The Winter Tree Problem: A Quantifiable Approach to Artificial General Intelligence

By Nicolas Kiely (nic.kiely@baitaservices.com)
2021-2024

The Winter Tree Problem (WTP) is a framework and given list of problems such that, if a given software application were to solve the problems and execute on that framework, it should be able to preform any task would be expected of an artificial general intelligence (AGI). Given the lack of a rigorous common definition of AGI, the Winter Tree Problem will serve as a stand-in for a set of specifications. The WTP is not necessarily a definitive or the most efficient approach but provides a way to create a roadmap to develop and break down the requirements needed to engineer an AGI-equivalent system.

The motivation behind the WTP came from a proprietary algorithm named the winter tree algorithm in late 2021 designed around dynamically constructing and refining action spaces for AI systems to solve problems with. The goal of this project was to create a generalized artificial intelligence system that would attempt to adapt and reason about novel problems it was presented with. As the scope of the project grew, it became clear that a system of cooperative algorithms would be needed to properly generalize the system, not just one. Hence the name of the Winter Tree Problem, and while the winter tree algorithm is a closed source implementation to solve one aspect of the WTP, the theory and approach is open to the public. The philosophy behind the WTP is to be as rigorous as possible while being technology-agnostic.

An open-source reference of the WTP specifications and framework code can be found here: https://github.com/NicolasKiely/Winter-Tree-Problem. Further code in this document will reference the files and classes in this repository.

As a final introductory note, the WTP is not related to generative AI. A program that solves WTP takes in information to update its world model, and outputs zero, one, or more actions in response. An action here meaning an attempted change to its own world model or in its environment. As a convention in the originally winter tree algorithm's code, actions act upon the AI's world model whereas explicit interactions with its environment are called activities.

Goal

The main goal is to build an AI system capable of following objectives it is given, while efficiently adapting constraints and problems that may arise. Ideally such a system should need little to no custom code to address any given task at hand, outside of interfacing with other software. Training and learning should be efficient, where the time involved in learning new tasks or information is independent of the size of an existing knowledge base. Improving algorithmic performance should take precedence over relying on increasing hardware compute resources. The operations of an ideal AI system should be transparent and inspectable, not opaque.

This ideal system should have the ability to understand differing degrees of abstraction in processing environmental information and in the formulation of responses. Feedback in improving operations can be processed an evaluated through different methods, as well as through different degrees of learned abstraction. Existing reinforcement learning can handle situations such as learning chess where the feedback is a Boolean flag indicating if the model won. However, if feedback is also given in the textual form “you lost because you kept the king unguarded”, that information should also be actionable in the learning process. Similar learning can be through the violation of learned expectations, which may not be related to the current objective.

In summary, this system should be able to learn through experience, through feedback, and through reinforcement. Some of the feedback and experience can involve indirect inference, such as first processing the intent of textual or audio input and learning how that input may relate to a positive or negative assessment.

Conventions and Definitions

Compliant

A software program is WTP compliant given that it meets the requirements in this document and should achieve the goals described above. Such a system is a prospective artificial [general] intelligence, however the desired framing is around building AI software with well-defined specifications, predictable behavior, and understandable code. Artificial intelligence systems built on monolithic black-box machine learning algorithms violate this framing. A compliant program may also contain individual AI subcomponents, so for clarity the rest of this document will avoid calling it an AI system.

World Model

The world model for a complaint program is the structured set of all knowledge it obtains and uses in executing its tasks. This includes information about the state of the environment it may interact with, the set of agents that may interact with the environment, the

list of actions each agent may take, and the potential effects of each action an agent may take upon the environment. This information may be concrete or abstract.

The set of world model data is denoted G and may be treated as a graph or relational data set. Transformations on a world model G to a new model G' are denoted by an idempotent function f, where f(G, …) = G'. World models have equality, where G_a=G_b implies f(G_a, …) = f(G_b, …). Subsets of the world model are denoted H and imply the set of all information relevant to a problem at hand.

For a code reference, see `BaseWorldModel` in `wtp/world_model_def.py`.

Action

An action is an ordered sequence of steps where each step is a piece of information indicating a change to a world model, or an execution of an external activity.

