Constructivist Robots Assembling Value-Based Decision Heuristics

Master Thesis by

Tyler Olson

In order to obtain the degree of Master of Science in Robotics
at Delft University of Technology,
To be defended publicly on Thursday August 28, 2025 at 2:00PM.

Student Number:	5946182
Faculty:	Mechanical Engineering
Thesis committee:	Dr. ir. Carlos Hernández Corbato (Supervisor)
	Dr. Alex Gabriel (Daily supervisor)
	Dr. Sebastian Dumančić
	Dr. Alexandra Kirsch
	Dr. Chris Pek

Constructivist Robots Assembling Value-Based Decision Heuristics

Tyler Olson
Cognitive Robotics, Delft University of Technology, Postbus 5, 2600 AA Delft, The Netherlands

Abstract

Modern robots make decisions in many ways, but rely on their designers to choose which strategies to employ and when. Adopting a perspective of bounded rationality from the cognitive sciences, we develop a definition of decision making for constructivist robots to formulate their own decisions based on their mission-specific values. Our model, extending Kirsch (2019), defines decision making with an iterative algorithm that captures a broad range of possible strategies and problem domains. Briefly, a set of decision alternatives are first assessed by a set of relevant cues. These assessments are then aggregated into a multi-dimensional evaluation of each alternative from which a preference ordering is created. Finally, based on the problem specification, a set of chosen alternatives is either accepted by a stopping rule or a new iteration is started, updating the sets of alternatives and cues. If no alternatives or cues remain after iterating, then the decision fails. Given this algorithm, we implement a toolbox of decision-making components as modules in a cognitive robot architecture and demonstrate a method of assembling them into complete decision strategies, represented as behavior trees, using an automated theorem prover. For proof of concept, we simulate three example strategies used by a domestic robot performing a search and retrieval task. We discuss new insights into designing and selecting decision-making strategies and make recommendations on how our proof of concept can be improved.

1. Introduction

In a letter to a British physicist, Benjamin Franklin described his *moral algebra*, a formula for deciding between two morally challenging options (Franklin, 1772/1987, in Gigerenzer et al., 1999, p. 76). In the centuries since, psychologists, philosophers, business leaders, and computer scientists alike have endeavored to describe how decisions are made in the human mind and how they can best be made by robots, organizations, or even entire populations. Decision-making methods effective for thinking individuals or teams of analysts are typically ear-marked with a requirement that human intuition accompany each step of the process to continuously evaluate and reevaluate the method. Unfortunately, this footnote has been difficult to reproduce in robotics or ignored entirely in favor of constraining the environments and problems modern robots solve.

Without the luxury of the deeper intuition employed by human decision makers, decision mechanisms in robotics and artificial intelligence (AI) are prone to spurious failures when they encounter unforeseen circumstances. Robots today can perceive and classify objects in their environment, but they do not perceive the systems giving meaning to those objects. Robots do not understand themselves or other agents as systems within the environment or how those systems relate to the values of their designers. Modern robots often have the implicit assumption that the models they use for understanding the world are complete. These systems cannot make sense of vagueness and must be confined to what Gigerenzer et al. (1999) call "small worlds" that can be nearly completely captured by the agent's knowledge.

For robots to break out of the lab into open uncertain worlds, they must adaptively make decisions in the context of their current situation. In these open environments, robot designers often cannot foresee all the problems their robots will encounter so are unable to instill in them a complete understanding. Contrary to *constructionist* architectures that are crafted and finely tuned by engineers, Thórisson et al. (2016) argue that *constructivist* mechanisms which are self-organizing or self-programming are better suited to open worlds because they can deepen their understanding automatically. Providing robots with the tools they need to think about their mission, their own capabilities, and their environment can enable them to break out of their small worlds.

Taking inspiration from the cognitive sciences, this work proposes that giving robotic agents an adaptive toolbox of decision strategies can enable them to formulate their decisions on their own and consider their choices according to their mission-specific values. State-of-the-art model-free deep learning methods are able to make excellent predictions for well-defined tasks but are resource-hungry, lack transparency, and are expensive to train (Sridharan, 2025). On the other hand, symbolic methods tend to be less resource-intensive at runtime and make more explainable decisions; however, they can be subject to a combinatorial explosion when attempting to capture the vast frontier of potential choices and consequences of even simple environments. In constructionist architectures, an engineer decides when and where to use these different strategies, making tradeoffs in an attempt to maximize the value the robot provides. To adapt to dynamic open environments, a constructivist agent needs to be able change how it makes decisions according to the problems it faces.

The objective of this research is to demonstrate how a robot can make ecologically rational decisions by assembling decision strategies from an adaptive toolbox of components in a constructivist cognitive architecture. Explicit components with a precisely designed purpose are superior in robot architectures because they are typically easier to compose, modify, add features to, and model (Scheutz and Andronache, 2004). Accordingly, we develop an assemblable model of decision making, largely inspired by Kirsch (2019), and formulate the decision-making process using this model as a set of independent modules in the **CoreSense** cognitive robot architecture (Gabriel et al., 2025). This approach encourages code reuse, makes decisions of robotic systems more explainable, and facilitates fine-grained self-adaptation through improved self-understanding.

The goal of **CoreSense**, a constructivist framework built on top of ROS 2 (Macenski et al., 2022), is to enable a robot to use its understanding of itself and the environment to maintain situational awareness, adapt itself according to its goals, and act according to its values. Drawing a distinction between *cognitive functions* and *cognitive capabilities*, Gabriel et al. (2023b) organize the architecture into distinct modules that each perform a specific cognitive function. They offer these definitions:

> **Cognitive function**: a system behavioral mechanism that transforms inputs into outputs and is powered by knowledge-based components (the means view)
> **Cognitive capability**: a system capability provided by a cognitive function that enables the system to produce intentional action (the ends view)

This framing enables **CoreSense** modules to be strung together to form interpretable and adaptive *cognitive structures*, demonstrating new cognitive capabilities from the composition. With explicitly modeled components, a constructivist robot can recognize which of its cognitive functions realize a desired cognitive capability.

1.1 Document Preview

In Section 2 we motivate our model with background on decision-making theory, bounded rationality, and heuristic decision making, motivating our methods. In Section 3 we introduce how to conceptualize decision problems then develop a model of the cognitive capability of decision making in Section 4. We describe a method of assembling decision strategies from a collection of components in Section 5. A suite of these components based on common heuristics from decision-making literature are detailed in Appendix B. We demonstrate a proof of concept in Section 6, analyzing three example decision strategies used by a simulated domestic robot searching for a book in an apartment. In Section 7, we discuss potential criteria for choosing heuristic components, observations of heuristic boundedness by how cues and alternatives are updated each decision iteration, and some of the limitations of assembling heuristics with automated theorem provers. Finally, we present our conclusions, identify some strengths and weaknesses of our model, and made some recommendations for future works to address them in Section 8.

2. Background

An agent with multiple decision strategies must answer the question: *How can I select the best solution(s) when facing a particular problem?* Katsikopoulos et al. (2018) divides decision problems into three major classes. In *inference* problems, a decision maker attempts to determine the appropriate category of some object or which object(s) in a collection best fit some desired category. *Preference* decisions are one-off strategic problems where the correct decision is not objectively known but based on the decision maker's internal preferences. *Forecasting* problems deal with predicting a future value of some decision alternative. For example, given a set of blocks to select, a robot may make an inference decision about which block is closest, a preference decision on which is its favorite color, or a forecasting decision about which block is most likely to earn it a high reward.

The ultimate objective of value-based decision making is to facilitate an agent to select the solution(s) which best align with its combination of potentially conflicting values. These *values* could be abstract desires like timeliness, beauty, or human safety or they could be concrete goals like getting a coffee, navigating to a location, or winning a game. Haan and Heer (2012) describe decision *alternatives* as any potential solution that is capable of effecting an agent's values. Alternatives can be any set of options to select from and may be given beforehand or retrieved from memory by the decision maker (Kirsch, 2019). To make a decision, an agent uses *cues*, also called features in artificial intelligence, predictors in statistics, or attributes in decision analysis, to evaluate each potential alternative (Katsikopoulos et al., 2018). How an agent selects and combines these cues differentiates its strategies for decision making.

2.1 Ecological rationality

The traditionally accepted theory of rational choice posits that “good” choices are those which could have been arrived at by deductively reasoning which alternatives are most likely to achieve the agent’s goals from all of the available information (Todd and Gigerenzer, 2000a). From this perspective, choices are judged using *coherence* criteria which measure how internally consistent decision strategies are. Robots using these criteria utilize logical soundness, expected value theory, and probability laws to assess the quality of decisions.

In contrast Gigerenzer et al. (1999) promote the perspective of *ecological rationality* in which the “goodness” of choices is based on external outcomes and takes into account the availability of information, processing power, and deliberation time. Measures of how choices correlate to positive external outcomes, called *correspondence* criteria, could include accuracy, frugal use of information, or speed of the decision process. Agents displaying ecologically rational behavior adapt their decision strategies according to the structure of information in the environment making choices based on correspondence criteria.

Trading coherence criteria of “good” choices for correspondence criteria allows agents to make choices that are seemingly self-conflicting (Gigerenzer et al., 1999). However, while an agent’s decisions can simultaneously exhibit both coherence and correspondence, for many problems the decision maker does not need to make consistent choices. In some decision problems being consistent is unimportant, like choosing what color shirt to wear with jeans, or even detrimental, such as always making the same predictable opening move in a game of chess (Todd and Gigerenzer, 2000a; Gigerenzer et al., 1999). We take the stance that robots should make choices that best correspond with the values of their operators. A robot can provide this value by achieving concrete goals their operator has or by bringing about their preferred states of the world. This means that robots should employ decision strategies based on correspondence criteria, even if these strategies are inconsistent.

Following Kirsch (2019), we will use the term *heuristic* to refer to any computational strategy used to make a decision, regardless of if these strategies exhibit coherence or correspondence. Gigerenzer (2001) frames heuristics as compositions of three kinds of rules: *search rules* which search memory for relevant cues and alternatives, *stopping rules* which halt the search, and *decision rules* which determine how to make a selection from the alternatives. This three-rule model assumes memory mechanisms can be used independently of decision making but cognitive architectures like Soar (Laird, 2012) and ACT-R (Anderson et al., 2004) model the whole process as interdependent (Baguley and Robertson, 2000). Kirsch (2019) models decision heuristics as recursive algorithms with interfaces specified for a series of subroutines based on the *Multiple Regression* (Payne et al., 1993) and *QuickEst* (Hertwig et al., 1999) heuristics. As a unifying model, Kirsch’s algorithm stands out as a promising candidate for modeling decision making in robots. We use their work as inspiration for an assemblable model of decision making, exploring many of the similarities and differences in Section 4.

2.2 Beyond the Multiple Regression heuristic

Perhaps the most frequently used heuristic in AI is *multiple regression*, also called the *Weighted Additive Rule* (WADD) or Multiple Criteria Decision Making in decision-making and operations research (Payne et al., 1993; Shah and Oppenheimer, 2008; Todd and Gigerenzer, 2000b). This model has been used to describe human decision processes, to make automated decisions and provide decision support, and to evaluate processes against. In WADD a single utility score is calculated for each given alternative by taking a weighted sum of the scores it receives from each identified cue. The alternative with the highest utility is selected. Benjamin Franklin's moral algebra, mentioned in the introduction, is an early description of multiple regression.

Unfortunately, many psychologists, strategists, and philosophers have co-opted multiple regression as a definition of rational decision making, rather than one possible strategy for doing so. For example, Haan and Heer (2012) define "rationalizing" as determining the merits of each alternative and comparing them on a well-balanced set of criteria versus "optimizing" which focuses heavily on one or a small number of very specific criteria. Multiple regression follows the *principle of total evidence* which posits that in order to make the best decisions, agents must use **all** of the available and relevant information (Gigerenzer and Brighton, 2009). This perspective of rationality and the ease of formalizing multiple regression methods has lead to their wide adoption in robotics. This is no surprise: examining how humans learn to select heuristics, Rieskamp and Otto (2006) observed that participants initially preferred WADD regardless of its suitability to the environment in three of four of their experiments. Even when monetary costs were higher for using WADD, participants on average searched for too much information but did eventually learn to favor a more ecologically rational heuristic.

The multiple regression heuristic assumes that the deciding agent exists in a closed world where it knows of all possible alternatives and cues and has a perfect predictive model for calculating the expected utilities of each alternative (Kirsch, 2019). While these assumptions are valid for some environments, they are not always appropriate for robots operating in large, open, and ill-defined worlds. Lacking a guiding intuition that humans use to make minor corrections as they solve problems, modern robots are not capable of recognizing when their assumptions no longer apply, when corrections need to me made, or when their decision heuristic is not ecologically rational. Multiple regression is ill-suited to making decisions when a robot has *incommensurable* values that are difficult or impossible to combine into a single utility. For example the value of a human life is difficult at best to place a monetary value on. This can also be observed in the difficulty of tuning the weights of different cost-map layers for robot navigation.

The principle of total evidence assumed by multiple regression suggests the existence of an accuracy-effort tradeoff where suboptimal decisions can be made using less effort or information (Gigerenzer et al., 1999). Algorithms exhibiting *optimization under constraints* explicitly measure trade-offs between exploration and exploitation, stopping computation when the expected costs outweigh the expected benefits. Constrained optimization algorithms common in robotics include Monte Carlo Tree Search for task planning, Adaptive Monte Carlo Localization for navigation and localization, and Model Predictive Control for a variety of system control functions.

2.3 Bounded rationality

In many open environments the amount of information available vastly outweighs a robot's ability to capture and process it all. To make an optimal decision, "rational" decision processes cannot take into account all of the potentially relevant information in the required amount of time. An inability to determine what information is relevant for a particular situation, known in philosophy as the *frame problem*, leaves perfectly "rational" robots with the gargantuan task of interpreting all possible data available before making a choice. The frame problem affects heuristics that have unbounded information search rules including exhaustive search algorithms of computer science and optimization under constraints.

In contrast the principle of total evidence, *bounded rationality* takes the view that choices can be "good enough" without taking all available information into account. Todd and Gigerenzer (2000a) offer that to solve the frame problem, an agent can simply use a heuristic that is "good enough" which allows it to escape the analysis paralysis required for making an optimal decision. Herbert Simon famously coined the term *satisficing* to describe agents that make choices by evaluating options sequentially until a certain level of aspiration is achieved (Simon, 1956). These methods are commonly used by humans when they do not know all of the possible options in advance such as house or clothes shopping. Satisficing methods are ubiquitous in robotics: any algorithm which uses an error threshold to stop a solution procedure can be described as satisficing. Frequently encountered examples include gradient descent methods, Bayesian networks, and Maes' behavior nets (Maes, 1989). While satisficing methods do not require measuring the exploration-exploitation tradeoff that constrained optimization does, Gigerenzer et al. (1999) are quick to point out that they may require significant deliberation to set an aspiration level or to compare options to that aspiration level.

Gigerenzer et al. (1999) promote *fast and frugal* decision strategies which search for "good enough" choices but restrict the amount of information used to evaluate each choice. Heuristics are considered "fast" if they do not involve much computation, such as ignoring dependencies between cues, skipping evaluation of long causal chains, or using simple selection criteria. They are considered "frugal" if they robustly make good decisions while using the available information sparingly. These heuristics use both bounded search and bounded stopping rules. Because fast and frugal heuristics tend to be simple, they can be more easily combined, adapted, and applied to many different kinds of problems than complex procedures (Todd and Gigerenzer, 2000b). A detailed background on fast and frugal heuristics is provided in Appendix A.

Heuristics that ignore information can take advantage of the *flat maximum* effect in environments with diminishing returns on solution quality, producing optimal or nearly optimal solutions with much less effort. In a series of 20 experiments, Czerlinski et al. (1999) demonstrated *less-is-more* effects in some environments where ignoring information actually increased predictive accuracy of decision heuristics. By ignoring information, heuristics can resist overfitting by compensating for a small increase in error from bias with a larger decrease in error from variance. These simple heuristics generally perform better than complex ones in "difficult" problems where information is not of high quality, or not enough information is available to draw reliable conclusions (Katsikopoulos et al., 2018).

