algorithm [33] [57] combines the strength of both sides. The critic learns a value function from an approximate estimation, e.g. neural network, and the actor learns its policy through gradient-based method with each value function update. The critic has an improved version of value function, which subtracts a baseline to reduce variance. An example value function is advantage function (A), defined as:

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t) \tag{3.1}$$

$V(s_t)$ is considered a good choice of baseline because it's the *expected* value of a state. t is the current time step.

Given that Q is the "the reward to go":

$$Q(s_t, a_t) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r(s'_t, a'_t) \tag{3.2}$$

So A can be re-written as:

$$\hat{A}^{\pi}(s_t, a_t) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r(s_{t'}, a_{t'}) - \hat{V}^{\pi}(s_t) \tag{3.3}$$

This means instead of fitting both Q and V functions, one only need to fit V function to approximate the advantage function (A). In practice, A is an estimator that comes from sampling, therefore the notation is $\hat{A}$. V is also a fitted value, not an analytical solution. π denotes policy, all these function estimations are under policy π, their values change along with new policy roll-out.

Batch actor-critic algorithm includes the following steps:

1. sample s_i, a_i from $\pi_\theta(a|s)$

2. fit $\hat{V}^{\pi}_{\phi}(s)$ to sampled values

3. evaluate $\hat{A}^{\pi}(s_i, a_i) = r(s_i, a_i) + \hat{V}^{\pi}_{\phi}(s'_i) - \hat{V}^{\pi}_{\phi}(s_i)$

4. $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta log \pi_\theta(a_i|s_i) \hat{A}^{\pi}(s_i, a_i)$

5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Through the advantage function (A), variance is reduced by subtracting the state-dependent baseline. However, if it relies on Monte-Carlo sampling, which means it's single-sample estimate, the variance is still pretty high. The variance comes from growing uncertainty when time progresses, as in we have higher confidence in the prediction of a time point that's closer to the sampling time, compared to a prediction for far away future. One way to deal with this issue is to have a cutoff $t' = n$, as shown in 3.4. Intuitively, it means only trust the value prediction before nth time steps, after that there will be too many possibilities

that the prediction is no longer useful. $n > 1$ proven to be effective in practice. However, deciding n still takes experiences and fine-tuning.

$$\hat{A}^{\pi}(s_t, a_t) = \sum_{t'=t}^{t+n} \gamma^{t'-t} r(s_{t'}, a_{t'}) - \hat{V}^{\pi}(s_t) + \gamma^n \hat{V}^{\pi}(s_{t+n}) \tag{3.4}$$

Schulman et al. [54] introduced Generalized Advantage Estimation (GAE) to solve this problem. Instead of manually choosing a n, GAE cut everywhere all at once and get the average A, as shown in:

$$\hat{A}^{\pi}_{GAE} = \sum_{n=1}^{\infty} w_n \hat{A}^{\pi}_n(s_t, a_t) \tag{3.5}$$

$$w_n \propto \lambda^{n-1} \tag{3.6}$$

This gives a weighted combination of n-step returns, and the weights w_n falls down exponentially as n increases. Intuitively, this is interpreted as preference of earlier cutoff, which is equivalent of less variance. In [54], batch actor-critic with GAE was used to solve 3D locomotion tasks in continuous space, learning running gaits for bipedal and quadrupedal.

Mnih et al. [42] used asynchronous online actor-critic with parallelized batch (AC3) and n-step returns ($n = 4$) to solve Atari games and navigate random 3D mazes using visual input. AC3 is light weight, compared to many current learning algorithms, can run on a single multi-core CPU. And it surpasses the current state-of-the-art algorithms on Atari domain.

3.4 Experiments

In this section, we tested behavior tree augmented reinforcement learning (BT-RL) agent in various difficulty tasks, and compared its performance to reinforcement learning agent (RL). We found that BT-RL learns with higher efficiency than RL alone. And BT-RL has an initial performance boost even without training. The reasons are two-fold: 1) behavior tree provides a natural problem formulation, which only permits feasible actions suitable for

current sub-task. 2) The modularity of behavior tree enables the agent to re-use acquired knowledge much more effectively.

3.4.1 General Algorithm Description

In behavior tree augmented reinforcement learning setting (BT-RL), learning happens in each action node. Whenever an action is taken, it puts a sample (s, a, r, s') record into replay buffer, and takes one gradient step with mini-batch in Q model fitting. Target model is updated every 10,000 steps. Every priority/selector node (denoted by "?") is replaced by Q-value node. Q-value node computes the values of each action, and ticks its children (usually a subset of all actions) in a decreasing order of values. The main difference between BT-RL and RL is the use of behavior tree.

3.4.2 Experiments - Taxi Domain

Description

A taxi drives around a 5x5 map, a passenger waits at a randomly chosen corner (out of the 4 positions, R, G, B, and Y), and the goal is to pick up the passenger and drive to destination (also randomly chosen one of of RGBY). Some of the blocks are separated by walls (indicated as $\|$ in the map), which means the taxi can't drive through it. Each episode, the starting position of the taxi is chosen among all blocks of the map, passenger and destination are chosen from the corners. It is possible that the taxi, passenger and destination are located in thesame block at the start of a game. The total number of states is 500.

There are 6 actions available: 0 - move south, 1 - move north, 2 - move east, 3 - move west, 4 - pick up, 5 - drop off. Other than drop off action, all actions have a -1 point reward. Drop-off signals the end of an episode, if passenger is dropped off at the correct destination, it generates a positive reward of 10 points, otherwise -10 points. The maximum number of steps in one episode is 200. The game ends either by the drop-off action, or by reaching the step limits.