Actions are denoted as X. For an action X_a with n steps, the steps are denoted x_{a1}, x_{a2}, x_{a3}, …, x_{an}. Actions are applied to a world model G by $f_{act}(G, X) = G'$. Given an established constant set of symbols S, one for each type of procedure a step may execute, and a fixed list of parameter sets P_1, P_2, … P_n defined for the set S, $x \in S \times P_1 \times P_2 \times \ldots \times P_n$.

Any action X and any step x is readable and modifiable by the runtime of a compliant program. The set of step symbols S and parameter sets P_i are readable but not modifiable.

For a code reference, see `BaseAction` and `BaseStep` in `wtp/action_def.py`.

Agent

An agent is something that may instigate a change in the world model or external world. Agents may be intelligent actors attempting to reach an objective or systems following a set of deterministic rules. For instance, both a human chess player and the local weather could be considered agents. Agents have sets of actions they may execute. A complaint program has at least one implicit agent with the objective to guide its learning process and determine how to respond to external stimuli.

Agents in a world model are readable and modifiable by the runtime of a compliant program. Modification and creation of agents would be relatively rare, and generally only in situations where a complaint program is learning brand new information.

Two of the goals of a compliant program are to create reasonably accurate representations of agents and execute its own goals as an agent. It is not necessary, or even possible, to accurately predict the behavior of agents. To develop a good representation, a simple statistical or rules-based model may suffice.

For a code reference, see `BaseAgent` in `wtp/agent_def.py`.

Constraint

A constraint is a piece of information describing a Boolean evaluation of a world model with respect to an action step. If a constraint is evaluated as false during the predicted execution of an action, that action should not be taken. For instance, a set of constraints would be the rules in a game of chess, or a set of rules of thumb in evaluating a strategy.

Constraints are both readable and modifiable by the runtime of a compliant program. Some constraints would rarely be modifiable, such as the rules of a boardgame. Others may be frequently created and changed, such as internal rules governing heuristics to improve the execution of a task.

Metric

A metric is a set of information describing a computable function of a world state and returns a numerical value. Metrics are denoted m, and are evaluated against a world model G by M(G, m). For two world models G_a and G_b and one metric m_1, if $G_a = G_b$ then $M(G_a, m_1) = M(G_b, m_1)$. Evaluating a by itself metric should not change a world state.

Metrics are both readable and modifiable by the runtime of a compliant program. Their evaluation may be triggered by an action.

Discovery

Discovery is the process of a compliant program executing actions with the objective to learn more about its environment. This may be triggered in response to executing a task where there is not enough information in the world model to establish how any action can lead to reaching a desired objective. Discovery may also be triggered at the complaint program's discretion as a general self-improvement directive, and not necessarily in response to an immediate task to accomplish.

Discovery itself is not reinforcement learning, although how and when it is triggered may be done with an implementation-specific machine learning algorithm.

Diagnostic Reasoning

Diagnostic reasoning is the ability to deduce the source of inconsistencies between an expected and transformed world state. Given an initial problem world state G_i, expected world state G_e, action X_i to act upon G_i, and a metric m, assert that $M(f_{act}(G_i, X_i), m) = M(G_e, m)$. If this assertion fails, then diagnostic reasoning is the ability to identify the smallest set of information $H_i \subset G_i$ and action X_o where $f_{act}(H_o, X_o) = H_i$ and $M(H_o, m) = M(G_e, m)$ for some reference world state H_o.

The end goal of finding good candidates for H_i and X_o is to narrow down the search space during learning or adapting to a task. A common example is learning to solve a new puzzle that is similar to a previously solved one. The metric m is a Boolean value indicating the puzzle has been solved. G_i has a representation of the current puzzle, and H_o is a recently solved puzzle that is very similar. X_i is a technique that previously solved H_o, and the WTP program

expects to work for G_i as well. So $M(G_e, m) = 1$. However the potential solution $f_{act}(G_i, X_i)$ does not solve the problem in this case. So the WTP program is using diagnostic reasoning if it can identify a transformation X_o that indicates a difference between the previous puzzle and this one that potentially explains why X_i does not succeed. Ideally X_o contains information about how to improve or generalize X_i to solve the puzzle in G_i, but that is outside of the scope of diagnostic reasoning.