2.4 Assembling heuristics

From the ecological rationality perspective, no heuristic is inherently good or bad, but only more or less useful in particular environments (Todd and Gigerenzer, 2000b). This tradeoff between heuristics leads to what Shanks and Lagnado (2000) describe as the *selection problem* where it is not often clear *a priori* what the best heuristic is for a given decision. The selection problem supposes that in order to make the best decisions, agents must first select the best heuristic for their given problem, but then an agent must also decide on how to make that decision too, etc. This inescapability of the selection problem is also known as *infinite regress* (Feeney, 2000).

The selection problem can be solved similar to the frame problem by dropping the assumption that an agent must optimize its heuristic selection procedure. At some level of meta-decision, an agent can simply use a meta-heuristic which makes "good enough" guesses as to which decision strategy to use for the decision problem of interest (Todd and Gigerenzer, 2000a). Payne et al. (1993) and Minsky (1988) popularized models of cognition where agents use a variety of different strategies to make decisions depending on the topic, effort, and accuracy demanded for the decision problem. Strategy Selection Learning (SSL) (Rieskamp and Otto, 2006) is an unsupervised reinforcement learning model of how humans learn which strategies result in the highest monetary reward for solving binary inference tasks. Payne et al. (1993) describe a cost-benefit approach where a heuristic's costs are weighed against its benefits. These costs could be computational effort, resource loss or consumption, or tertiary negative consequences from wrong decisions. Benefits could be high accuracy, resource gain, or social acceptability. Exemplar-based models memorize which heuristics were correlated with positive outcomes in similar past situations and use the strategy with the highest probability of making a good choice (Juslin et al., 2003).

These existing methods are insufficient for a constructivist robot in an open world that needs to solve novel decision problems. Because each decision has specific constraints and requirements independent of the class of alternatives or cues used to evaluate them, there is a vast array of possible problems that could be encountered. Providing robots with a mechanism to assemble new heuristics enables them to solve a broader class of decision problems than if they only used a library of pre-existing ones. By providing a robot with some base heuristic component selection mechanism, it can use this to assemble "good enough" decision heuristics for novel decisions it encounters. This creates a stopping rule for the meta-decision of which heuristic to use. Like other heuristics, this meta-heuristic could be refined through experience, or have been constructed by some external process (e.g. evolution in humans or the engineers designing robotic agents).

Gigerenzer (2001) uses the metaphor of a "backwoods mechanic" assembling heuristics from a toolbox of search, stopping, and decision rules which may not be perfect for the decision problem at hand, but work well enough to provide solutions. These *exaptive* strategies can be seen throughout evolution where organisms take some organ or capability they already have and apply it to a new task, potentially with minor modifications, a strategy copied by evolutionary algorithms in machine learning (Wimsatt, 2000). Assembling heuristics also leverages the explosion in the number of possible combinations of individual heuristic components to take better advantage of different environment information structures.

3. Decision problem conceptualization

To assemble a decision heuristic a robot must first encode all of the aspects of the decision problem of interest necessary to collect a *choice set* of potential alternatives, a *working set* of cues, and ensure that the decision matches the form of its desired solution. This encoding must identify enough aspects of the decision that the robot can reasonably expect to generate a valid heuristic. Ideally, it also contains enough information to generate a heuristic that can take advantage of the structure of the environment to save the agent effort. We adopt the problem conceptualization method of Haan and Heer (2012) and encode the problem into two distinct parts: a "gap" and a set of "dilemmas." The gap is a precise description of the difference between the robot's current and desired state of the world. Each alternative is then a potential solution to their problem that closes this gap. Dilemmas are things the agent values and does not want to sacrifice while solving the decision problem. Different alternatives may offer different tradeoffs between these dilemmas and to solve the problem an agent needs to decide between the alternatives based on these tradeoffs.

For example: *Suppose it is a hot day at the beach and you want some ice cream. You walk up to an ice cream stand and are presented with several different flavors to choose from. You only have 5 euros in your pocket that you brought to the beach.* The gap is the difference in the state of the world now where you do not have ice cream and the future world where you do. The alternatives are the different flavors of ice cream that you could purchase from the stand. Some dilemmas might include your preferences of different flavors of ice cream, any allergies you might have, and the different costs of the different ice cream options.

Haan and Heer (2012) argue that both a gap and at least one dilemma are necessary components of a decision problem. A gap without any dilemmas does not present a decision problem because any alternative is equally valid since there are no sacrifices to be made between them. In this case an agent can select any alternative and does not need to spend energy trying to decide between them. Suppose you have no food allergies, no flavor preferences, and find out all the ice cream is free, then there are no dilemmas preventing you from immediately solving your problem: any flavor is equally valuable so you may as well pick the first one and get to enjoying your ice cream. Similarly, faced with only dilemmas but no gap, the agent is in its currently desired state and thus does not need to change it. If you happen to already be holding some ice cream, then you should not attempt to buy more because there is no difference between the state of the world now and your desired state. In both of these environments, there is no decision problem to be solved.

3.0.1 Gap

The gap must capture the class of alternatives that could potentially solve the problem so that an agent can identify its options. Possible alternatives within this class may be immediately present when the decision problem is formulated or may need to be retrieved or generated by the agent. In the ice cream example, the class of alternatives includes all of the flavors of ice cream available at the stand. If you were instead at home making ice cream from scratch, you would need to retrieve possible ice cream flavors to consider from memory. The description of this class of alternatives could vary in the complexity of inclusion and exclusion criteria. A robot must strike a balance between the costs of evaluating fewer but more specific alternatives and more but less specific ones.

The gap must also capture the shape of the solution, that is all of the properties a chosen set must have or could afford, independent of the alternatives present in the set. These properties could include bounds on the size of the chosen set, if a good enough solution can exist, and if so what would make it good enough. In order for an agent to assemble an appropriate heuristic, it needs to select heuristic components that ensure these requirements are met and can take advantage of affordances to select heuristic components that reduce effort.

3.0.2 Dilemmas

Completing the problem conceptualization, an agent must model the dilemmas alternatives may pose when closing the gap. Although a robot may select specific dilemmas to focus on for the decision problem at hand, any feature of an alternative that conflicts with the robot's values or impedes its mission represents a dilemma. This means that while each decision problem has a different gap, the set of dilemmas remains the same for any given mission. Cues perform the function of measuring these dilemmas for each alternative with respect to the others in the choice set. Through the decision process an agent combines all of these measurements to arrive at an overall value judgment for each alternative based on what sacrifices they forfeit.

Where the class description encoded in the gap specifies what it means to be a possible alternative, cues measure how much each potential alternative sacrifices when used as the solution to the decision problem. Each cue is then a precise measurement of the agent's perceived value of each alternative along some dimension. These measures define what makes the end state desirable, rather than the degree to which some specific solution was achieved. Because robots should make choices according to the values of their operators, these values need to modeled in their mission specification. Haan and Heer (2012) offer some insights into how to break down abstract mission values into specific measurable cues using a tool called goal trees.

For AI safety researchers alarm bells should be ringing as precisely communicating all of our human desires to machines has proven to be a non-trivial task. This is referred to as the *alignment problem*. A mission designer must carefully identify which of their values to model for a specific mission, which of these values are incommensurable, and how individual cues can measure them from perceptions of the world. Ultimately, we leave resolution of the alignment problem and the development of specific cues to the mission designer. Luckily, with heuristics assembled from toolbox components, the mechanism and cues used to make each decision are transparent and easy for an engineer to interpret. This explainability is a key motivation for deriving decisions from mission values.

4. Definition of Decision Making

In this section we formally define the cognitive capability of decision making by identifying a set of distinct cognitive functions such that the execution of a cognitive structure composed of each of these cognitive functions (and the required plumbing) is sufficient to produce a decision. Accordingly, we refer to a cognitive structure that affords the cognitive capability of decision making as a decision *heuristic* and to each constituent cognitive function as a *heuristic component*.[1] We

1. Kirsch (2019) calls each of these a "function," but we use the term "heuristic component" to avoid confusion.

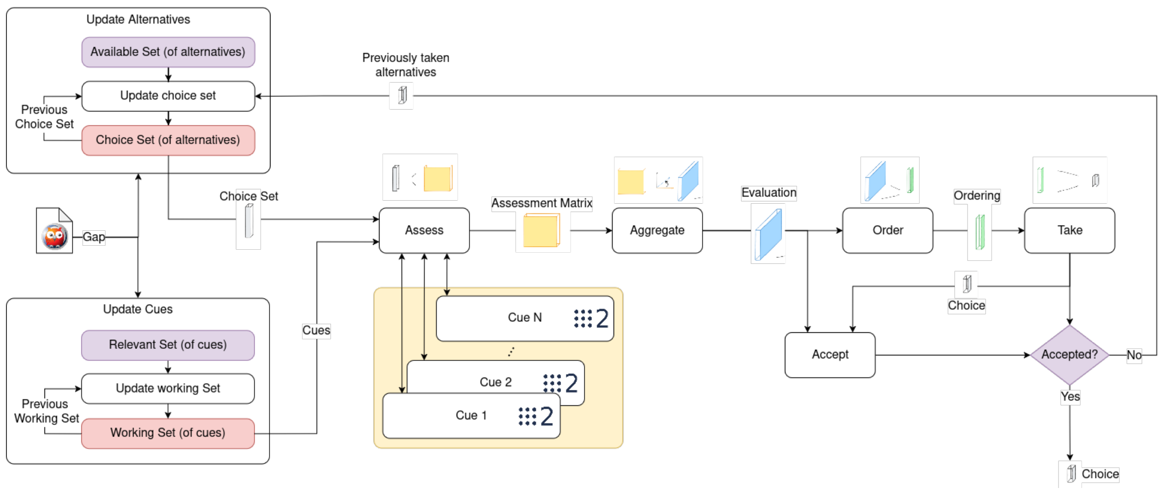

Figure 1. An iterative model of the cognitive capability of decision making.

will progressively isolate key requirements of decision making that separate it from other possible cognitive capabilities while capturing a wide variety of decision problems. We aim to leave as much room as possible for different implementations of the various heuristic components. Once we have developed this definition, we will use it to realize the iterative decision-making cognitive structure illustrated Figure 1).

Our approach is guided by the unifying model of Kirsch (2019) and shares many similarities. Katsikopoulos (2019) praises Kirsch's model for its pragmatic ability to describe decision making in complex open worlds. Kirsch (2017b) demonstrates a precursor to the model by simulating robots completing navigation tasks with multiple cooperating modules making different decisions. Throughout this section we will compare and contrast the cognitive functions we develop to the subroutines in the Kirsch algorithm.

Assume the robot provides a description of the decision problem, specifying the class of alternatives and required properties of the choice. Suppose there exists some set, $\mathbf{G}$, of all possible descriptions of a decision problem understandable by a robot agent where a gap $G \in \mathbf{G}$ describes a single decision problem of interest. We will assume the gap can be encoded into a finite string and that the robot making this decision is capable of interpreting the encoded specifications. Let A be the set of all possible alternatives to a given decision problem, according to the class of alternatives defined in the gap, and $\mathbf{A} = 2^A$ be the power set of A.

Lets start by formalizing the simplest definition of decision making: to choose a subset from a set of alternatives. By this definition, at bare minimum a cognitive structure exhibiting the cognitive capability of decision making must employ some cognitive function

$$\textsc{Decide} : \mathbf{A} \rightarrow \mathbf{A}$$

where any input $C \in \mathbf{A}$ is a choice set of alternatives, and any output $C' \subseteq C$ is a chosen set. This model of decision making may be useful from the outside, but it provides no hints at how the function should behave or that distinguish it from any other subset relation.

4.1 Accepting

Let us assume, like Kirsch (2019) and Gigerenzer et al. (1999), that decision making can be an iterative process. First, in order for a decision to be made by agents that are governed by bounded rationality (which includes humans and robots that make progress), this iteration needs to be stopped somehow. There are many ways of stopping an iterative (or recursive) process, but they all are governed by what Gigerenzer (2001) calls a stopping rule. This could be a satisficing condition: *Stop if a chosen set has been identified that is good enough*, a comparative condition: *Stop if a chosen set has been identified whose elements are strictly better than all the rest*, or a non-condition: *Always stop*. The common thread among these stopping rules is some boolean criteria measured on the chosen set. Using the terminology of Kirsch (2019), we will say that a decision process iterates until the chosen set has been *accepted*. Failure to find an acceptable chosen set is a failure to make a decision.

The decision process must also ensure that the chosen set meets all of the requirements laid out in the gap. A chosen set which does not meet all of these requirements is unacceptable to a decider and cannot be a final decision. While deciding on an acceptable choice could be accomplished by ruling out each of the unacceptable alternatives, ultimately the acceptable choice is the only one that matters. Imagine saying "Of all the available ice cream flavors, I have decided against mango." While this may be valuable understanding, it does not solve your initial problem of deciding which ice cream to purchase, only on deciding which not to.

Let us then add a first requirement that a chosen set C' must be accepted, and define a cognitive capability of acceptance. From now on we will refer to the boolean domain as $\mathbb{B} = \{\top, \bot\}$ where $\top$ (*verum*) represents true and $\bot$ (*falsum*) represents false. Let

$$\text{ACCEPT} : \mathbf{A} \rightarrow \mathbb{B}$$

be a cognitive function indicating if a chosen set C' has been accepted or rejected. Going forward we will describe a chosen set as having been accepted when $\text{ACCEPT}(C') = \top$ and an alternative as having been accepted when it is a member of and accepted chosen set, $a \in C' \wedge \text{ACCEPT}(C') = \top$. Note that there is currently some implied parameterization based on the agent's current understanding state needed to implement an ACCEPT component to reflect the specific desired stopping rule: this will be addressed more explicitly in Section 4.4.

Now we can define a *decision* as an accepted subset of the choice set, represented as the pair $(C', \top)$. This also means we must refine our model of the cognitive function of decision making.

$$\text{DECIDE} : \mathbf{A} \rightarrow (\mathbf{A}, \mathbb{B})$$

In the event this function returns a rejected choice, $(C', \bot)$, the decision process has failed.

4.2 Taking

The cognitive capability to accept or reject is predicated on the existence of a chosen set, and many decision problems place requirements on the size of this chosen set. Much of the literature on decision making only considers decision problems with a single chosen alternative (e.g. Czerlinski et al.

(1999); Kirsch (2019); Shah and Oppenheimer (2008)), however there are plenty of decision problems that require a different fixed, bounded, or even unbounded number of alternatives to be chosen. With fixed or bounded size requirements, there could be some alternatives in the choice set that are "better" than others, but some required number still must be taken. Consider one such decision problem: *choose the two most valuable from a set of coins, each of a different denomination.*

In some decision problems alternatives are ranked relative to each other within the context of the choice set and a change in the choice set could result in a different outcome of second place, third place, etc. In the United States, the president and vice president were elected as the candidates with the first and second most votes respectively until 1804 (U.S. Const. amend. XII). Consequently, the nation's two most important officials were often members of different political parties, leading to dysfunction in their administrations. Removing the winning candidate and holding a second election for vice president may have alleviated these frustrations, but most likely would have produced very different outcomes. After the 12th amendment, ballots contained pairs of president and vice president, reducing the size of the chosen set from two individuals to one pair. Take note of the notion of ranking alternatives; we will return to it in the next section.

Another requirement could be the fitness of the alternatives, where the chosen set must contain all of the alternatives that succeed a certain criteria and none that fail it. In these decision problems it is acceptable to select multiple fit alternatives, resulting in a chosen set of uniform fitness but unspecified size. Fitness requirements are leveraged by one-good-reason heuristics to assign inclusion or exclusion in the chosen set, stopping when only precisely one alternative is included and the rest are excluded. These can easily be generalized to decision problems with other size requirements.