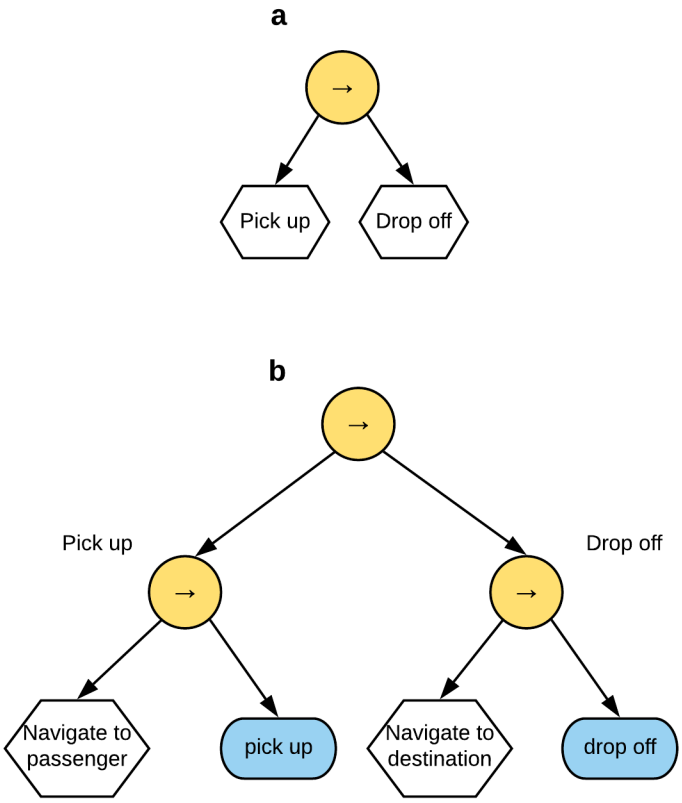


Figure 3.1: Hierarchical BT Disign for Taxi game

3.4.3 Hierarchical Behavior Tree Design

To complete a successful ride, the taxi needs to first pick up the passenger and then drops off passenger at the destination, as shown in 3.1 a). To expand, pick-up is a sequential process of navigation to passenger and pick-up action; similarly, drop-off is a two-step process of navigation to destination then the drop-off action, as shown in 3.1 b). The navigation component consists of a set of actions, north, south, east, and west, which are governed by a modified priority node, children are ticked based on decreasing Q-value order. The detailed BT design for reinforcement learning is shown in 3.2.