Imperative Learning

Imperative learning is a type of learning where a sequence of procedural instructions and assertions are given to a WTP program to replicate. This is similar to how learning is handled in a traditional classroom or workshop setting. The WTP does not need to create brand new internal models from scratch to learn how to accomplish a task. Rather, imperative learning is about rapidly internalizing a given explanation of how to accomplish the task, and then fine tuning the internalized model during following testing against various assertions.

The focus of a WTP program during imperative learning is to avoid needing to iterate over solutions as much as possible, to identify stumbling blocks or inconsistencies in an internal model, and to build effective layers of abstraction. Let Y be a data input of a sequence of input instructions and assertions. Then imperative learning implies the generation of an internal action X_y and Boolean assertion check m_y, where. If not, the WTP program should apply diagnostic reasoning to identify any discrepancies to adjust X. If there is an external agent generating Y, a summary of the results of the diagnostic reasoning can be given to the agent to adjust Y accordingly.

Further reinforcement learning may be applied to improve X_y with respect to other metrics while applying the constraint $M(f_{act}(G, X_y), m_y)=1$.

Declarative Learning

Declarative learning is a type of learning where a sequence of procedural instructions and assertions are given to a WTP program to define an objective to meet. However, unlike imperative learning, the WTP program is not given a well-defined set of instructions on how to meet that objective.

Let Y be a data input of a sequence of input instructions and assertions. Let X_y be an action generated from Y, and m_y be the Boolean assertion check of a successful solution. Then declarative learning implies the iterative creation of a potential solution X_i where $M(f_{act}(f_{act}(G, X_i), X_y), m_y) = 1$.

X_y may be generated through imperative learning. Further reinforcement learning may be applied to improve X_i, while holding the constraint $M(f_{act}(f_{act}(G, X_i), X_y), m_y) = 1$. Multiple rounds of imperative and declarative learning may be used to assist a WTP program in building internal models of abstraction.

An example is teaching a WTP program a board game. Y is the information about the rules of the game and the winning condition. X_y is the action that computes the status of the game, and m_y is the metric indicating the game is being played properly. X_i is the strategy generated by the WTP program to play the game, and once a valid strategy is generated it can be improved upon to maximize the likelihood of succeeding. Follow up rounds of learning can impart existing strategies for the WTP program to use at its discretion.

Meta Model

A meta model is a set of code and information used to edit, read, and store information about the WTP's operations in the world model. For instance, operational data can include serialized data about agents and actions.

Directive

A directive is the combination of an objective to meet, along with a set of actions that may achieve that objective in a sequence of execution. A WTP program may be given a directive and must generate the sequence of actions that best meets the objective. Objectives may be a condition, or a metric to optimize for.

Invariance

An action X of a world state G is invariant with respect to a metric m if $M(G, m) = M(f_{act}(G, X), m)$. Given that actions are stored in G, transformations on actions can also be invariants.

Abstraction

An abstraction of an action X is the construction of a new action X_a such that it can achieve the objectives set out during the creation of X while meeting new objectives. Given a condition metric m_c, generating X_a is invariant with respect to m. X_a may be generated by combining the conditional execution of X with other actions, or by copying and modifying X.

A naïve approach to creating abstractions is to create random perturbations X_p of the world state, and find changes to X to generate X_a such that the likelihood that $M(f_{act}(G, X_a), m_c) = M(f_{act}(f_{act}(G, X_p), X_a), m_c)$. A more sophisticated approach would be to use information about the problem at hand to parameterize action steps in X, or to include information about actions used in similar problems.

Characteristic

A characteristic is a metric generated from a world model that may be used in the generation of abstractions. Characteristics are enumerable and well ordered. So given two characteristics C_a and C_b, either $C_a < C_b$, $C_a = C_b$, or $C_a > C_b$. The cardinality of characteristics is a rough approximation of the likelihood of being used in generating useful abstractions for a given problem at hand.

A list of characteristics may be generated by diagnostic reasoning, by iterating over combinations of properties in an internal model, or by a custom action a WTP has developed.

WTP Formal Criteria

<incrementally learn information>

Starting with Directives

The core loop of a WTP program is the execution of directives. In this situation, a set of potential actions is given alongside an objective to meet. With these given, the WTP program must identify which sequence of actions best meet the objective criteria.

<Knowledge Data Store>

<Action Space Collection for a Problem>

<Refinement>

<Discovery>

<Indirect objectives>