Each of these decision problems about ice cream has a different combination of size and fitness requirements:

- "I want one scoop of ice cream, but I don't know which flavor I want."
- "I want two different scoops of ice cream, but I don't know which flavors."
- "I want up to three different scoops of ice cream, but I'll only take flavors I haven't tried before."
- "I want to try all of the new flavors of ice cream."
- "I want to at least try the flavor of the week, but maybe others as well."

While all requirements could be enforced in the ACCEPT component, constructing a new component that can incrementally build a chosen set enables a robot to include or exclude alternatives individually, stopping early if all the requirements are met. Early stopping is a mechanism many iterative heuristics use to reduce effort (see analysis by (Shah and Oppenheimer, 2008)). After all of the requirements from the decision problem are met by this new component, it can be composed with the ACCEPT component to accept or reject the whole chosen set producing a decision. Let

$$\textsc{Take} : \mathbf{A} \rightarrow \mathbf{A}$$

be a cognitive function where the input is a choice set of alternatives, and the output is a chosen set. Notice that this is identical to our original definition of DECIDE: this is no coincidence. However, to promote early stopping, we add the restriction that there must exist some well-defined indicator function $\mathbf{1}_{C'} : A \to \mathbb{B}$ where $\forall(C), (a \in \textsc{Take}(C) \implies \mathbf{1}_{C'}(a) = \top)$ such that it specifies inclusion or exclusion of each alternative into the chosen set. Applying this indicator function directly to each $a \in C$ gives TAKE linear computational complexity, $O(n)$, where n the size of the choice set.

Going forward we will refer to an alternative where $\mathbf{1}_{C'}(a) = \top$ as having been "taken." If a condition of the chosen set cannot be decomposed into conditions on individuals then it must be enforced in ACCEPT. TAKE is a generalization of a subroutine Kirsch (2019) defines, called FIRST, that takes a single "best" alternative from the choice set. However, FIRST pushes the enforcement of any fitness requirements to the ACCEPT function and is only applicable to decision problems requiring a chosen set with a size of exactly one.

Armed with TAKE and ACCEPT we can redefine DECIDE as their composition:

$$\textsc{Decide} \cong (\textsc{Accept} \circ \textsc{Take}).$$

In plain English: deciding is accepting a set of alternatives taken from an initial set of options.

4.3 Ordering

To be useful to the decision maker, DECIDE should express the agent's values so that the chosen alternatives are those it prefers for the decision problem. In this context preferred alternatives are those that the agent favors because it perceives the consequences of choosing them to result in a preferred world state. For inference and forecasting problems, an alternative that is the correct or most accurate answer is assumed to be preferred over incorrect or less accurate ones.

Preferences on alternatives in the choice set must be *strongly connected*, meaning any two alternatives are comparable. By definition, if some alternative was incomparable with another, it would be impossible to say which was preferred or even if they were of equal preference. Assuming these preferences are *transitive* removes the possibility of cyclical preferences which also leave the decision maker unclear on how to resolve them. These two conditions are fully satisfied by representing an agent's preferences with a *weak ordering*, also called a preference relation or total preorder. Let a weak ordering $a \lesssim b$ represent that alternative b is either preferred to alternative a or they are of equal preference ($a \lesssim b$ and $b \lesssim a$). Strong connectedness also implies *reflexivity* where any alternative $a \lesssim a$ is of equal preference to itself.

Even if a chosen set is acceptable, and even if all of the taken alternatives individually fulfill the requirements imposed by the decision problem, as currently specified our robot would have no way to measure or enforce that it is making "good" decisions, only ones that meet the decision problem requirements. Because TAKE considers alternatives for membership in the chosen set one-by-one, there is currently no mechanism to ensure that an alternative exists that was not taken but would have been preferred over those in the chosen set if only it were considered in a different order.

One way of fixing this problem is to drop the requirement of TAKE to evaluate alternatives incrementally and instead evaluate them all simultaneously. Preferred alternatives could be taken before less preferred ones, however each alternative would need to be compared to every other one to determine if it should be taken. While a possible implementation, this increases the computational complexity of any TAKE engine by a factor of $O(n)$. Instead, if supplied with a weak ordering on the choice set (equivalent to sorting all of the alternatives before taking them), then TAKE can retain its linear complexity and ability to stop early.

Let $\mathcal{L}_C$ be the set of all possible weak orderings of a choice set of alternatives C. Let us redefine the signature of TAKE as:

$$\text{TAKE} : (\mathbf{A}, \mathcal{L}_C) \to \mathbf{A}$$

where the inputs are the choice set, C, and a weak ordering on that choice set, $\lesssim_C \in \mathcal{L}_C$, and the output is the chosen set, C'. With access to the preference relation, TAKE can iteratively include alternatives starting with the most preferred and proceeding in order of preference until the decision problem restrictions are satisfied. Similarly it could iteratively eliminate alternatives starting with the least preferred and proceeding in the opposite order. Because there is a possibility of taking alternatives which may be tied in preference to not-taken alternatives, it may be advantageous for a decision heuristic to randomly select between all of the alternatives tied with the worst taken alternative until the size requirements are met. We implement this functionality in our two exemplar TAKE components in Appendix B.

In order to produce a pair of $(C, \lesssim_C)$ from the choice set C, we can introduce a new cognitive function,

$$\text{ORDER} : \mathbf{A} \to (\mathbf{A}, \mathcal{L}_C).$$

This ORDER component implements what Gigerenzer and Brighton (2009) refer to as a decision rule, enforcing the preferences of the decision maker based on the features of each alternative. In essence this component ranks all of the elements in the choice set, with the added information that some elements could be tied.

This new ORDER component, along with our new definition of TAKE, can be added to the composition defining DECIDE.

$$\text{DECIDE} \cong (\text{ACCEPT} \circ \text{TAKE} \circ \text{ORDER})$$

In plain English: deciding is accepting a set of alternatives preferred over all the others taken from an initial set of options.

4.4 Judging

The number of ways an agent could order alternatives changes for different kinds of decision problems. In inference problems, there is only one natural way to order alternatives based on the agent's knowledge, assuming it has a complete enough model to make eternally verifiable inferences. Similarly, for forecasting problems the number of ways of ordering alternatives is small, depending on the assumptions an agent makes about the future and how many different sets of assumptions it considers. For preference decisions, however, any ordering of any choice set could be valid, with any particular ordering associating a choice set to a different configuration of values. For robot agents, these preferences must be given to them by the mission designer for any possible class of alternatives. This implies that for any possible class of alternatives and any kind of decision problem (previously encountered or never before seen), the decision maker must have some ORDER engine fit for the job. This means that the number of individual ORDER engines required by a robot is at minimum the number of different classes of alternatives it is capable of understanding multiplied by the number of possible combinations of decision requirements. Clearly for all but the smallest worlds, this is intractable.

There are three major ways that implementing an adaptive agent to make all of these decisions can be made tenable. First is to recognize that according to our decision problem conceptualization, agents may posses some incommensurable measures of value that cannot be combined into a single utility. However, in order to create a preference ordering of any choice set the agent needs some way to compress all of its measures of value into a single ordinal rank for each alternative. Kirsch (2019) identifies computational social choice functions as a prime candidate for ordering alternatives based on incommensurable measures of value. A social choice function maps a set of *profiles*, ranked ballots of all alternatives based on some measure, into a single combined preference ranking. Voting methods like these are commonly encountered for behavior coordination in robotic cognitive architectures. Pirjanian (1999) identifies so-called *arbitration* methods such as Subsumption (Brooks, 1986), Maes behavior networks (Maes, 1989), and Discrete Event Systems (Košeckà and Bajcsy, 1994), that use different mechanisms to select one of a choice set of robot actions, focusing scarce system resources on tasks considered to be relevant. Note that while voters in social choice functions are typically assumed to be of equal weight, an agent may elect to weight some of its values more highly than others, (Kirsch, 2019).

Second is that the methods the agent uses to make these value tradeoffs may be defined separately of the measures of value themselves. An agent's values may change or it may be unclear in any particular situation how they should be combined. The *significance structure* of the environment may change creating different risks and gains associated with each value (Bullock and Todd, 1999). This means that a robot may need to use multiple different value-combination methods for different scenarios, missions, or alternatives. The particular method to select for a given problem must be something that is either derivable from the agent's knowledge, or chosen by the agent in another decision process. To avoid introducing infinite regress, we assume that the agent has some predefined mechanism for identifying this method.

Third, because these methods of combining values may not depend on the agent's particular value measures, alternatives can be evaluated according to each of the value measures separately instead of all at once. This increases the number of potential ways an agent can to calculate value tradeoffs by the number of values measures it uses. A good metaphor for this gain is using product types instead of sum types to describe a set of unique objects. An agent implementing value-agnostic combination mechanisms still needs to be able to assess each understandable class of alternatives along each of its value dimensions, but can then combine these assessments using a small set of parametrizable combination methods. We will address these assessments in the Section 4.5.2.

Given these insights, let us narrow the definition of the ORDER component, removing the responsibility of evaluating each alternative and focusing on the compression of incommensurable value judgments to order the alternatives in the choice set. We assume that any sets of commensurable value judgments that can be combined into a single representative value are aggregated together, leaving only mutually incommensurable value dimensions. In the ice cream example, value measures of independent color channels of each flavor can be combined into one overall value measure of color and value measures of the different aspects of taste can be combined together, but the two aggregate values are mutually incommensurable so they need to be compressed by the ORDER component. We will detail this aggregation process in Section 4.5.3.

Suppose there exists some set of k incommensurable measures of value defined for the agent, $V = \{v_1, v_2, \ldots, v_k\}$, where each measure takes the form $v_i : A \to \mathbb{R}$ and evaluates alternatives along a single, specific value dimension. Let a value *judgment* be a row vector of scores of a single alternative along each value dimension.

$$J_a = \begin{pmatrix} v_1(a) \\ v_2(a) \\ \vdots \\ v_m(a) \end{pmatrix}^T$$

Let an *evaluation* of a choice set, $\mathcal{E}_C$, be a matrix of size n alternatives by k value measures where each row is a value judgment of a different alternative in the choice set.

$$\mathcal{E}_C = \begin{bmatrix} v_1(a_1) & v_2(a_1) & \cdots & v_k(a_1) \\ v_1(a_2) & v_2(a_2) & \cdots & v_k(a_2) \\ \vdots & \vdots & \ddots & \vdots \\ v_1(a_n) & v_2(a_n) & \cdots & v_k(a_n) \end{bmatrix}$$

Let ORDER now be the cognitive function:

$$\text{ORDER} : \mathbb{R}^{n,k} \to \mathcal{L}_\mathcal{C}$$

which takes as input an evaluation of a choice set, $\mathcal{E}_C$, and produces as output a weak ordering of the choice set, $\lesssim_C \in \mathcal{L}_\mathcal{C}$.

To make a decision, we now require a new cognitive function which takes as input a choice set C, and produces as output a contextualized evaluation so that the alternatives in the choice set can be ordered. Let

$$\text{JUDGE} : \mathbf{A} \to \mathbb{R}^{n,k}$$

be a cognitive function which outputs an evaluation of a choice set $\mathcal{E}_C : C \in \mathbf{A}$ from an input choice set C. We call the cognitive capability afforded by this cognitive function *judging*, borrowing the term Shah and Oppenheimer (2008) use to describe the calculation of an overall value of each alternative. We can refer to the cognitive capability provided by the composition of the other three decision components as "choosing," implying that deciding is choosing based on an evaluation of the alternatives.

4.4.1 Accepting using an evaluation

For practical purposes, it may be necessary to stop attempting to decide or change the decision strategy if a decision has not been reached after expending some amount of resources. Kirsch recommends including guards (e.g. *Stop if too many iterations have passed*) as acceptance criteria. However, because these criteria do not operate based on the agent's values, but instead measure other external factors, we consider them separately. To express this restriction, we add the requirement that any ACCEPT engine may only be conditioned on the evaluation of the choice set $\mathcal{E}_C$. This can be enforced by refining its signature to explicitly include this evaluation and the chosen set, and discarding our previous assumption that the component used in the decision heuristic is implicitly parameterized by the current understanding state.

$$\textsc{Accept} : (\mathbf{A}, \mathbb{R}^{n,k}) \to \mathbb{B}$$

Now the definition of any ACCEPT engine must be predetermined before the decision is attempted.

With our new component definitions,

$$\text{with } \mathit{accepted} = \textsc{Accept}(\textsc{Take}(\textsc{Order}(C, \textsc{Judge}(C))), \textsc{Judge}(C)),$$

$$\textsc{Decide}(C) \cong \begin{cases} \textsc{Take}(\textsc{Order}(C, \textsc{Judge}(C))), & \text{if } \mathit{accepted}. \\ (\{\}, \bot), & \text{otherwise.} \end{cases}$$

In plain English: deciding is accepting a set of alternatives preferred over all the others taken from an initial set of options based on an evaluation of each alternative with respect to the decision maker's values.

4.5 Assessing and Aggregating

Our current definition of judging leaves open how an agent evaluates each alternative, only requiring that the commensurable measures of value are aggregated into incommensurable ones. With different decision problems, different features of the alternatives can be more or less relevant. Returning to our ice cream example, a customer deciding which flavor to buy is very different than the vendor deciding which flavors to sell for the day, despite the decisions being about the same set of alternatives. The customer and the vendor could even be the same individual on two different days. This illustrates that the particular goal that an agent has when making a decision can change how it values the same alternative, meaning that the specific value measures it uses must capture this nuance. As with ordering, if a robot uses methods for assessing these values that are defined separately from how it combines them, then it increases the number of unique ways at its disposal to make judgments with the same total number of cognitive functions.

Along the same line of reasoning as before, to reduce the number and complexity of its value measures, a mission designer can implement a small reusable set of simpler ones which a robot can combine for the decision problem at hand using one of a handful of value-agnostic combination mechanisms. Again, the particular method required for any given decision must be something that is either derivable from the agent's knowledge, or chosen by the agent in another decision process. We return to this in Section 4.6.2.

For some decision problems, it is difficult for an agent to ascribe an absolute value measure to each alternative, but given a choice set it is easy to measure alternatives against each other. It could be quite difficult to determine an absolute value of each flavor of ice cream that exists, but when comparing them side-by-side it is easy to identify which is better in some way than another. This is likely at some level due to the nature of the underlying mathematical structure: it only takes one way for a flavor to be better than another, but a vast amount of information is required to measure an ice cream flavor in its totality. For making a decision, an agent does not need to capture the complete value of any of the alternatives, but only how they relate to the others in the choice set.

Therefore, we introduce three new cognitive functions to realize the cognitive capability of judgment and make it tractable for assembling decision heuristics in robots:

- *Cues* - versatile cognitive functions of limited scope that measure one specific dimension of value,
- the ASSESS component - simply executes each cue, producing a multi-dimensional value measure of each alternative with respect to the choice set,
- and the AGGREGATE component - aggregates commensurable value measures into a judgment of multiple incommensurable value dimensions.

4.5.1 Cues

We can now formally define *cues* in our model. Suppose there exists some measure of value of each alternative in a choice set, $z : (\mathbf{A}, A) \to \mathbb{R}$ where the inputs are the choice set C and some alternative $a \in C$ and the output is a real-valued score. This measure could be absolute, or relative to the other alternatives in the choice set. Let an *assessment* be a column vector of scores of each alternative in a choice set according to the value measure z:

$$Z_C = \begin{pmatrix} z(C, a_1) \\ z(C, a_2) \\ \vdots \\ z(C, a_n) \end{pmatrix}.$$

We define a cue to be any cognitive structure implementing the cognitive function

$$\textsc{Cue} : \mathbf{A} \to \mathbb{R}^n$$

whose input is the choice set, C of size n, and output is an assessment of the choice set Z_C. We make the same assumption as Kirsch (2019), that cues could implement scoring measures, indicating some degree of perceived value of each alternative, or ranking measures which simply indicate a weak ordering between alternatives just as ORDER does. For convenience we assume that cues always encode higher value with a higher score and leave mapping from the agent's knowledge to these value measurements to the cue implementation. By this construction, any ranking cue can be converted into a scoring cue where each alternative is scored by the number of alternatives it is preferred over.