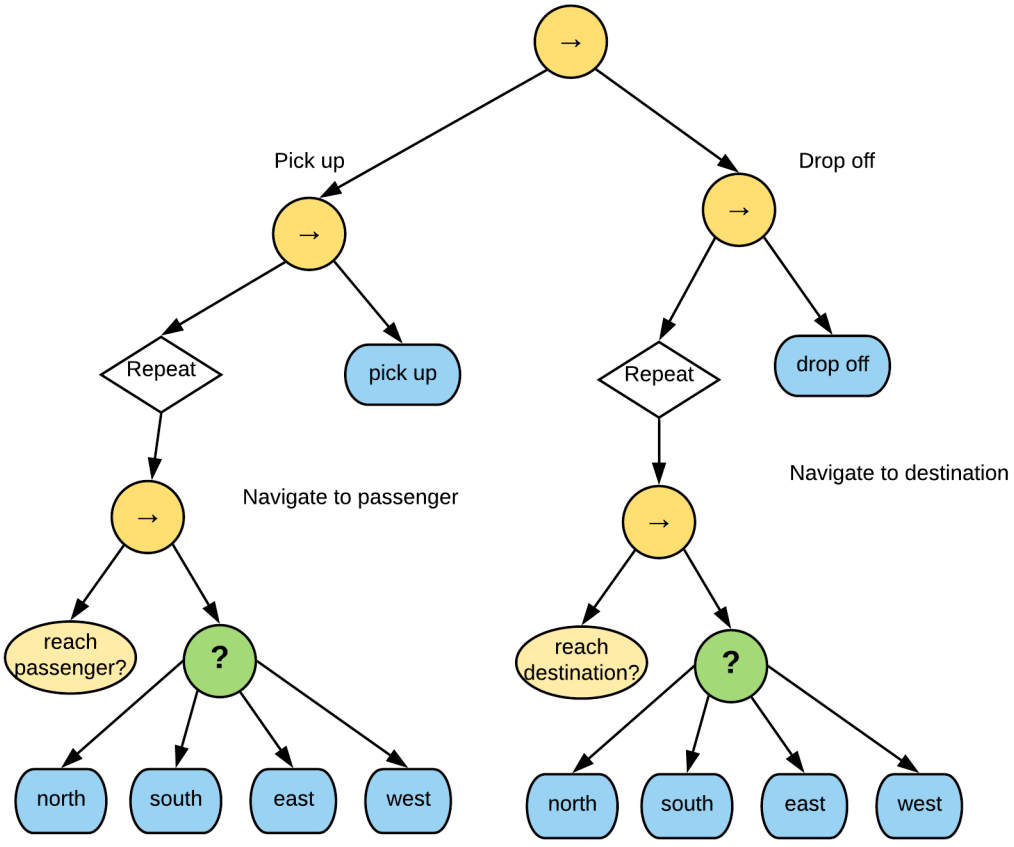


Figure 3.2: BT Design for Taxi game

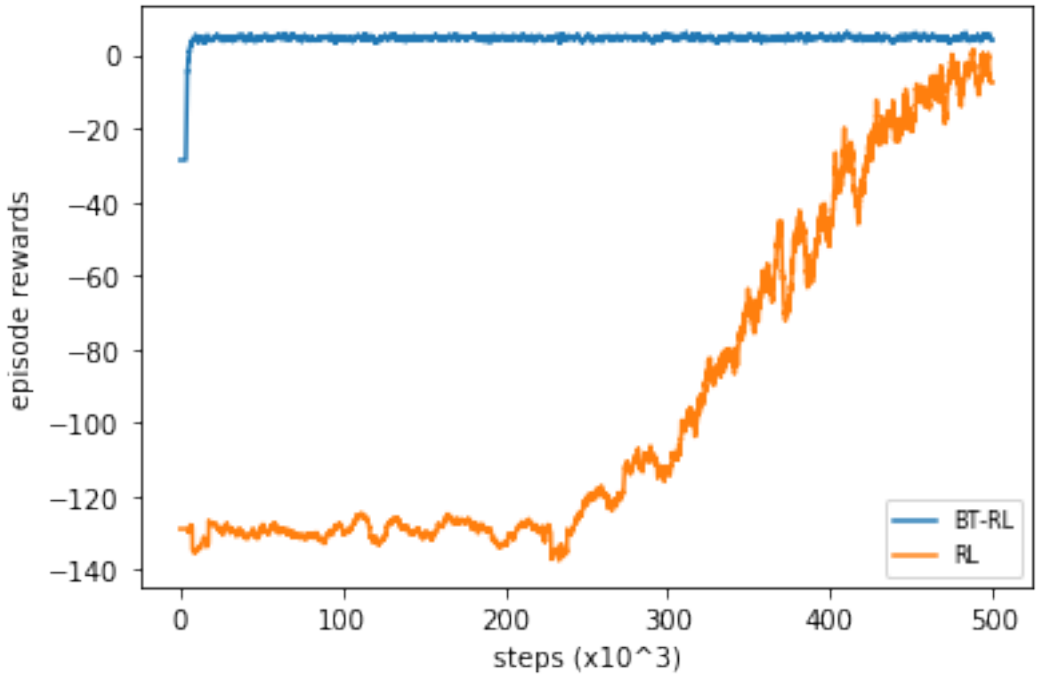


Figure 3.3: Taxi Results RL vs. BT-RL

Performance Comparison

As shown in 3.4, behavior tree augmented reinforcement learning (BT-RL) took a very short amount of time to master the game (less than 20K steps), while RL alone needs 500 K steps. At the beginning of the training, BT-RL outperforms RL, this is because that BT provides such a structure that it only searches the feasible actions. And the structure, or the separation of subtasks, helps tremendously on the learning efficiency. The advantage of BT's modularity is demonstrated on the faster learning - the agent learned how to navigate through the map during the pick-up phase, and this knowledge is applicable when dropping off passenger. Even though the sub-goal changed from pick-up to drop-off, BT-RL agent doesn't re-learn navigation from the scratch, unlike RL-only agent.

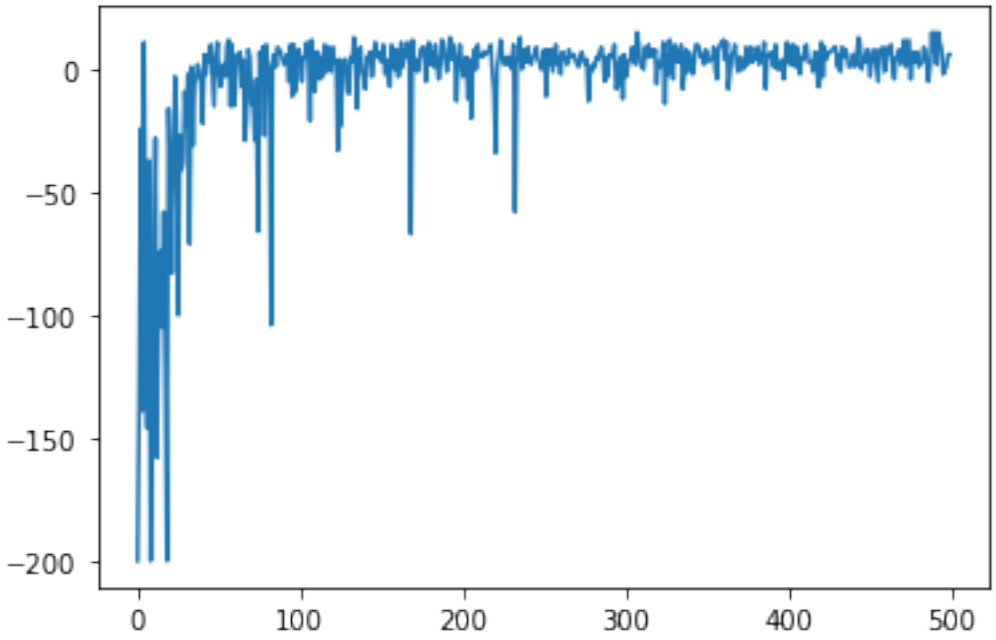


Figure 3.4: Taxi Results BT-RL only (need smoothing)

3.4.4 Pandemic [1]

Consider a scenario of pandemic control, where an unknown infectious disease breaks out, a few cities report infected cases, but the early detection method is unreliable, and people still get to travel among cities. A group of medical staff try to control this outbreak. The goal is to collect enough samples to research the cure, and eventually eradicate this disease. Pandemic the board game has 7.5 million states.

City Map

The map consists of 7 cities. Cities are represented as nodes in a graph, with connectivity ranging from 1 to 4. The population of a city is a function of its connectivity. Similar to cities in the real world, the nodes and edges are fixed for the map, as shown in **??**.

[1] This is inspired by Pandemic: The Board Game. The author does not have sufficient domain knowledge to make any claims that pertinent to disease control or epidemiology in general.

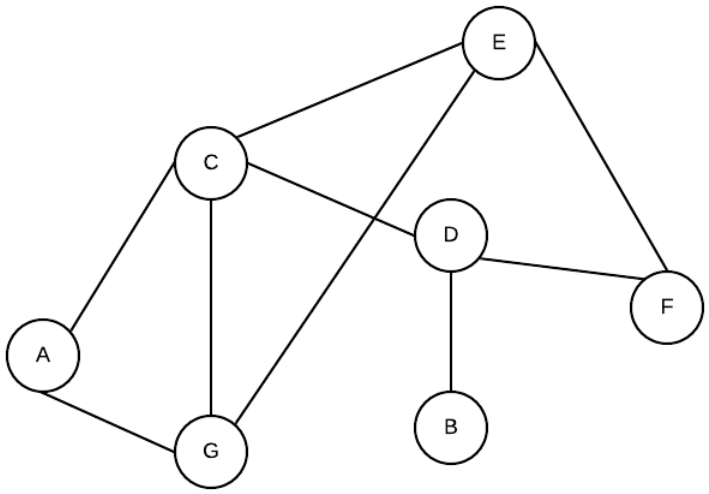


Figure 3.5: Pandemic Game Map

Game play

At the beginning of each episode, two cities are randomly chosen to be "epicenters", with 3 (capped at maximum) infection units, while the rest of the cities have 0 infection units. The infection only spreads from epicenters to connected cities with a probability of 0.3. Upon spread, the neighbour city gets 1 more infection unit. When a city reaches 3 infection units, this city is considered to be epicenter and capable of spreading to nearby neighbors.

Each turn, the medical team, controlled by AI agent, has 4 actions, including travel, collect sample, research, and distribute cure. After AI takes 4 actions, the disease can spread for one step, simultaneously throughout the map. The winning condition is 0 epicenters on the map, meaning no more outbreaks/spreads. On the other hand, the AI loses if the disease gets out of control, all cities become epicenters.

Medical staff act on reinforcement learning with behavior tree or reinforcement learning alone. They have the following action available:

- Travel to a city. Traveling along each edge rewards/costs -1. Each travel move takes

up 1 action. Successful travel only happens when two cities are connected, otherwise, medical staff stays in the same city. Medical staff can only perform an action on a city when they're are physically in the city.

- Collecting samples. Takes 1 action to collect sample from one city. Two samples from different infected cities is needed for research for the cure. Successful collecting happens when more samples are needed, and the city is infected and uncollected before. Intermediate reward of 2 is given for a successful collecting, otherwise the reward is -5.