Cues could be implemented as boolean predicates, expected utility functions, or projections of sensory measurements into some abstract value space. Some cognitive architectures model internal factors such as artificial emotions, long term drives, or artificial personality traits for agents to gauge their decisions (Kotseruba and Tsotsos, 2020). Albus (1991) refer to these internal factors as "value state variables" that can be measured along a continuum (e.g. good-bad, pleasure-pain), overlaid on maps to direct robot attention to different points in space and time. Kotseruba and Tsotsos (2020) identify cognitive architectures used in robotics that learn expected utilities of various alternatives via reinforcement learning such as ICARUS (Choi and Langley, 2018), CLARION (Sun et al., 2016), Soar (Laird, 2012), and PRODIGY (Fink and Blythe, 2005). Some also use cues from self-observation to drive decision-making including CLARION, RALPH (Russell, 1991), and COGNET (Zachary et al., 1996).

Pragmatically, cues should be specific enough to be understandable to human designers, however it is common to represent them using hierarchical trees or graph structures. For instance, in the RCS architecture (Albus, 1991), value judgment nodes exchange information at different hierarchies, over different orders of magnitude in space, time, and level of abstraction. Graph Retrieval-Augmented Generation (Hu et al., 2025) is a cutting-edge generative AI method that uses cues spread over graph databases to help large language models make better value judgments of the relevance of different nodes in the graph.

As cues are cognitive structures that calculate value judgments for a set of alternatives, they could be composed from the heuristic components we have already identified: ASSESS, and some AGGREGATE engine that always produces one-dimensional value judgments. However, cues at some level of abstraction need to be defined in an agent's knowledge base by the mission designer. With cues as compositions of decision components, this conceptualization is necessary to avoid infinite regress.

The particular classes of alternatives each cue measures, how they relate to each decision problem, and how they are stored in memory are specific to the environment and mission of the robot, therefore we do not interrogate their structure further. Given that the field of cognitive science has identified cues as a useful abstraction, this seems to be a reasonable modeling choice.

4.5.2 Assessing

We now add a working set of cues $W = \{c_1, c_2, \ldots, c_m\}$ to the inputs of DECIDE, assuming that the agent has identified a relevant set of cues to attempt the current decision. Upon failure to accept a choice in a given iteration, the agent may use that information to update the working set of cues (and alternatives) and try again. Mechanisms for updating cues in the context of iterative decision heuristics are discussed in Section 4.6.2. Given this working set, we assume that an agent commits to assessing each cue on all the alternatives of the choice set for one iteration.

Let an *assessment matrix* be a matrix, M_C, of size n alternatives by m cues where each column is an assessment of the choice set C by a different cue in the working set. Abusing notation, we use $s_j(a_i)$ to denote the score $z(C, a_i)$ in the assessment Z_C produced by cue $c_j \in W$:

$$M_C = \begin{bmatrix} s_1(a_1) & s_2(a_1) & \cdots & s_m(a_1) \\ s_1(a_2) & s_2(a_2) & \cdots & s_m(a_2) \\ \vdots & \vdots & \ddots & \vdots \\ s_1(a_n) & s_2(a_n) & \cdots & s_m(a_n) \end{bmatrix}$$

Haan and Heer (2012) refer to this data structure an "impact table." Let the cognitive function

$$\text{ASSESS} : (\mathbf{A}, W) \to \mathbb{R}^{n,m}$$

produce the assessment matrix M_C of all input cues in W and alternatives in the choice set C. Kirsch (2019) calls this function `FillDecisionMatrix`, and the output a "decision matrix," however we have renamed each of these to avoid confusion. Note that we (inconsequentially) swap the rows an columns such that different alternatives are evaluated on different rows.

4.5.3 Aggregating

As previously stated, we assume that when making value judgments an agent aggregates sets of co-commensurate assessments from cues into a smaller set of incommensurable value dimensions. To do so, we introduce the cognitive function

$$\textsc{Aggregate} : \mathbb{R}^{n,m} \to \mathbb{R}^{n,k}$$

which takes as input an assessment matrix M_C and produces and evaluation $\mathcal{E}_C$.

While an assessment matrix looks very similar to an evaluation, we note three distinguishing differences. First, an assessment matrix has $m = |W|$ columns, one for each cue, but an evaluation has $k \leq m$ columns, one for each incommensurable value measure of the alternatives after aggregation. Second, an assessment matrix can be constructed by concatenating cue assessments $s_j(a_{1\dots n})$ across all alternatives in the choice set (column-wise), while an evaluation can be constructed by stacking judgments of single alternatives $v_{1\dots m}(a_i)$ across all incommensurable measures of value (row-wise). Third, where each assessment evaluates all alternatives in the context of the choice set in a single cue-specific dimension, each judgment evaluates each alternative independently of the others, indicating a multi-dimensional decision-specific value. We make this distinction because, according to our problem conceptualization, the value of an alternative to an agent is based on the state of the world in which that alternative is chosen as part of the decision, not in comparison to the other options.

A simple AGGREGATE component may just perform a single matrix multiplication, possibly weighing each commensurable cue by some contextual weight parameter. For example, suppose there are five color alternatives which an agent assesses with a different cue for each of the four RGBA color channels. However, for the current decision, the agent expresses preferences on colors by their intensity and opacity, aggregating each of the RGB assessments into judgments containing a single intensity value dimension that is incommensurable with the opacity value dimension (the A channel).

$$\overset{\text{Assessment Matrix}}{\begin{bmatrix} \text{🔴} & 255 & 0 & 0 & 0.5 \\ \text{🟢} & 0 & 255 & 0 & 1.0 \\ \text{🔵} & 0 & 0 & 255 & 0.5 \\ \text{⚪} & 128 & 128 & 6 & 0.3 \\ \text{🟣} & 199 & 71 & 235 & 0.9 \end{bmatrix}} * \begin{bmatrix} 1/3 & 0 \\ 1/3 & 0 \\ 1/3 & 0 \\ 0 & 1 \end{bmatrix} = \overset{\text{Evaluation}}{\begin{bmatrix} 85 & 0.5 \\ 85 & 1.0 \\ 85 & 0.5 \\ 87.3 & 0.3 \\ 168.3 & 0.9 \end{bmatrix}}$$

Notice that after aggregating, the first and third alternatives, 🔴 red and 🔵 blue respectfully, are indistinguishable so will receive an equal ordering no matter what ORDER component is used.

Many different aggregation methods are commonplace in robotics. Pirjanian (1999) identifies a class of behavior coordination mechanisms in robotics which they call *command fusion* mechanisms that combine the results from different behaviors (cues) into one command to be sent to a robot's joints and motors (alternatives). These behavior coordination mechanisms can be differentiated by the aggregation mechanisms they use. Some examples include DAMN (Rosenblatt, 1997) which produces one-dimensional judgments, its predecessor SAMBA (Riekki and Roning, 1997) which produces multi-dimensional judgments, and extensions of DAMN that use fuzzy logic (Yen and Pfluger, 1995; Saffiotti et al., 1995).

4.6 Completing the cognitive structure

With these three cognitive functions specified, we can complete our model. A cognitive structure which provides the cognitive function implemented in Algorithm 1 affords the cognitive capability of decision making.

```
Algorithm 1: The cognitive function of decision making
  Input: A choice set of alternatives, $C$
  A working set of cues, $W$
  Output: A decision, composed of a choice $C' \subseteq C$ and a boolean indicating if it was
          accepted or rejected
1 $M_C \leftarrow$ ASSESS$(C, W)$
2 $\mathcal{E}_C \leftarrow$ AGGREGATE$(M_C)$
3 $\lesssim_C \leftarrow$ ORDER$(\mathcal{E}_C)$
4 $C' \leftarrow$ TAKE$(C, \lesssim_C)$
5 if ACCEPT$(C', \mathcal{E}_C)$ then
6 |   return $(C', \top)$
7 else
8 |   return $(\{\}, \bot)$      /* The agent failed to make a decision. */
9 end
```

Adding two additional helper functions to update the alternatives and cues can help distinguish between different iterative heuristics. These two helper functions are parametrized by the gap G which, among other things, defines the class of alternatives and can be used in memory retrieval processes to collect all of the available alternatives and relevant cues. Figure 1 illustrates how each of the update functions constructs a choice set of alternatives or working set of cues from relevant objects retrieved from memory. Kirsch (2019) identifies that the memory retrieval processes for updating cues and updating alternatives may influence each other depending on how the memory structure works. Because we treat these two update steps separately, it may require implementers to cache the results of an interactive retrieval process and collect them in each independent step.

4.6.1 Updating Alternatives

Updating the choice set in real agents can be a complicated affair, as any alternatives that were not known at the start of the decision process can be expensive to discover or generate. However, in iterative heuristics, only considering a subset of the available alternatives, $C \subseteq A$, in each iteration can substantially reduce effort. For this reason, the way in which the choice set is constructed each iteration is an important difference between heuristics that is independent of how the available alternatives are retrieved from memory. Let

$$\text{UPDATEALTERNATIVES} : (\mathbf{G}, \mathbf{A}, \mathbf{A}) \to \mathbf{A}$$

implement the heuristic component updating alternatives, taking as input a gap defining the current decision problem, $G \in \mathbf{G}$, the previously used choice set of alternatives, $C^{-1} \subseteq \mathbf{A}$, and the previously taken choice, $C' \subseteq C^{-1}$, and outputting a new choice set $C \subseteq \mathbf{A}$. We assume that if this function returns the empty set $\{\}$, then there are no more viable alternatives and the agent has failed to make a decision.

In heuristics that reconsider alternatives in subsequent iterations, the agent must choose which alternatives to keep or eliminate from the choice set. Kirsch (2019) leaves this choice to the implementation of the UPDATEALTERNATIVES component, providing the evaluation computed in the previous iteration as an extra argument to the component. However, pulling this decision out of the UPDATEALTERNATIVES function avoids a circular dependency on the definition of making a decision to include another decision.

We recognize that choosing which alternatives to keep or eliminate based on an evaluation of the choice set is precisely the responsibility of the TAKE component. This brings back the idea of "deciding against" alternatives in order to reduce the difficulty of the decision problem in future iterations. Moving this responsibility from UPDATEALTERNATIVES to TAKE is another motivation for generalizing Kirsch's FIRST function to take multiple alternatives instead of just one. Evaluating if an alternative is worth further consideration is yet another dimension of value judgment, so any criteria used to update the choice set could also be calculated in the AGGREGATE function, potentially as an extra incommensurable dimension. This conveniently removes the need for UPDATEALTERNATIVES to take the results of the previous evaluation as an argument.

4.6.2 Updating Cues

Similar to updating alternatives, updating the working set of cues can require expensive memory retrieval processes but specifying how the relevant cues should be used in successive iterations is important to the functionality of iterative heuristics. Suppose there is a set of relevant cues the agent has retrieved from memory, $\mathcal{W}$, whose power set is denoted $\mathbf{W} = 2^{\mathcal{W}}$. Let

$$\text{UPDATECUES} : (\mathbf{G}, \mathbf{W}) \to \mathbf{W}$$

implement the heuristic component updating cues, taking as input a gap defining the current decision problem, $G \in \mathbf{G}$, and the previously used working set of cues, $W^{-1} \subseteq \mathcal{W}$, and producing a new working set $W \subseteq \mathcal{W}$ as output. We assume that if this function returns the empty set $\{\}$, then there are no more relevant cues and the agent has failed to make a decision.

Because not all relevant cues need to be added to the working set each iteration, an agent may make a meta-decision on which cues are the most relevant for the next iteration, a common tactic of one-good-reason heuristics. In this meta-decision, all available cues are now the meta-alternatives and meta-cues measure their relevancy for the decision problem at hand. The *take-the-best* heuristic uses a meta-cue call the *validity* which measures the true positive rate of each cue in making past predictions (Gigerenzer and Goldstein, 1996). Other example meta-cues include how recently a cue was used like *take-the-last* heuristic, or just selecting cues randomly like the *minimalist* heuristic (Gigerenzer et al., 1999). Contrary to UPDATEALTERNATIVES, this meta-decision within the UPDATECUES component is implementation specific and not definitional. It is distinct from the decision problem at hand because the meta-alternatives are which cues to consider in the next iteration (for the current decision) not which alternatives to consider. Therefore, just as with retrieving relevant alternatives, we assume that this meta-decision is part of the relevant cue retrieval process. We similarly assume that the set of relevant cues is given in each iteration and do not include a mechanism for their retrieval or generation in this model. In Section 6 we demonstrate heuristics both with and without this secondary meta-decision on cues.

Algorithm 2: Assemblable iterative decision heuristic

```
Input: A gap, $G$, specifying the current decision problem
Output: A decision, composed of a choice $C' \subseteq C$ and a boolean indicating if it was
        accepted or rejected
$W \leftarrow \{\}$                                   /* Initialize working set */
$C \leftarrow \{\}$                                   /* Initialize choice set */
$C' \leftarrow \{\}$                                  /* Initialize choice */
$accepted \leftarrow \bot$
while not $accepted$ do
    $C \leftarrow$ UPDATEALTERNATIVES$(G, C, C')$
    $W \leftarrow$ UPDATECUES$(G, W)$
    if $C = \{\}$ or $W = \{\}$ then
        return $(\{\}, \bot)$   /* The agent failed to make a decision. */
    end
    $\mathcal{E}_C \leftarrow$ AGGREGATE(ASSESS$(C, W)$)          /* Judging */
    $C' \leftarrow$ TAKE$(C,$ ORDER$(\mathcal{E}_C))$              /* Choosing */
    $accepted \leftarrow$ ACCEPT$(C', \mathcal{E}_C)$
end
return $(C', \top)$
```

Including UPDATEALTENATIVES and UPDATECUES in the cognitive structure, we arrive at an algorithm for an assemblable iterative decision heuristic, shown in Algorithm 2. This algorithm takes as input the gap defining the current decision problem and produces as output a decision. As illustrated in Figure 1, an iteration starts by updating the cues and alternatives, proceeds through a pipeline of judging cues and collecting a choice, then either terminates if the choice is accepted or starts a new iteration. In the next section we present a method for assembling a cognitive structure which provides this cognitive function.

We describe a small number of different implementations of each heuristic component in Appendix B based on heuristics from the decision-making literature. It may be desirable for an agent to have some minimal set of components to choose from, as deciding which components to use together for which decision requires effort. Rieskamp and Otto (2006) identify that having too many existing heuristics makes it more difficult to learn when to use them, but too few leads to worse

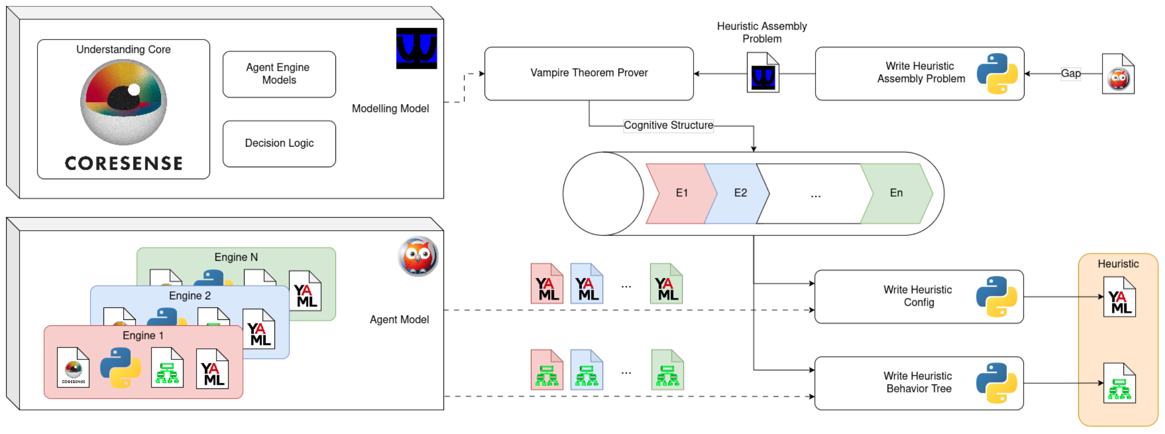

Figure 2. An implementation of a heuristic assembly process, based on theorem proving.

agent performance. It is also possible that different combinations of heuristic components produce the same outcomes. We take the pragmatic view that as long as an agent has enough tools in its toolbox to accomplish its mission and take advantage of the structure of its environment, then eliminating the possibility of duplicate solutions is not worth the trouble. This assumption has yet to be validated and is a great candidate for further investigation.

5. Assembling and Executing Heuristics

Once a gap has been identified by a robot, it can begin assembling or searching for a heuristic to solve the decision problem. If a sufficiently similar problem has been solved previously, it can just reuse the heuristic and avoid the cost of assembly. We implemented a heuristic assembly method, shown in Figure 2, based on using theorem proving techniques to generate behavior trees. Briefly, behavior trees are modular directed acyclic graphs composed of control flow and execution nodes (Colledanchise and Ögren, 2018). A "tick" signal is propagated through the tree, with control flow nodes directing which of their children to forward the signal to and leaf nodes executing some function when they receive this signal. For an example of an assembled behavior tree, see Figure 5. We demonstrate this heuristic assembly and execution method in Section 6.[2]

Each heuristic component is implemented as a **CoreSense** module associated with a particular execution node in the behavior tree and robot knowledge is stored in a Prolog[3] database. **CoreSense** modules (shown in Figure 3) encapsulate a paired engine and model which together provide a cognitive function (Gabriel et al., 2023a). The *engine* is an executable program that performs computations or actions, implementing a specific cognitive function. At runtime an agent can attempt to modify its understanding or environment by exerting this engine on an actionable *model* of the world. **CoreSense** modules also contain internal structural elements:

2. Source code for this demonstration is available upon email request to the author.
3. We are using SWI-Prolog version 8.4.2 for x86_64-linux.

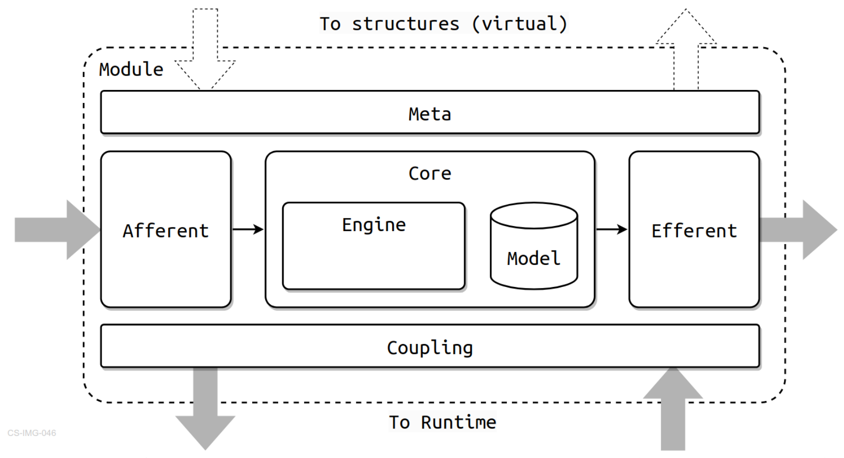

Figure 3. Schematic of a **CoreSense** cognitive module. CS-IMG-046 adopted from Gabriel et al. (2023a).

- The *afferent* and *efferent* provide a means for handing input and output to other modules respectively.
- The *coupling* interfaces with the **CoreSense** *runtime system* which manages module deployment and control.
- The *meta* handles the module itself such as monitoring the state of the module or controlling it as part of an aggregate.

The meta describes the engine including the cognitive capability it provides, the shape of its inputs and outputs, and the cognitive function it employs (Gabriel et al., 2025). Each module is implemented as a ROS[4] node and uses the ROS communication layer to interface with other modules.

We use the **CoreSense** *Understanding Core* (Gabriel et al., 2025) to assemble heuristics (top of Figure 2). Each heuristic component is modeled as an engine with a specific configuration and semantic interface with chains of engine exertions composing a cognitive structure. The inputs and outputs of each heuristic component are modeled explicitly with rich semantic types. The Understanding Core is implemented declaratively in the Typed First-order Form language (TFF), a subset of the TPTP languages used for automated theorem proving (Sutcliffe, 2024). At the time of writing, the Understanding Core models engine exertions with a situation calculus where constructing a cognitive structure is achieved by generating a proof of the existence of the final desired output. We describe each heuristic component at an agent's disposal with the TFF interface used by the Understanding Core, then use it to derive a cognitive structure that produces a decision with the properties encoded in the gap. We use the Vampire Automated Theorem Prover[5] (Bártek et al., 2025; Kovács and Voronkov, 2013) as the backend to generate this proof.

In our demonstrations, we model an agent control architecture using behavior trees that call ROS actions (see Figure 4). For each engine exerted in the derived cognitive structure, the matching

4. We are using the ROS 2 Humble release.
5. We are using Vampire 4.9 Release build linked to Z3 4.14.0.

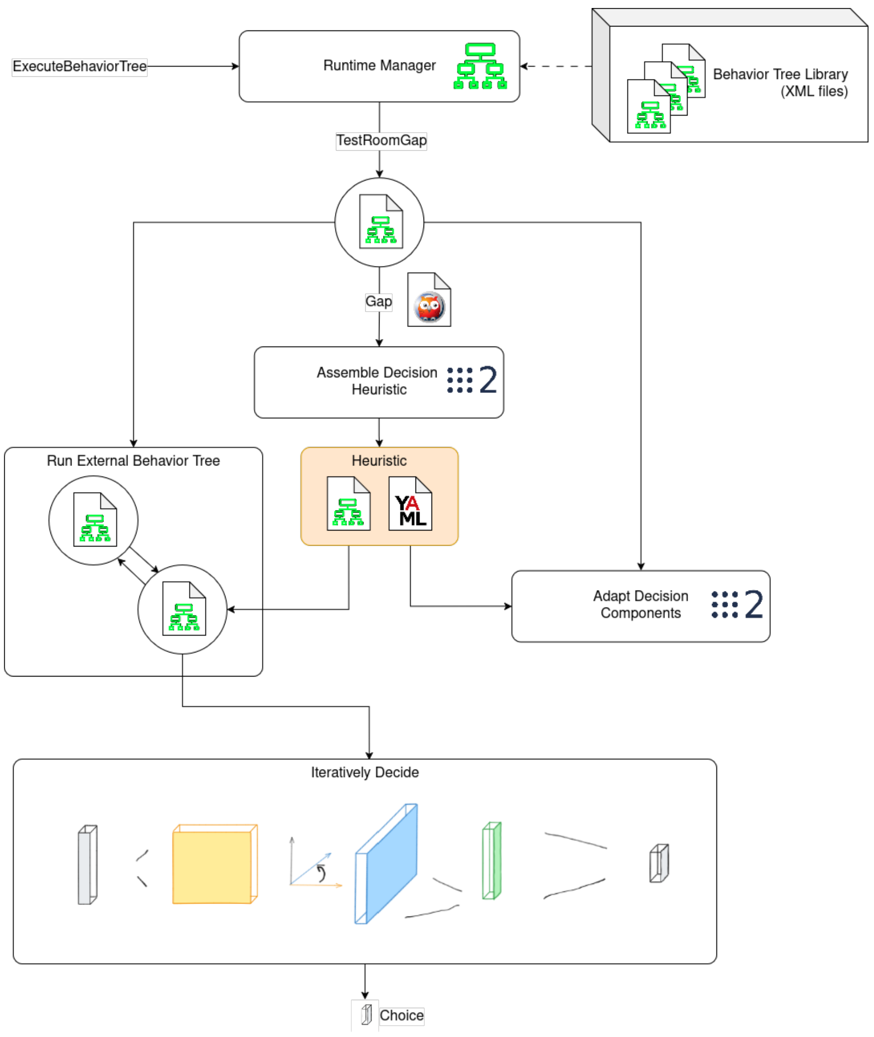

Figure 4. A schematic diagram of the agent control structure used in our demonstrations for assembling and executing decisions.

meta and configuration are fetched from the agent's knowledge base and added to a pair of XML and YAML files, respectively, representing the decision heuristic (bottom of Figure 2). These XML files can be run by a BehaviorTree.CPP (Faconti, 2019) runtime manager and YAML files are used to adapt the parameters for each ROS node in the behavior tree. The meta of each heuristic component is a partially implemented behavior tree that ensures all of the preconditions for running the component are met before it is executed and any post conditions are set appropriately depending on success or failure. Each meta, highlighted with different colors in Figure 5, is a sub-tree starting with a named fallback node. When assembling the heuristic, each meta is linked to the running ROS action that provides the cognitive function it models. This design follows the backward chaining strategy recommended by Colledanchise and Ögren (2018) for creating behavior trees that provide a high degree of proactive explainability and minimize essential complexity (Biggar et al., 2022).[6]

When making a novel decision, an agent must first assemble the heuristic, then can "insert" it into the running behavior tree. We achieve this insertion by implementing a leaf node which spawns a second runtime manager in a new thread that executes the decision. When ticked by the parent tree, this leaf node returns `RUNNING` until the child thread has finished and then returns either `SUCCESS` or `FAILURE` based on the returned status of the decision heuristic. To execute the decision, an agent must simply load the YAML config file to adapt the appropriate ROS nodes, execute the XML behavior tree, then revert to its previous configuration if necessary. Successful execution of a heuristic writes the solution to the Prolog knowledge base where it can be retrieved by the agent at any time.

6. Proof of Concept

To demonstrate our model of decision making and heuristic assembly method in action, we present a proof-of-concept scenario based on Zamani et al. (2025) where a robot has the complete toolbox of heuristics detailed in Appendix B at its disposal.[7] Imagine a domestic service robot is asked to quickly retrieve a book from somewhere in an apartment by one of the occupants. This person communicates their abstract value of timeliness and goal of having a book to the robot. The apartment is made of four rooms: and office, bedroom, living room, and a kitchen. Assume the robot understands that when it is looking for something, places it has already visited are unlikely to contain that object, and that locations which are further away will take longer to investigate. The robot has been in this apartment for some time so it knows what all the rooms are and has learned the reading habits of some of the occupants.

- The book belongs in the office, so without any other clues it is most likely there.
- The occupant who leaves their dirty tableware around usually reads in bed so if it encounters dirty tableware the book is most likely to be in the bedroom.
- The other occupant who cleans their tableware but forgets it usually reads in the living room, so if clean tableware is found the book it most likely there.
- If both clean and dirty tableware is found, the robot assumes someone was reading but doesn't know who, so the bedroom and living room are equally as likely, but above the others.

To complicate matters, the manufacturer programmed the robot with a maintenance rule mandating that whenever it passes through a doorway, if something was blocking it that object must be put away. We assume this robot can deduce that if doorways are blocked it will take some time to clear them before it can pass through and that any object in the doorway will be expensive to investigate.[8]

6. This method of behavior tree generation is not without its flaws. See Colledanchise and Ögren (2018) for details.
7. The complete problem description can be made available upon email request to Carlos Hernández Corbato
8. Notice that with this self-awareness, the robot will act according to the values of the mission, circumventing a manufacturer rule to solve its task. Which rules are okay to circumvent is an alignment problem.

This particular robot uses a two-step strategy to find the book: *first it chooses a room in the house to visit based on its understanding of where the book might be, then once it is in this room, it decides which objects to investigate to see if they are the book.* If it fails it tries this process again with the updated knowledge of the objects it has investigated. We assume the robot can make some weak predictions about the likelihood that each object in it's current room are the book based on its perceptions of the objects from afar, but is not certain until it visits each of them.

We implemented two decisions the agent will need to make while looking for the book based on its simple strategy. In the first decision the robot must choose a room to target. In the second decision the robot decides which objects to investigate.

6.1 Room decision walkthrough

In the first decision, choosing a room, the gap consists of the Prolog clauses shown in Listing 1.

```
%% A handle for the gap
gap(room_gap).

%% The class of alternatives the agent should consider
alternative_of(A, room_gap) :-
    room(A).

%% Requirements on the choice
requirement_of(absolute_size_req, room_gap).
```

Listing 1. Problem definition for choosing a room, as represented in the robot's Prolog knowledge base.

We will assume the robot has not attempted this specific task before so it will need to assemble a heuristic to solve the task. The understanding core then assembles the following cognitive structure which we will refer to as the *Decide-On-Room* heuristic:

- `Update Alternatives with Elimination` - Keeps only the alternatives that were taken in the previous iteration. The robot is able to take advantage of the fact that all alternatives were known at the start and can be progressively eliminated.
- `Update Cues, Take The Best` - Gathers one working cue per iteration, and never reuses them. *Take-The-Best* is used to decide which cue to select each iteration.
- `Assess` - There is only one ASSESS component to select.
- `Aggregate Preferences` - This just uses the assessments directly as value judgments. Because this heuristic only uses one cue at a time (it is a one-good-reason heuristic), this has limited effect since no aggregation is necessary.
- `Order Dominating` - Orders alternatives by majority rule. Because this heuristic only uses one cue at a time, any alternatives that are not tied for first will be tied for last.
- `Eliminate Worst` - Eliminates all alternatives tied for last in each iteration, unless they are all tied. Due to the ordering heuristic selected, this has the same effect as `Take Best.`
- `Accept Size` - Accepts when the taken set of alternatives has cardinality $N = 1$, enforcing the requirement that precisely one room is chosen.

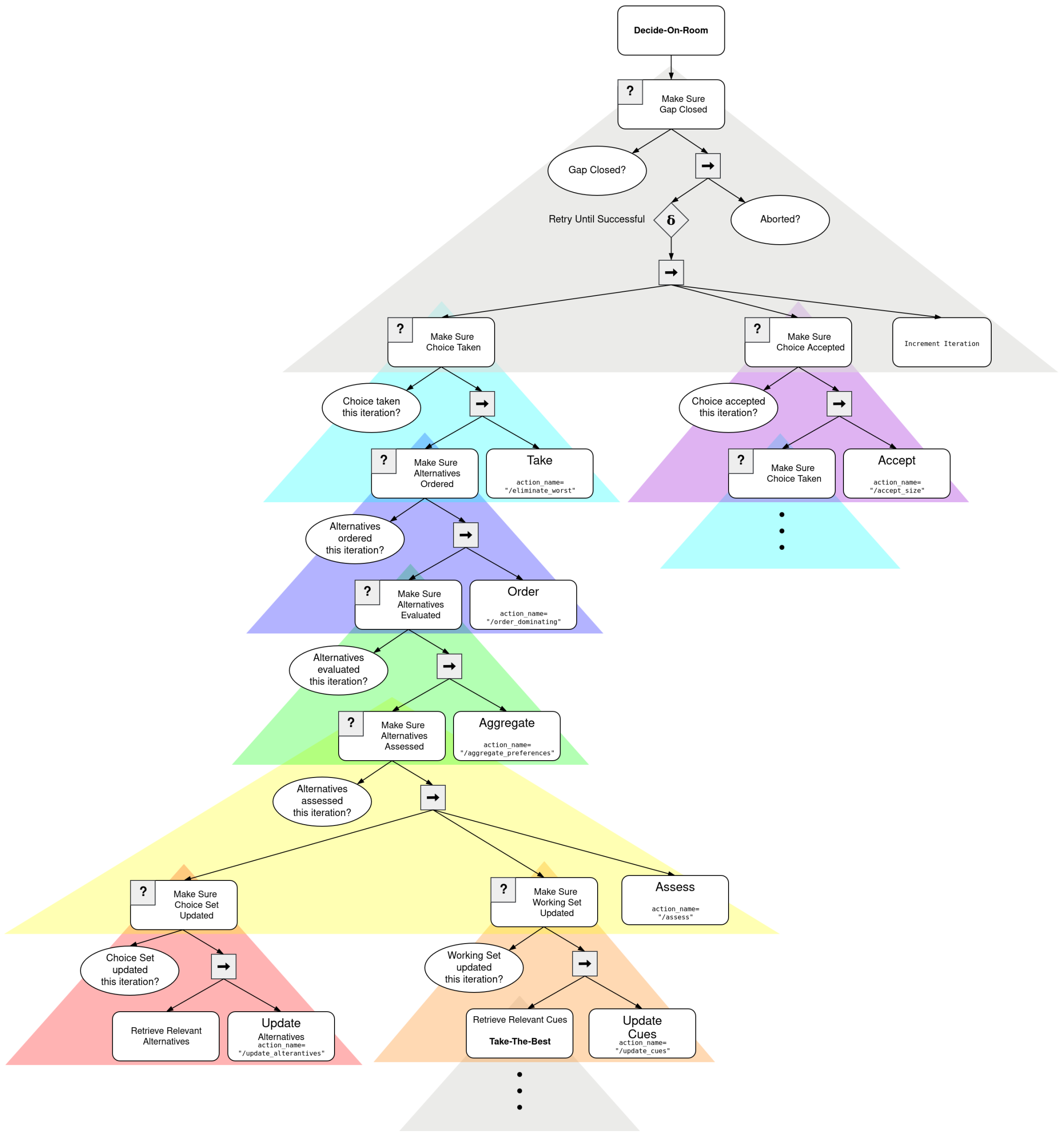
Decide-On-Room
?
Make Sure Gap Closed
Gap Closed?
Retry Until Successful
δ
Aborted?
Make Sure Choice Taken
Make Sure Choice Accepted
Increment Iteration
Choice taken this iteration?
Choice accepted this iteration?
Make Sure Alternatives Ordered
Take
action_name= "/eliminate_worst"
Make Sure Choice Taken
Accept
action_name= "/accept_size"
Alternatives ordered this iteration?
Make Sure Alternatives Evaluated
Order
action_name= "/order_dominating"
Alternatives evaluated this iteration?
Make Sure Alternatives Assessed
Aggregate
action_name= "/aggregate_preferences"
Alternatives assessed this iteration?
Make Sure Choice Set Updated
Make Sure Working Set Updated
Assess
action_name= "/assess"
Choice Set updated this iteration?
Working Set updated this iteration?
Retrieve Relevant Alternatives
Update Alternatives
action_name= "/update_alterantives"
Retrieve Relevant Cues
Take-The-Best
Update Cues
action_name= "/update_cues"

Figure 5. A behavior tree assembled to implement the *Decide-On-Room* heuristic. Sequence nodes are represented with an arrow, fallback nodes with a question mark, condition nodes as ellipses, and execution nodes as rectangles. Sub-trees have been condensed are indicated by three vertical dots.