- Research for cure. Takes 1 action to research for the cure. Sufficient sample number (2) is needed, and the cure is not discovered yet. Successfully research has a reward of 5. For attempting research before having sufficient samples or after the cure already discovered, the penalty is -2.

- Distribute the cure. After the cure is discovered, medical staff can go to each city and cure the whole city. Once the city has the cure (vaccine), it won't be susceptible to infection again. Medical team gets a reward of -1 for distributing cure in a city that previously doesn't have this cure. For attempting to give a city that already has the cure, medical team uses more resources, the reward is -2.

3.4.5 Hierarchical Behavior Tree Design

Intuitively, the AI agent needs to perform a series of tasks of collecting enough samples, research, then distributing cure, to achieve the long-term goal of eliminating outbreaks, as shown in the highest level abstraction 3.6 a).

On the second level, "collect samples" is expanded to repeatedly collect samples on a infected and uncollected city, until we have enough samples. Research component is a two-step process, first checking the condition of whether or not we have enough component, if yes, proceeds to research for cure. "Distribute cure" to cities, similar to the first component,

is a repetition of giving cure to cities until there's no epicenters/outbreaks. This level of abstraction is shown in 3.6 b).

On the third level, "collect" is expanded into two components: navigate to a city that's infected and uncollected, and the collect sample action. Similarly, "cure" is expanded into navigation and send cure, as shown in 3.6 c).

The final expansion is on "navigation", where a set of all possible 7 travel actions is governed by a modified priority node. Travel action is considered to be successful if the indented city (A through G) is connected to the current location of the agent. The modified priority node will tick each child based on their respective Q value, estimated by a neural net. The final design of the behavior tree is shown in 3.7.

Conditions and Performance Comparison

Other than that behavior tree augmented reinforcement learning (BT-RL) employs behavior tree, it uses identical neural nets as reinforcement learning (RL) alone, for double Q learning. More in detail, the network structured as a series of dense layers, 24, 32, XXXXX. It use epsilon-greed policy, with $\epsilon = 0.1$, learning rate 10^{-4}. Target model is updated every 10^4 steps.

Compared to reinforcement alone (RL), behavior tree augmented reinforcement learning (BT-RL) is more efficient in learning tasks. The training period is 1.4 million steps, as shown in 3.8, behavior tree augmented reinforcement learning is much more efficient in learning the pandemic game, and reaching good performance in episode rewards within 1.4 million steps, while reinforcement learning alone barely made any progress. It's worth pointing out that the bump of performance of BT-RL at the start of the training is attributed to behavior tree, its structure only permits feasible actions.This formulation provides a better performance even at the beginning of the training, and further makes the exploration more effective. In addition, BT-RL agent can re-apply the acquired knowledge of navigating the map from collecting samples, to distributing cures. This sharing knowledge across sub-tasks is enabled by the modularity of behavior trees.

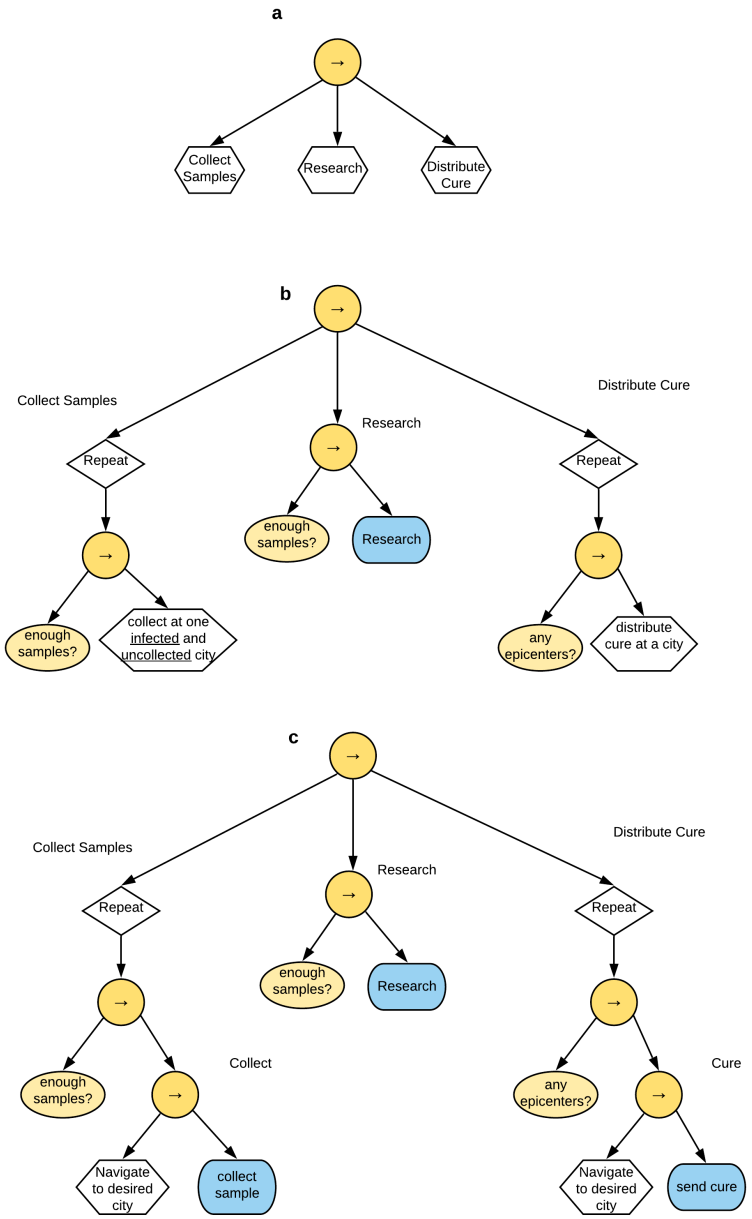


Figure 3.6: **Hierarchical Behavior Tree Design (Intermediates).** a) First level, highest level of abstraction. b) Second level abstraction, expanding three main tasks c) Third level abstraction, expanding "collect on one city" and "distribute cure to a target city" tasks.