A schematic diagram of the generated behavior tree is given in Figure 5 where each heuristic component can be seen as a rectangular leaf node. This heuristic falls into the category of *elimination* heuristics, which select which cue to use each round by their validity. It demonstrates how an agent can use a meta-decision to determine which cues are relevant. The specific meta-decision selected is an example of an agent reusing a heuristic already implemented in it's toolbox, however this meta-heuristic could also be assembled at runtime.

Now that the robot has a heuristic assembled, it adapts its heuristic components and spawns a new runtime manager to execute the assembled behavior tree. First the robot updates its choice set, selecting all four rooms in the apartment to start. Next it updates its working set of cues, triggering a meta-decision on which cues are relevant using *Take-The-Best*. This pre-existing meta-heuristic is implemented with these components:

- `Update Alternatives, keep none` - Uses all meta-alternatives discovered so far each iteration. Once a meta-alternative has been considered it will not be considered again.
- `Update Cues, reuse all` - Adds all available meta-cues each iteration. The meta-cue addition policy has no effect because only one meta-cue, *validity*, is used each round.
- `Aggregate Preferences` - This just uses the assessments directly as value judgments. Because this meta-heuristic only uses one meta-cue, this engine has limited effect.
- `Order Lexicographical` - Orders meta-alternatives lexicographically according to their validities. Because this heuristic only uses one meta-cue, this engine has limited effect.
- `Take 1 Best` - Takes the single best meta-alternative, breaking ties randomly. This enforces the one-good-reason pattern that precisely one cue is selected each decision iteration.
- `Accept Satisficing` - Accepts when the chosen cue has a validity of at least 0.5. If this fails, the agent will need to go search for more cues because none that it found so far are positively correlated with the main decision problem (in this case, deciding on a room).

Deviating from Gigerenzer et al. (1999), we implement of the cue retrieval portion of *Take-The-Best* as satisficing, rather than always accepting. Consequently, this meta-heuristic can handle situations where an agent may not know all of the possible cues before starting, explicitly encoding the ability to retrieve more from memory before giving up.

In this scenario suppose that the robot identifies four cues and has knowledge of their validities for selecting rooms based on its past experience. We model this knowledge with the Prolog clauses shown in Listing 2.

```
% ...

available_for('/visited', room_gap).
available_for('/habits', room_gap).
available_for('/doorway_status', room_gap).
available_for('/distance', room_gap).

validity('/visited', 0.8).
validity('/habits', 0.7).
validity('/doorway_status', 0.6).
validity('/distance', 0.6).

% ...
```

Listing 2. Cues and their validities represented in the robot's Prolog knowledge base.

Executing the meta-decision, each cue that has not already been used for this decision is now retrieved from memory as a meta-alternative and assessed by their validities. The assessment is used by the robot as a value judgment and one meta-alternative with the highest validity, `/visited`, is taken. This cue causally links the robot's value of obtaining a book to its belief that the book is unlikely to be in a location it has already visited. It scores the rooms by anticipating its value of being in each alternative room in the future, indicating higher anticipated value with a higher score. Based on these future projections, it assigns a 1.0 to rooms that have not been visited, and a 0.0 to rooms that have. Because the cue has a validity greater than 0.5, it is acceptable and chosen as the first cue to decide which room to target.

Returning to the *Decide-On-Room* heuristic, the agent updates it working set of cues to be {`/visited`} and assesses each of the rooms. As the robot has just started its mission, no room has been visited yet so each gets the same score of 1.0. It simply passes these assessments along as final value judgments, and since none are dominating, all rooms are tied with rank 0. The robot then attempts to eliminate the worst alternatives, however since they are all tied it eliminates none instead and creates a choice containing all four rooms. As this choice has size four instead of one, it is not accepted and a new decision cycle is started.

Once again the agent updates its choice set, keeping only the alternatives taken in the last iteration; in this case it happened to take all of them. Since there are no more rooms it does not add any additional alternatives from memory. Next it updates its working set of cues, not reusing the cues from the previous round, triggering the meta-decision on cues again. Note that this is a new instantiation of the meta-decision so a new gap is created and the agent restarts at meta-iteration 0. This time, `/visited` is not identified as a meta-alternative since it was already used for *Decide-On-Room* in a previous iteration. The meta-heuristic proceeds as before, now choosing `/habits` since it has the highest validity (0.7), and accepting it since that validity is greater than 0.5.

Back in *Decide-On-Room*, the agent assesses each room based on its observations of tableware and beliefs about how that relates to the book's location. Let's suppose it already discovered both clean and dirty tableware, meaning that the book is likely in the bedroom or living room, but it isn't sure which. The `/habits` cue therefore gives the office and kitchen a score of 0.0 and the bedroom and living room a higher, but equal score of 1.0. This assessment is again aggregated and ordered, with the agent eliminating the office and kitchen and taking the bedroom and living room. Since the choice has size two instead of one, the agent again rejects it and starts a new decision cycle.

Finally, in the third cycle the agent randomly chooses between the `/doorway_status` and `/distance` cues since they are tied for validity. Suppose it chooses the `/distance` cue which favors locations by the lowest distance from the robot. The robot is 8.24 meters away from the bedroom and 0.60 meters from the living room, giving them scores of -8.24 and -0.60 respectively. Accordingly, it takes the living room and finally accepts as the choice has a cardinality of one.

6.2 Object decision walkthrough

For the second decision, we assume that the robot already has a heuristic in its toolbox for deciding which objects to target. Perhaps an engineer foresaw this decision needing to be made and programmed the heuristic by hand.

The *Decide-On-Objects* heuristic is implemented as a non-iterative decision and makes no requirements on the size or quality of the choice. It uses the following heuristic components:

- `Update Alternatives, keep none` - Uses all alternatives each round. Because this is a non-iterative heuristic, the keep policy is irrelevant.
- `Update Cues, reuse none` - Adds all available cues each round to the working set. Again, the reuse policy is irrelevant.
- `Aggregate Utility, Dawes' rule` - Assessments are aggregated using Dawes' rule (Dawes, 1979), interpreting cues only as positive or negative.
- `Order Condorcet Extension, Copeland` - Orders alternatives by their net preferences compared to other alternatives. In this case because there is only one dimension of value created by the aggregation function, this has little effect.
- `Take All Best` - Takes all alternatives tied for best. Consequently the size of the choice set may vary depending on the state of the world and could even take no alternatives.
- `Accept Always` - A no-op that always accepts a choice, making the heuristic non-iterative.

Suppose the robot finds six objects in its current room that are available alternatives. Two of these objects are in a doorway, two it has already visited, and for each object it calculates some confidence between $[0, 1]$ that the object is the book. We model this world with the Prolog snippet in Listing 3.

```
% ...

% Stacked to demonstrate interesting results from dawes' rule and copeland
    method
in_doorway(object_1).
in_doorway(object_2).

visited(object_3).
visited(object_4).

confidence_is_book(object_1, 0.8).
confidence_is_book(object_2, 0.2).
confidence_is_book(object_3, 0.0).
confidence_is_book(object_4, 0.6).
confidence_is_book(object_5, 0.6).
confidence_is_book(object_6, 0.6).

% ...
```

Listing 3. Objects world model

The robot then populates the relevant set of cues with all available, finding four. In this case it can reuse two cues that it used to assess rooms as they reflect the same stance the agent has towards the alternatives: `/distance`, and `/visited`.[9] Unfortunately `/distance` always returns a

9. For traveling salesman problems, which the task of investigating several unknown objects comes down to, it is generally not optimal to visit nodes in the order of shortest distance from the current node. However, it can be shown

negative distance, which is unhelpful for differentiating alternatives with the selected aggregation method. However, the robot is still able to decide between alternatives because the other cues are more appropriate for use with Dawes' rule. The third cue, `/in_doorway` scores objects highly that are not in a doorway. The fourth cue, `/is_book`, scores objects based on the agent's confidence that they are the book, with confidences above 0.5 scored by their confidence and below 0.5 scored as $-1 + confidence$. Because of this scheme, confidences greater than 0.5 will be considered positive by the aggregation method and lower than 0.5 will be considered negative.

Assessing all of the alternatives with each of these cues, the agent then aggregates the assessments, orders the alternatives based on its evaluation, and takes objects 5 and 6 which had the biggest difference between positive and negative cues. In a real robot, assessing the fourth cue `/is_book` may represent significant calculation effort, potentially involving sophisticated computer vision models, inference tasks, or Bayesian networks. If the agent instead used an iterative elimination heuristic, ordering cues by their difficulty to assess, it could have used significantly less effort, only calculating the `/is_book` cue for objects 5 and 6. In that case, the robot would have arrived at the same decision but spent less effort, displaying higher ecological rationality.

6.3 Performance measurements

We measured the performance of our method as a baseline for future works to compare against. We profiled 50 runs of the example scenario, capturing results for each of the three heuristics in Table 1. Profiling was performed using Ubuntu 22.04 LTS on a consumer laptop featuring an Intel i9-12900H processor and 16 gigabytes of memory. Naturally, as the *Decide-On-Room* heuristic uses *Take-The-Best* internally to decide which cues are relevant in each iteration, its total runtime is much longer than the other two heuristics which did not have to search for cues. As seen in Table 1, the time spent assembling the *Decide-On-Room* heuristic only represented 8% of the total, demonstrating that it is inexpensive to create new heuristics at runtime for iterative decisions. With runtimes on the order of seconds, a robot could use our method to make decisions with medium to long term effects, but it is unsuitable for reactive decisions such as avoiding obstacles or having conversations.

Table 1. Profiling results for each example heuristic. Note that the total runtime of *Decide-On-Room* had a bimodal distribution where trials taking 3 and 4 iterations had average runtimes of 3358 and 4189 milliseconds respectively.

Decision Heuristic	No. Trials	Avg. Total Runtime (ms)	Avg. Heuristic Assembly Time (ms)	Avg. No. Iterations
Decide-On-Room	50	3690	281	3.4
Take-The-Best	170	480	-	1
Decide-On-Objects	50	579	-	1

that visiting nodes in this way is never worse than twice the optimal path length. This means that admissible heuristics display a flat maximum effect, with diminishing gains from the naive method. This heuristic takes advantage of this information structure in the agent's task.

Table 2. Profiles of example heuristic average runtimes, excluding memory retrieval processes. Units are milliseconds. Average time per iteration for *Decide-On-Room* is shown in parentheses. Breakdown of the second column into each component is shown in Figure 6.

Decision Heuristic	Sum Heuristic Components	BT Setup and Teardown	Component Communication	Self Adaptation
Decide-On-Room	135 (40)	275	1153 (340)	97 (29)
Take-The-Best	36	285	165	21
Decide-On-Objects	66	297	196	21

The total average time taken for each decision was significantly larger than the sum of the decision components themselves. A deeper analysis, presented in Table 2, shows that the majority of decision time is spent setting up the behavior trees and communicating between the heuristic components. This result suggests that using ROS actions to communicate between the heuristic components and the runtime manager is a costly architectural design. The extra time taken for component communication in each *Decide-On-Room* iteration can be attributed to an extra re-adaptation of the decision components after running the *Take-The-Best* heuristic to update the working set of cues. Instead of our method of adapting each running heuristic component for each decision, using multiple running instances of the same components with different parameter settings would eliminate the need for this self-adaptation step. Potential improvements include switching from ROS actions to services for the fast-running heuristic components, using a ROS nodelet architecture to reduce inter-process communication, or using a dynamic behavior tree model that can be modified at runtime instead of launching and tearing down multiple parallel behavior trees.

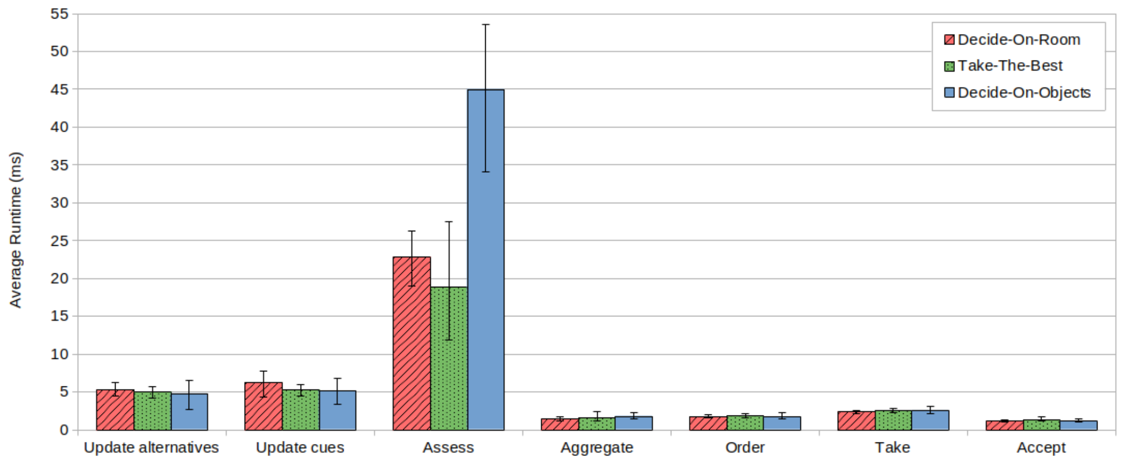

Figure 6. Runtime of each heuristic component used in each of the example heuristics. The average runtime for *Decide-On-Room* is shown in red stripes, *Take-The-Best* in green dots, and *Decide-On-Objects* in solid blue. Error bars indicate the minimum and maximum runtimes measured over 7 trial iterations for each example decision.

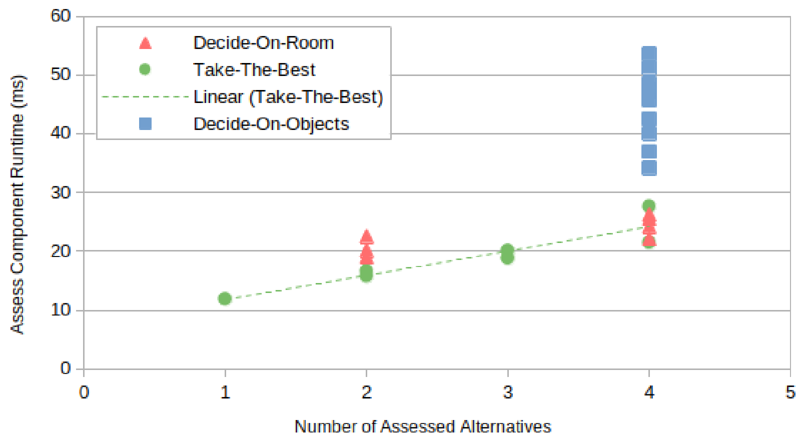

Figure 7. Runtimes of the ASSESS heuristic component vs. the number of assessed alternatives in each of the example heuristics. Iterations of *Decide-On-Room* are shown as red triangles, *Take-The-Best* as green circles, and *Decide-On-Objects* as blue squares. A linear trend-line illustrates a correlation between the number of assessed alternatives and ASSESS component runtime.

We measured the individual runtimes of each heuristic component used in each of the example decisions. As can be seen in Figure 6, for all but the ASSESS component, there is an insignificant difference in runtime between the three decisions. Updating cues and alternatives took around 5 ms each on average, the AGGREGATE, ORDER, TAKE, and ACCEPT components ran for 1 to 3 ms on average, but the ASSESS component was a clear outlier with a wide range of 12 to 54 ms run times.

The wide range of the ASSESS component runtimes can be largely explained by two factors: a difference in the number of alternatives and cues assessed. Illustrated in Figure 7, there is a strong linear correlation between the number of assessed alternatives and the ASSESS component runtime in the *Take-The-Best* decision on cues. A weaker, but positive, correlation can be seen in iterations of *Decide-On-Room* with either 2 or 4 alternatives assessed. Where only one cue was assessed at a time in these two decisions, four cues were assessed at once in *Decide-On-Objects*, resulting in a much higher average ASSESS component runtime of 45 milliseconds. These limited results suggest that robots concerned with decision speed should focus on limiting the number of cues and alternatives to be assessed. With a small number of cues that are quick to assess like the ones we demonstrated, it is faster to consider them all at once instead of iteratively.

7. Discussion

In this section we discuss observations of decision heuristics that we made while developing the model in Section 4 and implementing the heuristics in Section 6. We first note some considerations for selecting heuristic components using a meta-heuristic, then examine the bounded rationality of heuristics based on how they update the choice set of cues and working set of alternatives. We evaluate theorem proving as a method for heuristic assembly and describe some potential alternatives. Finally we discuss some of the strengths and weaknesses of our model.

7.1 The heuristic heuristic

We attempted to derive which heuristic components should be used by an agent from the decision requirements, but there proved to be a minefield of special cases, gotchas, and optimizations left unrealized in our proof-of-concept demonstration. Perhaps obviously, instead an agent could use some meta-heuristic to decide which heuristic or heuristic components to use for a given decision problem. Besides the previously discussed problems with infinite regress, this method can also use a small set of predefined heuristics to grow an agent's library of heuristics. We identified some useful cues for deciding between "heuristic patterns" based on observations of heuristics used in literature, but have not yet implemented them. These patterns could then be filled in with specific components that ensure all the requirements are met for the decision problem at hand. Decision requirements themselves can be helpful in making these coarse filtering decisions.

Some example binary cues that we identified to rule these heuristic patterns in or out include:

- Are all alternatives available?
- Are all cues available?
- Is information scare or expensive to evaluate?
- Does the agent already have a good understanding of the ordering of cues for this decision?
- Does the agent already have a good understanding of the validities of cues for this decision?

Similarly, Svenson (1979) identifies heuristic patterns by the coarseness of cues, if they are commensurable or not, and if they can be used iteratively (such as in one-good-reason methods) or all at once (such as in multiple regression). Interestingly, many of these cues motivate the retrieval of alternatives and cues prior to selection or assembly of a heuristic. To take advantage of these heuristic-cues, we propose priming heuristic assembly by first retrieving a relevant set of cues and available alternatives, as these steps are always required in the first heuristic iteration anyway.

7.2 Bounded rationality, and updating cues and alternatives

The nature of how heuristics collect alternatives and cues can be grouped into four major categories by how they update their respective working sets of objects. We call an update function *accumulating* if it continuously adds more objects to its working set, *diminishing* if it continuously removes objects from its working set, *static* if its working set does not change, and *variable* otherwise. To be useful for decision heuristics, update components must always provide at least one cue or alternative if they are available otherwise the decision will fail immediately.

For an iterative decision heuristic to exhibit bounded rationality it must be able to halt even if a decision cannot be made. If both update components of a heuristic are static, then failure in the first iteration will not change the outcome of the decision and the agent can halt immediately. Otherwise, at least one of the update components must be diminishing so that an agent will eventually run out of either cues or alternatives and halt. With an accumulating update component, in each iteration more alternatives or more cues are considered until all possible alternative or cues have been exhausted. If the sizes of these sets are small, an agent could potentially afford an exhaustive search, however with sufficiently large or unbounded sizes, this is prohibitively impractical. Even if a heuristic uses one variable update function and one either static or variable update function it is not guaranteed to halt. In both cases, although the size of the working sets do not change monotonically, by definition they must each have a minimum size of one.

Therefore, only combinations which include one diminishing update component or two static update components should be considered by agents operating under bounded rationality, absent other information about their environment or mechanism for stopping the decision process. For example, agents using the *satisficing* heuristic (which uses a static set of cues and an accumulating set of alternatives) must have some external mechanism for halting the decision process or artificially limit the number of considered alternatives. In robotics, this often takes the form of timeouts or hard limits on the number of allowed iterations.

Notably, this observation could be used when assembling a decision heuristic to restrict the parameters of each of the update components based on the availability of cues and alternatives at the start of the decision. Table 3 and Table 4 show the behaviors for each update component resulting from combinations of their parameter settings (assuming the working sets are filled to capacity each iteration). Each parameter is described in Appendix B.

Table 3. Behavior of iterative use of UPDATEALTERANTIVES based on parameter settings and availability of alternatives at the start of the decision process.

Keep policy	Capacity	All known at start	Behavior
all	free	-	accumulating
all	fixed	-	variable
all	fixed=start	T	static
none	-	F	variable
none	fixed	T	diminishing
taken	-	T	diminishing
taken	free	F	variable
taken	fixed	F	variable

Table 4. Behavior of iterative use of UPDATECUES based on parameter settings and availability of cues at the start of the decision process.

Reuse	Iterative add	All known at start	Behavior
-	free	T	static
T	-	F	accumulating
T	fixed	T	variable
F	-	F	variable
F	fixed	T	diminishing

7.3 Theorem proving as a method for heuristic assembly

We implemented a heuristic assembly mechanism using an automated theorem prover to derive valid cognitive structures for decision making. This mechanism appeared to be promising at the outset of the implementation process, however it proved quite difficult when more components were added and the decision structures got more complex. As a consequence of using first order logic, in general the proofs we attempted to generate were only semi-decidable, meaning that if a solution does exist, it will be solvable in finite time, but if it does not exist there there is no guarantee of finding one. Additionally, even if a solution could be found in finite time, state-of-the art theorem provers may take a long time as the speed at which they can generate solutions depends highly on the structure of the problem (Suda, 2022). There is no known generally applicable strategy for solving any satisfiability problems with tight performance bounds in the size of the problem (Suda, 2022). Because the problems generated by our assembly logic were on the order of 800-1000 clauses, it was difficult to find strategies which converged in a reasonable amount of time (we limited all proof attempts to 60 seconds during tuning).

This fragile unboundedness goes counter to the ethos of bounded rationality, so we believe it is ill-suited as a method for robots to adaptively assemble decision heuristics. After an initial attempt to model the heuristic components using higher order logic proved too difficult for capable theorem provers to solve (we attempted this with the Leo-III automated theorem prover (Steen, 2025), and a build of Vampire for higher order logic), we decided to switch to first order logic, sacrificing some expressive power for more stable solver performance. If using theorem proving methods in future works, we recommend using either fragments of first order logic that are easier to prove, such as horn clauses like Prolog, or description logics like PDDL. Non-Axiomatic Reasoning Systems (Xu and Wang, 2012) which perform reasoning using bounded rationality are also very attractive candidates.

One alternative is modeling the heuristic assembly process as a learning task where the agent learns a reward function for each set of heuristic components based on the decision problems it is solving. Rieskamp and Otto (2006) demonstrate the SSL framework which models how humans select one of two decision strategies for solving binary inference tasks. They assume that these strategies already exist, but note that their model could be extended to strategy components. Dodampegama and Sridharan (2023b) demonstrate a method of constructing fast-and-frugal trees (Martignon et al., 2008) to model heuristics used by team-mates and opponents in a fort defense scenario. In Dodampegama and Sridharan (2023a), they build on this work by enabling online selection, learning, and adaptation of these models. We do not foresee any inherent difficulties in learning when certain heuristic components of our model are effective in a particular environment, however these techniques would likely require extra information to be stored between the execution of various heuristic elements. These modifications look very similar to the knowledge base modifications we make throughout the decision process and can easily be added to the meta surrounding each engine exertion.

Another potential alternative is using program synthesis techniques to assemble decision heuristics. Program synthesis attempts to generalize a small number of example programs to larger sets of problems by identifying common patterns between many problem solutions. Given a few successful heuristics, such as those found in literature or imagined by engineers, and knowledge of the building blocks of decision heuristics, these methods could potentially help capture the difficult-to-identify structural patterns in robot environments, learning generally applicable rules for assembling heuristics.

7.4 Model strengths and weaknesses

Our model of decision making displays high generality, as it is widely applicable to many decision problems and domains. However this comes at the expense of high flexibility because there are (infinitely) many parameters and implementations of the various heuristic components. This makes our definition very useful as a prescriptive model to make decisions, but much less so as a descriptive model that can explain why one black-box decision was made over another.

We identified many kinds of decision problems that humans and robots could face, but there could be others that our model is too cumbersome or not flexible enough to contend with. We encourage further exploration of similar cognitive capabilities to decision making that can reuse the components we have defined in new ways. Our algorithm could be useful for cognitive scientists to develop new models of human decision-making or to describe the mechanisms of existing ones. However it is not suitable (nor was intended) to describe the precise realization of that process in the human brain.

We proved that adaptive heuristic assembly is possible, but only demonstrated a few decisions in one environment. Extension of this proof of concept to a complete robot mission is a natural next step in testing its strengths and weaknesses. This should also include a comparison to alternative methods of generalized decision making. Kotseruba and Tsotsos (2020) suggests using competitions and ranking benchmarks, however adopting a perspective of ecological rationality requires different evaluation methods than robotics and AI have grown accustomed to. Generally applicable heuristics will by nature perform worse on any particular problem than fine-tuned ones, so they need to be measured across a range of problems in order to make a fair comparison (Kirsch, 2017a). Unfortunately this requires developing problems which are difficult enough that they can demonstrate the differences between heuristics (see the flat maximum effect in Section 2.3) while still being measurable enough to facilitate scientific analysis.

We assume that a robot using our method to make decisions is provided with enough knowledge to link the specific decision it needs to make with its overall mission. However, this is highly demanding of the mission designer and the memory system of the robot and requires modeling robot knowledge in unconventional ways. A natural extension of our model is a mechanism for retrieving alternatives and cues from domain knowledge or deriving specific cues from abstract values. One possible step researchers could take in retrieving relevant cues is encoding a link between the agent's values and how each cue measures those values for different classes of alternatives. Laird et al. (2012) approach this task in the Soar architecture by proposing an expected value function based on background knowledge, then refining it with reinforcement learning. Haan and Heer (2012) describe a tool called causal diagrams to connect observable properties to mission goals that we think merits further investigation.

We implemented **CoreSense** modules that each realize one specific heuristic component, but there is evidence that humans use "partial heuristics" before abandoning them and jumping to another (Huber, 2000). With our assembly method, as long as these partial heuristics have the correct inputs and outputs, they could reduce effort in constructing and executing similar heuristics. Similar to the Kirsch model, each component or partial heuristic could be augmented with checkpoints to reconfigure the heuristic or attempt different one.

We also assumed that only one of each heuristic component was used at a time, but for some decision problems, a robot may want to use multiple different components to make complex judgments. For example, it may have found a set of boolean cues to aggregate together for one dimension of value and a disjoint set of scoring cues to aggregate into another dimension. Kainen (2000) describes the *first-fit-decreasing* heuristic that uses two reasonable strategies in parallel but takes the result of whichever finishes first. With small modifications to our algorithm, assessments, judgments, or even complete heuristics could be processed in parallel streams that are recombined in some way to complete the decision.

8. Conclusion

In this work we developed a definition of decision-making for constructivist robots to formulate their own decision heuristics based on their mission values. By modeling each cognitive function explicitly, it was straight forward to implement components common to well-studied heuristics. We developed a toolbox of heuristic components as modules in the **CoreSense** architecture using ROS actions to communicate between modules and behavior trees to orchestrate their execution.

We simulated three heuristics used by a robot to search for and retrieve a book in an apartment, showing how a constructivist robot can make decisions using both pre-existing heuristics and assemble novel ones from a toolbox of components. This simulated robot was able to reason about which of its cognitive functions to use based on its self-understanding, then was able to adapt itself to make mission-critical decisions. It saved cognitive effort by ignoring one of the possible cues in *Decide-On-Room*, but was also able to use all of the available information at once to save time in *Decide-On-Objects*. Through analysis of the constructed heuristics, we were able to precisely explain the decision made by the robot and directly link it to the mission designer's values.

We profiled each of the example heuristics and discussed their ecological rationality in different environments. From this profiling we also identified ways to improve the performance of our implementation, how these improvements effect the speed of different kinds of heuristics, and how an agent could leverage the performance statistics to assemble better heuristics. We described some potential cues for choosing heuristic components and how updating cues and alternatives can affect heuristic boundedness. Finally we discussed some of the limitations of assembling heuristics with automated theorem provers, identified some strengths and weaknesses of our model, and made some recommendations for future works to address them.

Modern robots make decisions in many ways, but rely on their designers to choose which strategies to employ and when. By taking the perspective of bounded rationality, we present robots with a new kit of tools for completing their missions. By understanding their values, taking cognitive short cuts, and tactfully using information, they can make novel and explainable decisions in an open and uncertain world.

References

Albus, J. S. (1991). Outline for a theory of intelligence. *IEEE transactions on systems, man, and cybernetics*, 21(3):473–509.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., and Qin, Y. (2004). An Integrated Theory of the Mind. *Psychological Review*, 111(4):1036–1060.

Baguley, T. and Robertson, S. I. (2000). Where does fast and frugal cognition stop? The boundary between complex cognition and simple heuristics. *Behavioral and Brain Sciences*, 23(5):742–743.

Biggar, O., Zamani, M., and Shames, I. (2022). On Modularity in Reactive Control Architectures, with an Application to Formal Verification. *ACM Transactions on Cyber-Physical Systems*, 6(2):1–36.

Brandstätter, E., Gigerenzer, G., and Hertwig, R. (2006). The priority heuristic: Making choices without trade-offs. *Psychological Review*, 113(2):409–432.

Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation*, 2(1):14–23.

Bullock, S. and Todd, P. M. (1999). Made to Measure: Ecological Rationality in Structured Environments. *Minds and Machines*, 9(4):497–541.

Bártek, F., Bhayat, A., Coutelier, R., Hajdu, M., Hetzenberger, M., Hozzová, P., Kovács, L., Rath, J., Rawson, M., Reger, G., Suda, M., Schoisswohl, J., and Voronkov, A. (2025). The VAMPIRE Diary. In Piskac, R. and Rakamarić, Z., editors, *Computer Aided Verification*, volume 15933, pages 57–71, Cham. Springer Nature Switzerland. Series Title: Lecture Notes in Computer Science.