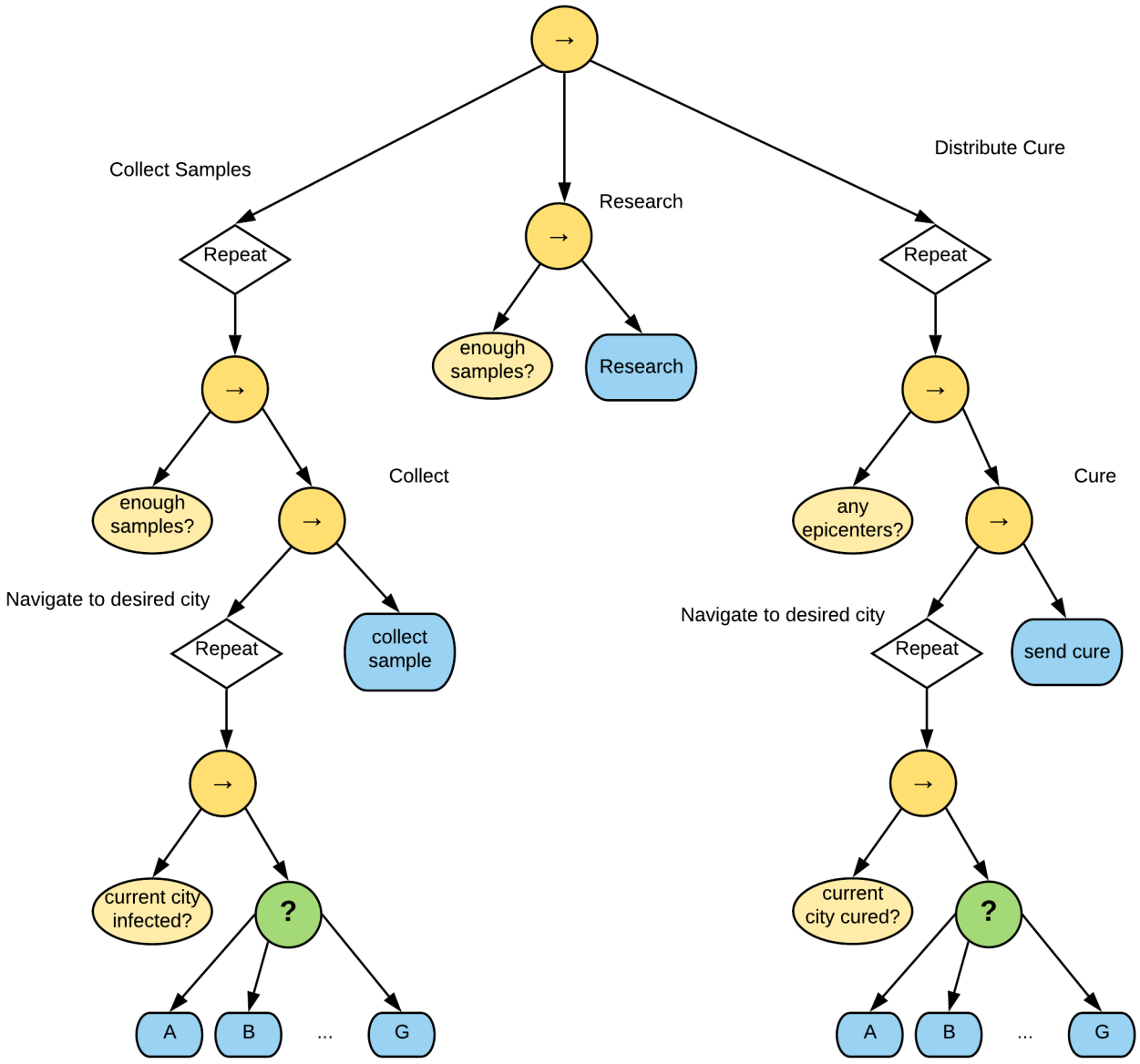


Figure 3.7: Pandemic Behavior Tree Architecture.

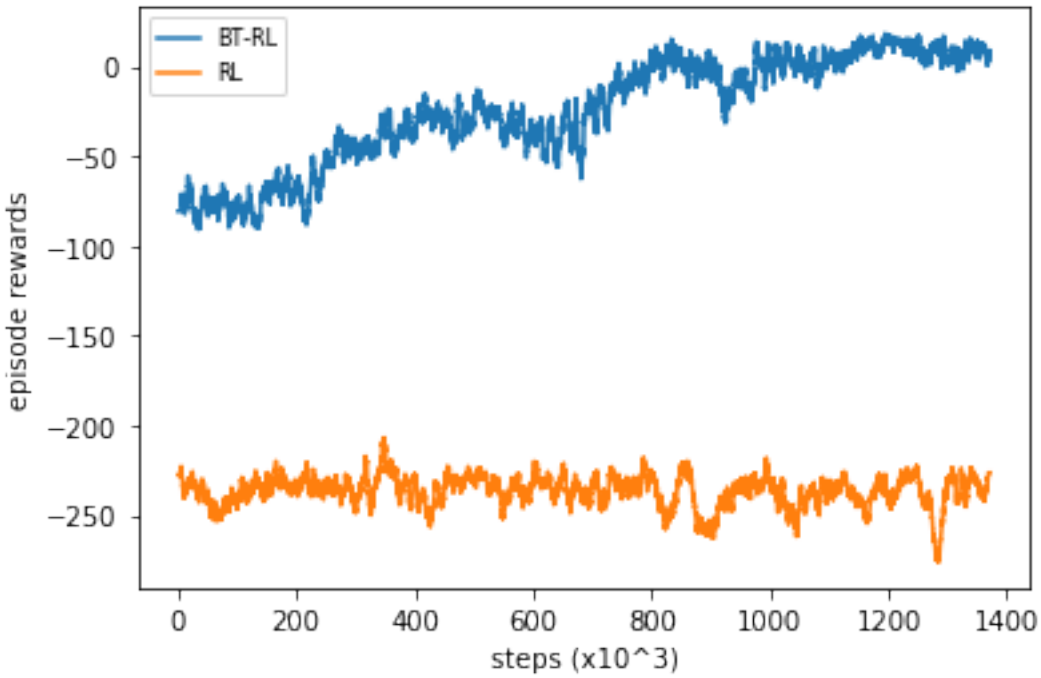


Figure 3.8: Pandemic results. Blue line is BT-RL results, orange is RL results.

3.4.6 The counterexample - Card Game

In this simple card game, we demonstrated BT-RL has advantages over RL when the task is long horizon and with certain complexity. The card game is a simple problem, where BT-RL shows no advantage.

Description

Each round the agent is dealt with a card valued from 1-10, the goal of the simple card game is to get cards that sum up to 21 as close as possible, without exceeding the limit. The state is a scalar number, indicating the sum of current cards. The agent has two actions: action 0, draw another card; and action 1, end the game/episode. One game episode can end in two ways: the agent chooses to do so by action 1, or, the last card dealt to the agent makes the sum greater than 21. The reward of one episode is calculated at the end of each game by the following:

1. if sum <= 21, reward = sum - 21

2. if sum > 21, reward = -21

The range of reward for one episode is [-21, 0]. Due the the simplicity of this game, traditional Q-learning is used, following the update equation:

$$Q'_{i,action_x} = (1-\alpha)Q_{i,action_x} + \alpha(r + \gamma max_a Q_{i+1,a})$$

Where $Q'_{i,action_x}$ is the updated Q value for state i, chosen action x, and $Q_{i,action_x}$ is the previous value. α is the learning rate, γ is discount factor ranging [0, 1], maximum of the next state Q values over all possible actions is $max_a Q_{i+1,a}$.

The corresponding behavior tree has has two main modules, the "draw" module and the "end" module, for the two actions (0 and 1) respectively. Each module is consisted of a condition node and an action node. The number of states is 23.

Conditions and Results

For Q-learning the agent is trained in a 5000 episodes of games, and Q values for each state-action pair are updated accordingly. The results are listed in TABLE. The trained policy is further translated into conditions for Q value improved behavior tree (Q-BT). Derived from the Q value table, the policy stands: keep drawing if sum < 14.

To compare the performance of RL and BT-RL, Q-agent with zero exploration rate and Q-BT are tested in 5000 episodes of games. The resulting returns are comparable, XXXX, respectively. This result makes sense, because this card game setting is fairly simple, any training process or design can reach the correct or optimal solutions. This comparison proves that Q-BT can at least perform equally well as reinforcement learning methods in this game scenario.

There's is no obvious performance advantage of BT-RL compared to RL alone. This is because all actions are available to all states, which is different than the more complex settings in the two examples of taxi and pandemic. In the card game, it doesn't need re-use of knowledge.

3.5 Discussion

In this paper, we demonstrated the advantage in formulating problems with behavior tree when solving long horizon, sequential, sparse rewards problems. This formulation provides information pertinent to the problem (and the sub-tasks), as well as the actions available to

	Mean	Std
BT-RL	-5.22	4.62
RL	-5.30	4.61

Table 3.1: Card game tests

the agent, given the current state and sub-task goal. Problem formulation is often critical in solving complex tasks . Moreover, behavior tree facilitate modular knowledge; knowledge acquired in one module can be easily re-applied for a different sub-task, within the same problem. This re-applying knowledge, compared to re-learning knowledge (regular RL agentS), determines the high efficiency of learning in BT-RL agents.

Our experiments are proof-of-concept for BT-RL agents' capability of handling complex tasks. In this setting, the state abstraction are given to the agent - we extract the precise information needed to describe a state (as state vector). To realize the full potential of BT-RL, agents will need to learn how to acquire such problem abstraction [34]. For novel problem, such as Pandemic the board game, this abstraction is not a trivial process; it took a few trial-and-errors even for human experts. And we have tremendous pool of knowledge embedded in our common senses, which an AI agent needs to acquire. The ability to abstract problem holds the key to the next level of unsupervised learning.

Chapter 4
CONTRIBUTIONS

In this thesis, I explored applications of behavior tree in solving high-cognitive problems (IKBT), such as analytical inverse kinematics, and solving hierarchical reinforcement learning problems more efficiently (BT-RL). The contributions of IKBT are:

1. First automated, generalized solution for symbolic inverse kinematics
2. Automatic dependency tracking and solution generation
3. **Modular knowledge** for extensible toolbox
4. **Human-interpretable** strategy

For BT-RL, the contributions are:

1. New learning paradigm for hierarchical reinforcement learning
2. New RL learning environment - Pandemic
3. **Modular** subtask
4. **Human-interpretable** behavior tree design

These are just a few examples demonstrating the value of behavior trees. With the two characteristics of behavior tree - modular and human-interpretable, the potential of behavior tree in AI applications is limitless.

Chapter 5

APPENDIX

5.0.1 Behavior Trees as MDP Problems

Behavior Trees can describe MDP problems, where each action node contains state and action. Result of an action can lead to a new state. Here is to prove the lemma: **Each node can only be reached by success or failure exclusively**. In other words, for each node in A matrix, only one of the columns ($P_{SUCCESS}$ and $P_{FAILURE}$) is checked, never both.

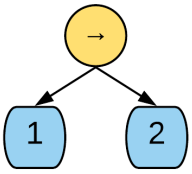
To consider the base cases with 2 action leaves: **case a**) nodes with parent sequence node, action node 2 is ticked iff action node 1 succeeds, transition matrix shown in 5.1; **case b**) for nodes with selector node, action node 2 is ticked iff action node 1 fails, as shown in 5.2. These two cases satisfies the lemma.

Complex trees with more actions nodes can be built by adding them one by one. If the third node is added without additional composite nodes (fig. c and d), the access to the 3rd node depends on the parent node type, while the original two nodes remain unchanged. Transition matrix for **case c** 5.3 shows that Node 2 is reached through the success of Node 1 (or P_S, similar to case a), and Node 3 is reached through the success of Node 2 (P_S). In **case**

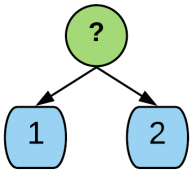
	Node 1		Node 2		Output	
	P_S	P_F	P_S	P_F	P_S	P_F
Node 1			✓			✓
Node 2					✓	✓

Table 5.1: Transition matrix for base case a (fig.a)

Base Cases:



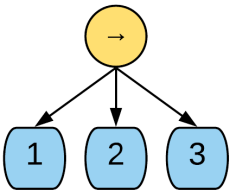
a



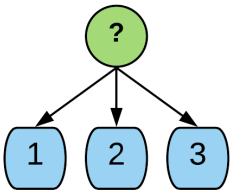
b

+ 1 Cases:

Without adding composite node

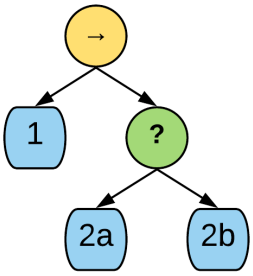


c

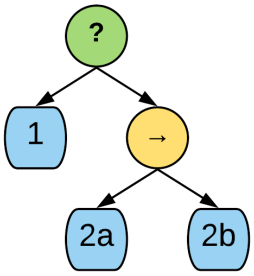


d

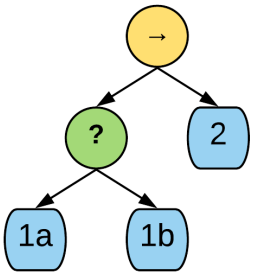
Adding one composite node



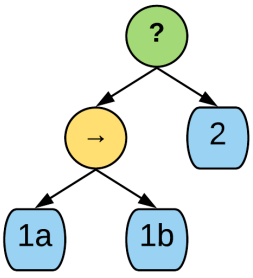
e



f



g



h

	Node 1		Node 2		Output	
	P_S	P_F	P_S	P_F	P_S	P_F
Node 1				✓	✓	
Node 2					✓	✓

Table 5.2: Transition Matrix for base case b (fig.b)

	Node 1		Node 2		Node 3		Output	
	P_S	P_F	P_S	P_F	P_S	P_F	P_S	P_F
Node 1			✓					✓
Node 2					✓			✓
Node 3							✓	✓

Table 5.3: Transition Matrix for case c (fig.c)

d, Node 3 is ticked iff the previous two nodes fail 5.4. These two cases satisfy the lemma.

When additional composite nodes are involved, it can be interpreted as an expansion - one node is expended as a subtree. For example, fig.e is an expansion of the tree in fig.a, where node 2 in fig.a expands a subtree of selector and two action nodes node 2a and 2b in fig.e. The first child (node 2a) in the new subtree inherits from its ancestor (node 2) - the first child is reached the same way as the ancestor. The second child (node 2b) is reached by SUCCESS or FAILURE depending of its parental composite node. Therefore, the first child of said subtree doesn't violate the lemma, neither does the second child of the subtree. And the other "unexpanded" node in the original tree remains unchanged. The detailed cases are shown in transition tables 5.5 5.6.

Above are all the possible scenarios of adding a new action node, and all cases satisfy the lemma. Q.E.D

	Node 1		Node 2		Node 3		Output	
	P_S	P_F	P_S	P_F	P_S	P_F	P_S	P_F
Node 1				✓			✓	
Node 2						✓	✓	
Node 3							✓	✓

Table 5.4: Transition Matrix for case d (fig.d)

	Node 1		Node 2a		Node 2b		Output	
	P_S	P_F	P_S	P_F	P_S	P_F	P_S	P_F
Node 1			✓					✓
Node 2a						✓	✓	
Node 2b							✓	✓

Table 5.5: Transition Matrix for case e (fig.e)

	Node 1		Node 2a		Node 2b		Output	
	P_S	P_F	P_S	P_F	P_S	P_F	P_S	P_F
Node 1				✓			✓	
Node 2a					✓			✓
Node 3b							✓	✓

Table 5.6: Transition Matrix for case f (fig.f)

	Node 1a		Node 1b		Node 2		Output	
	P_S	P_F	P_S	P_F	P_S	P_F	P_S	P_F
Node 1a				✓	✓			
Node 1b					✓			✓
Node 2							✓	✓

Table 5.7: Transition Matrix for case g (fig.g)

	Node 1a		Node 1b		Node 2		Output	
	P_S	P_F	P_S	P_F	P_S	P_F	P_S	P_F
Node 1a			✓			✓		
Node 1b						✓	✓	
Node 2							✓	✓

Table 5.8: Transition Matrix for case h (fig.h)

5.1 Conversion between transition matrix to behavior tree

It is obvious that given any behavior tree, there exists a transition matrix. It is also true that for any given transition matrix, it can be converted to a behavior tree. Here is how:

1. the last row, corresponds to the last node of a tree, have both success and failure check marks in "Output" column.

2. for any row that's not the bottom row, if one of the checks lands in "Output" column, the corresponding node is on the top level of the behavior tree.

3. how to locate a subtree: from the first row that doesn't have a check mark in "output" column, to (but exclude) the column that has more than one check marks.

4. the type of composite node (sequence or selector) can be inferred from how one node transits to another.

5. different sub-columns in "Output" indicates that new subtree(s). This rule is especially useful for subtrees that are in the right side of tree, otherwise, it's hard to tell.

BIBLIOGRAPHY

[1] Krishna Aluru, Stefanie Tellex, John Oberlin, and James MacGlashan. Minecraft as an Experimental World for AI in Robotics. In *AAAI Fall Symposium*, 2015.

[2] David Andre and Stuart J. Russell. State abstraction for programmable reinforcement learning agents. In *Eighteenth National Conference on Artificial Intelligence*, page 119–125, USA, 2002. American Association for Artificial Intelligence.

[3] J. A. Bagnell et al. An integrated system for autonomous robotics manipulation. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2955–2962, Oct 2012.

[4] Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(1–2):41–77, January 2003.

[5] F. Chapelle and P. Bidaud. A closed form for inverse kinematics approximation of general 6r manipulators using genetic programming. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation*, volume 4, pages 3364–3369 vol.4, 2001.

[6] I-Ming Chen and Yan Gao. Closed-form inverse kinematics solver for reconfigurable robots. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation*, volume 3, pages 2395–2400 vol.3, 2001.

[7] M. Colledanchise, R. Murray, and P. Ogren. Synthesis of Correct-by-Construction Behavior Trees. In *Intelligent Robots and Systems (IROS 2017), 2017 IEEE/RSJ International Conference on*, pages 1482–1488, Sept 2017.

[8] M. Colledanchise, R. Nattanmai Parasuraman, and P. Ogren. Learning of behavior trees for autonomous agents. *IEEE Transactions on Games*, pages 1–1, 2018.

[9] M. Colledanchise and P. Ögren. How behavior trees generalize the teleo-reactive paradigm and and-or-trees. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 424–429, Oct 2016.

[10] Michele Colledanchise, Alejandro Marzinotto, Dimos V. Dimarogonas, and Petter Ogren. The advantages of using behavior trees in multi-robot systems. In *International Symposium on Robotics (ISR)*, June 2016.

[11] Michele Colledanchise, Alejandro Marzinotto, and Petter Ogren. Performance analysis of stochastic behavior trees. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3265–3272. IEEE, 2014.

[12] Michele Colledanchise and Petter Ogren. How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees. *IEEE Transactions on Robotics*, 33(2):372–389, 2017.

[13] P. I. Corke. A robotics toolbox for matlab. *IEEE Robotics Automation Magazine*, 3(1):24–32, Mar 1996.

[14] John J. Craig. *Introduction to Robotics: Mechanics and Control*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1989.

[15] R. Dey and C. Child. Ql-bt: Enhancing behaviour tree design and implementation with q-learning. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, pages 1–8, Aug 2013.

[16] R. Dey and C. Child. Ql-bt: Enhancing behaviour tree design and implementation with q-learning. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, pages 1–8, Aug 2013.

[17] Rosen Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, 2010.

[18] Thomas G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Int. Res.*, 13(1):227–303, November 2000.

[19] Ramón García-Martínez and Daniel Borrajo. An integrated approach of learning, planning, and execution. *Journal of Intelligent and Robotic Systems*, 29(1):47–78, Sep 2000.

[20] Kelleher R Guerin, Colin Lea, Chris Paxton, and Gregory D Hager. A framework for end-user instruction of a robot assistant for manufacturing. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6167–6174. IEEE, 2015.

[21] Dan Halperin. Automatic kinematic modelling of robot manipulators and symbolic generation of their inverse kinematics solutions (extended abstract), 1991.

[22] Blake Hannaford, Antal Bejczy, Pietro Buttolo, Manuel Moreyra, and Steven Venema. Mini-teleoperation technology for space research. In *Proceedings Micro Systems, Intelligent Materials and Robots (MIMR-95*, pages 524–527, 1995.

[23] Blake Hannaford, Danying Hu, Dianmu Zhang, and Yangming Li. Simulation results on selector adaptation in behavior trees. *CoRR*, abs/1606.09219, 2016.

[24] Blake Hannaford, Danying Hu, Dianmu Zhang, and Yangming Li. Simulation results on selector adaptation in behavior trees. *arXiv preprint arXiv:1606.09219*, 2016.

[25] Hado V. Hasselt. Double q-learning. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2613–2621. Curran Associates, Inc., 2010.

[26] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 2094–2100. AAAI Press, 2016.

[27] L. G. Herrera-Bendezu, E. Mu, and J. T. Cain. Symbolic computation of robot manipulator kinematics. In *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, pages 993–998 vol.2, Apr 1988.

[28] D. Hu, Y. Gong, B. Hannaford, and E. J. Seibel. Semi-autonomous simulated brain tumor ablation with ravenii surgical robot using behavior tree. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3868–3875, May 2015.

[29] Damian Isla. Building a better battle: The halo 3 ai objectives system. http://web.cs.wpi.edu/~rich/courses/imgd4000-d09/lectures/halo3.pdf, 2016.

[30] A. Johansson and P. Dell'Acqua. Emotional behavior trees. In *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 355–362, Sept 2012.

[31] Laura Kelmar and Pradeep K. Khosla. Automatic generation of forward and inverse kinematics for a reconfigurable modular manipulator system. *Journal of Robotic Systems*, 7(4):599–619, 1990.

[32] J. C. Knight. Safety critical systems: challenges and directions. In *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, pages 547–550, May 2002.

[33] Vijay R. Konda and John N. Tsitsiklis. Actor-critic algorithms. In S. A. Solla, T. K. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 1008–1014. MIT Press, 2000.

[34] George Konidaris. On the necessity of abstraction. *Current Opinion in Behavioral Sciences*, 29:1 – 7, 2019. SI: 29: Artificial Intelligence (2019).

[35] George Konidaris and Andrew Barto. Building portable options: Skill transfer in reinforcement learning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 895–900, 2007.

[36] Tejas D. Kulkarni, Karthik R. Narasimhan, Ardavan Saeedi, and Joshua B. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 3682–3690, Red Hook, NY, USA, 2016. Curran Associates Inc.

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

[38] Chong-U Lim, Robin Baumgarten, and Simon Colton. *Evolving Behaviour Trees for the Commercial Game DEFCON*, pages 100–110. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[39] D. Manocha and J. F. Canny. Efficient inverse kinematics for general 6r manipulators. *IEEE Transactions on Robotics and Automation*, 10(5):648–657, Oct 1994.

[40] A. Marzinotto, M. Colledanchise, C. Smith, and P. Ögren. Towards a unified behavior trees framework for robot control. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5420–5427, May 2014.

[41] Alejandro Marzinotto, Michele Colledanchise, Christian Smith, and Petter Ogren. Towards a unified behavior trees framework for robot control. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5420–5427. IEEE, 2014.

[42] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning (ICML)*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR.

[43] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[44] Richard M. Murray, S. Shankar Sastry, and Li Zexiang. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1994.

[45] M. Nicolau, D. Perez-Liebana, M. O'Neill, and A. Brabazon. Evolutionary behavior tree approaches for navigating platform games. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(3):227–238, Sept 2017.

[46] M. Nicolau, D. Perez-Liebana, M. O'Neill, and A. Brabazon. Evolutionary behavior tree approaches for navigating platform games. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(3):227–238, Sept 2017.

[47] Petter Ogren. Increasing modularity of uav control systems using computer game behavior trees. In *2012 AIAA Guidance, Navigation, and Control Conference*, 08 2012.

[48] Petter Ogren and Maja Winstrand. Minimizing mission risk in fuel-constrained unmanned aerial vehicle path planning. *Journal of Guidance, Control, and Dynamics*, 31(5):1497–1500, Sep 2008.

[49] Junhyuk Oh, Valliappa Chockalingam, Satinder P. Singh, and Honglak Lee. Control of memory, active perception, and action in minecraft. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 2790–2799, 2016.

[50] Renato Pereira. Behavior3, 2015.

[51] D.L. Pieper and B Roth. The kinematics of manipulators under computer control. In *Proceedings of the Second International Congress on Theory of Machines and Mechanisms*, pages 159–169, 1969.

[52] Kirk Y. W. Scheper, Sjoerd Tijmons, Coen C. de Visser, and Guido C. H. E. de Croon. Behaviour trees for evolutionary robotics. *CoRR*, abs/1411.7267, 2014.

[53] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[54] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[55] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[56] Sanjit A. Seshia and Dorsa Sadigh. Towards verified artificial intelligence. *CoRR*, abs/1606.08514, 2016.

[57] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pages 1057–1063, Cambridge, MA, USA, 1999. MIT Press.

[58] Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1–2):181–211, August 1999.

[59] B. Vargas and E. F. Morales. Solving navigation tasks with learned teleo-reactive programs. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4185–4185, Sept 2008.

[60] M. Wenz and H. Worn. Solving the inverse kinematics problem symbolically by means of knowledge-based and linear algebra-based methods. In *2007 IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007)*, pages 1346–1353, Sept 2007.

[61] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256, May 1992.

[62] Dianmu Zhang and Blake Hannaford. Ikbt: Solving symbolic inverse kinematics with behavior tree. *Journal of Artificial Intelligence Research*, 65:457–486, 07 2019.

[63] Q. Zhang, Q. Yin, and Y. Hu. Modeling cgfs behavior by an extended option based learning behavior trees. In *2017 IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics (RAM)*, pages 260–265, 2017.