Choi, D. and Langley, P. (2018). Evolution of the Icarus Cognitive Architecture. *Cognitive Systems Research*, 48:25–38.

Colledanchise, M. and Ögren, P. (2018). *Behavior Trees in Robotics and AI: An Introduction*. CRC Press.

Czerlinski, J., Gigerenzer, G., and Goldstein, D. G. (1999). How good are simple heuristics? In *Simple heuristics that make us smart*, Evolution and cognition, pages 97–118. Oxford University Press, New York, NY, US.

Dawes, R. M. (1979). The robust beauty of improper linear models in decision making. *American Psychologist*, 34(7):571–582.

Dodampegama, H. and Sridharan, M. (2023a). Back to the Future: Toward a Hybrid Architecture for Ad Hoc Teamwork. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(1):3–10. Number: 1.

Dodampegama, H. and Sridharan, M. (2023b). Knowledge-based Reasoning and Learning under Partial Observability in Ad Hoc Teamwork. *Theory and Practice of Logic Programming*, 23(4):696–714.

Faconti, D. (2019). MOOD2Be: Models and Tools to design Robotic Behaviors. Technical report, Eurecat Centre Tecnologic, Barcelona, Spain.

Feeney, A. (2000). Simple heuristics: From one infinite regress to another? *Behavioral and Brain Sciences*, 23(5):749–750.

Fink, E. and Blythe, J. (2005). Prodigy bidirectional planning. *Journal of Experimental & Theoretical Artificial Intelligence*, 17(3):161–200.

Gabriel, A., Hernandez, C., Sanz, R., Aguado, E., Fernandez-Cortizas, M., GP-Lenza, G., Gonzalez, I., Rodriguez, F., and Martin, F. (2025). Specification of the CoreSense Architecture. Deliverable D2.2, The CORESENSE Project.

Gabriel, A., Zamani, F., Hernández, C., García, C., Aguado, E., and R.Sanz (2023a). Conceptual analysis and architecture requirements specification. Deliverable D21, The CORESENSE Project.

Gabriel, A., Zamani, F., Hernández, C., Narváez, J., and Sanz, R. (2023b). Cognitive capability domain analysis. Deliverable D3.1, The CORESENSE Project.

Gigerenzer, G. (2001). The Adaptive Toolbox. In Selten, R. and Gigerenzer, G., editors, *Bounded Rationality: The Adaptive Toolbox*. The MIT Press.

Gigerenzer, G. and Brighton, H. (2009). Homo Heuristicus: Why Biased Minds Make Better Inferences. *Topics in Cognitive Science*, 1(1):107–143.

Gigerenzer, G. and Goldstein, D. G. (1996). Reasoning the fast and frugal way: Models of bounded rationality. *Psychological Review*, 103(4):650–669.

Gigerenzer, G., Todd, P. M., Group, A. R., Gigerenzer, G., Todd, P. M., and Group, A. R. (1999). *Simple Heuristics That Make Us Smart*. Evolution and Cognition. Oxford University Press, Oxford, New York.

Haan, A. d. and Heer, P. d. (2012). *Solving complex problems: professional group decision-making support in highly complex situations*. Eleven International Publishing, The Hague, The Netherlands.

Hertwig, R., Hoffrage, U., and Martignon, L. (1999). Quick estimation: Letting the environment do the work. In *Simple heuristics that make us smart*, Evolution and cognition, pages 209–234. Oxford University Press, New York, NY, US.

Hu, Y., Lei, Z., Zhang, Z., Pan, B., Ling, C., and Zhao, L. (2025). GRAG: Graph Retrieval-Augmented Generation. In Chiruzzo, L., Ritter, A., and Wang, L., editors, *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 4145–4157, Albuquerque, New Mexico. Association for Computational Linguistics.

Huber, O. (2000). What's in the adaptive toolbox: Global heuristics or more elementary components? *Behavioral and Brain Sciences*, 23(5):755–755.

International Electrotechnical Commission (2018). *Failure modes and effects analysis (FMEA and FMECA)*. IEC 60812:2018, Genève, Suisse. https://webstore.iec.ch/en/publication/26359.

Juslin, P., Jones, S., Olsson, H., and Winman, A. (2003). Cue abstraction and exemplar memory in categorization. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 29(5):924–941.

Kainen, P. C. (2000). The role of mathematics in heuristic performance. *Behavioral and Brain Sciences*, 23(5):755–756.

Katsikopoulos, K. (2019). Kirsch's, and everyone's, bind: How to build models for the wild? *Cognitive Processing*, 20(2):269–272.

Katsikopoulos, K. V., Durbach, I. N., and Stewart, T. J. (2018). When should we use simple decision models? A synthesis of various research strands. *Omega*, 81:17–25.

Kirsch, A. (2017a). Lessons from human problem solving for cognitive systems research. *Advances in Cognitive Systems*, 5:13–24.

Kirsch, A. (2017b). A Modular Approach of Decision-Making in the Context of Robot Navigation in Domestic Environments. In *GCAI*, pages 134–147.

Kirsch, A. (2019). A unifying computational model of decision making. *Cognitive Processing*, 20(2):243–259.

Klein, G. A. (2017). *Sources of power: How people make decisions*. MIT press.

Kotseruba, I. and Tsotsos, J. K. (2020). 40 years of cognitive architectures: core cognitive abilities and practical applications. *Artificial Intelligence Review*, 53(1):17–94.

Kovács, L. and Voronkov, A. (2013). First-Order Theorem Proving and Vampire. In Sharygina, N. and Veith, H., editors, *Computer Aided Verification*, pages 1–35, Berlin, Heidelberg. Springer.

Košeckà, J. and Bajcsy, R. (1994). Discrete Event Systems for autonomous mobile agents. *Robotics and Autonomous Systems*, 12(3-4):187–198.

Laird, J. (2012). *The Soar cognitive architecture*. MIT Press, Cambridge, Mass London, England.

Laird, J. E., Derbinsky, N., and Tinkerhess, M. (2012). Online determination of value-function structure and action-value estimates for reinforcement learning in a cognitive architecture. *Advances in Cognitive Systems*, 2:221–238.

Macenski, S., Foote, T., Gerkey, B., Lalancette, C., and Woodall, W. (2022). Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074.

Maes, P. (1989). How to do the Right Thing. *Connection Science*, 1(3):291–323.

Martignon, L., Katsikopoulos, K. V., and Woike, J. K. (2008). Categorization with limited resources: A family of simple heuristics. *Journal of Mathematical Psychology*, 52(6):352–361.

Minsky, M. (1988). *Society Of Mind*. Simon and Schuster. Google-Books-ID: bLDLllfRpdkC.

Payne, J. W., Bettman, J. R., and Johnson, E. J. (1993). *The Adaptive Decision Maker*. Cambridge University Press, 1 edition.

Pirjanian, P. (1999). Behavior coordination mechanisms-state-of-the-art. Technical Report IRIS-99-375, University of Southern California.

Riekki, J. and Roning, J. (1997). Reactive task execution by combining action maps. In *Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robot and Systems. Innovative Robotics for Real-World Applications. IROS '97*, volume 1, pages 224–230, Grenoble, France. IEEE.

Rieskamp, J. and Otto, P. E. (2006). SSL: a theory of how people learn to select strategies. *Journal of experimental psychology: General*, 135(2):207.

Rosenblatt, J. K. (1997). DAMN: a distributed architecture for mobile navigation. *Journal of Experimental & Theoretical Artificial Intelligence*, 9(2-3):339–360.

Russell, S. J. (1991). An architecture for bounded rationality. *ACM SIGART Bulletin*, 2(4):146–150.

Saffiotti, A., Konolige, K., and Ruspini, E. H. (1995). A multivalued logic approach to integrating planning and control. *Artificial Intelligence*, 76(1-2):481–526.

Scheutz, M. and Andronache, V. (2004). Architectural Mechanisms for Dynamic Changes of Behavior Selection Strategies in Behavior-Based Systems. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, 34(6):2377–2395.

Shah, A. K. and Oppenheimer, D. M. (2008). Heuristics made easy: An effort-reduction framework. *Psychological Bulletin*, 134(2):207–222.

Shanks, D. R. and Lagnado, D. (2000). Sub-optimal reasons for rejecting optimality. *Behavioral and Brain Sciences*, 23(5):761–762.

Simon, H. A. (1956). Rational choice and the structure of the environment. *Psychological review*, 63(2):129.

Sridharan, M. (2025). Back to the Future of Integrated Robot Systems. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(27):28610–28615.

Steen, A. (2025). Leo-III 1.7.18.

Suda, M. (2022). Vampire Getting Noisy: Will Random Bits Help Conquer Chaos? (System Description). In Blanchette, J., Kovács, L., and Pattinson, D., editors, *Automated Reasoning - 11th International Joint Conference, IJCAR 2022, Haifa, Israel, August 8-10, 2022, Proceedings*, volume 13385 of *Lecture Notes in Computer Science*, pages 659–667. Springer.

Sun, R., Wilson, N., and Lynch, M. (2016). Emotion: A Unified Mechanistic Interpretation from a Cognitive Architecture. *Cognitive Computation*, 8(1):1–14.

Sutcliffe, G. (2024). Stepping Stones in the TPTP World. In Benzmüller, C., Heule, M. J., and Schmidt, R. A., editors, *Automated Reasoning*, pages 30–50, Cham. Springer Nature Switzerland.

Svenson, O. (1979). Process descriptions of decision making. *Organizational behavior and human performance*, 23(1):86–112.

Thórisson, K. R., Kremelberg, D., Steunebrink, B. R., and Nivel, E. (2016). About Understanding. In Steunebrink, B., Wang, P., and Goertzel, B., editors, *Artificial General Intelligence*, pages 106–117, Cham. Springer International Publishing.

Todd, P. M. and Gigerenzer, G. (2000a). How can we open up the adaptive toolbox? *Behavioral and Brain Sciences*, 23(5):89–100.

Todd, P. M. and Gigerenzer, G. (2000b). Précis of Simple heuristics that make us smart. *Behavioral and Brain Sciences*, 23(5):727–741.

Tversky, A. (1972). Elimination by aspects: A theory of choice. *Psychological Review*, 79(4):281–299.

Tversky, A. and Kahneman, D. (1973). Availability: A heuristic for judging frequency and probability. *Cognitive Psychology*, 5(2):207–232.

U.S. Const. amend. XII (1804).

Wimsatt, W. C. (2000). Heuristics refound. *Behavioral and Brain Sciences*, 23(5):766–767.

Xu, Y. and Wang, P. (2012). The Frame Problem, the Relevance Problem, and a Package Solution to Both. *Synthese*, 187(S1):43–72.

Yen, J. and Pfluger, N. (1995). A fuzzy logic based extension to Payton and Rosenblatt's command fusion method for mobile robot navigation. *IEEE Transactions on Systems, Man, and Cybernetics*, 25(6):971–978.

Zachary, W., Le Mentec, J.-C., and Ryder, J. (1996). Interface Agents in Complex Systems. In *Human Interaction with Complex Systems*, volume 372, pages 35–52. Springer US, Boston, MA. Series Title: The Kluwer International Series in Engineering and Computer Science.

Zamani, F., Gabriel, A., Silva, G., Şişman, B., and Hernandez Corbato, C. (2025). Krr course project instructions. PDF.

Appendix A. A zoo of heuristics

To expand the capabilities of deciding agents, their model of decision making should be usable in many kinds of decision problems and environments. Because decision-making is a fundamental component of many cognitive capabilities, a wide variety of decision heuristics have been studied and classified. This appendix presents a brief survey of heuristics that inspired our model.

A.0.1 Variations on multiple regression

The widespread use of the multiple regression heuristic has lead to many variations with minor differences. Shah and Oppenheimer (2008) assume that all relevant alternatives have been collected from memory or identified from the environment where other authors include this step in the decision-making process. Interested readers are directed to Chapter 3 of Haan and Heer (2012) or Gary Klein's book *Sources of power: How people make decisions* (Klein, 2017).

Shah and Oppenheimer (2008) categorize heuristics by which step of the weighted additive rule they modify to reduce effort. One major focus is how the weights of each cue are used. In expected utility theory, cue weights may represent the probabilities of the possible outcomes of selecting each alternative. They could represent preferences of particular alternatives as in Multi Attribute Utility Theory (see Katsikopoulos et al. (2018)), or levels of risk such as in Failure Mode and Effects Analysis (International Electrotechnical Commission, 2018). These weights can be assessed independently for each cue, dependently between cues, or equal cue weights can be used (a heuristic referred to as *Tallying* by Gigerenzer and Brighton (2009)).

A.0.2 Classes of Fast and Frugal Heuristics

Todd and Gigerenzer (2000b) identify three major classes of fast and frugal heuristics based on the amount and kind of cues available to the decision maker: ignorance-based, one-good-reason, and elimination. *Ignorance-based* heuristics are useful if the only information an agent has is whether or not each alternative has been encountered before. The *recognition* heuristic can be used in these situations to choose alternatives that have been recognized before over those that have not. The *fluency* heuristic, also called the "availability" heuristic by Tversky and Kahneman (1973), similarly can be used to select alternatives that come to mind the fastest or that an agent knows the most about.

One-good-reason heuristics apply ordered cues one at a time until a single winning alternative stands out. Examples differing only in their search rule include *take-the-best* which orders cues by their true positive rate (also called *validity*) in predicting the best alternative, *take-the-last* which prefers cues that most recently resulted in a decision even if they were incorrect, and *minimalist* which just uses cues in a random order (Gigerenzer and Goldstein, 1996; Gigerenzer et al., 1999). For these heuristics to be useful, cues and cue orderings do not need to causally or exactly match the environment structure, they just need to point in the right direction (Todd and Gigerenzer, 2000a). Fast and frugal trees are one-good-reason heuristics that strategically order cues so that each cue is guaranteed to rule out some of the alternatives (Martignon et al., 2008). Dodampegama and Sridharan (2023b) and Dodampegama and Sridharan (2023a) employ fast and frugal trees in robotics to model other agents in two ad hoc teamwork scenarios.

When cues cannot individually distinguish between alternatives, *elimination* heuristics can be used to reject alternatives in stages. Contrary to one-good-reason heuristics, *elimination-by-aspects* (Tversky, 1972) uses all cues found so far to discard alternatives instead of applying cues one at a time. In environments where the cost of being wrong far exceeds the cost of being right, Bullock and Todd (1999) showed that elimination heuristics can work well, but the order of cues becomes much more important for success. The *QuickEst* heuristic (Hertwig et al., 1999) can quantify objects along some criterion while using as little information as possible by iteratively selecting the cues that separate the most remaining common objects.

When all of the possible alternatives are not available at the start of decision making, for instance if they are expensive to find or appear only sequentially, satisficing heuristics can be used to stop searching when a good enough alternative is found. The *confirmation rule* (Gigerenzer and Brighton, 2009) suggests waiting until two cues suggest the same alternative, then choosing that one.

A.0.3 Heuristics in different environments

A major research focus of heuristic decision-making literature has been the analysis of the structure of information in the environment which heuristics can take advantage of to exhibit ecologically rational behavior. In environments with scarce information, *take-the-best* can leverage the few known cues relative to the number of alternatives to simply decide based on the first cue that divides one winning alternative from the rest. When the set of alternatives to choose from is constantly shrinking, such as finding a mate or choosing an apartment, satisficing heuristics can lead to better results than exhaustive search because all the best options may already be gone by the time they are evaluated. The *equality* heuristic suggests dividing resources evenly between all alternatives rather than committing to one (Shah and Oppenheimer, 2008). This can be useful for making investments in uncertain markets or distributing food among hungry individuals. Social heuristics can help agents discover successful strategies quickly from others such as *imitating the majority* or *imitating the most successful* (Gigerenzer and Brighton, 2009). In robotics, imitation learning from human demonstrations exploits this technique to mimic human behaviors.

Non-compensatory environments are those where a high score from one cue cannot compensate for a low score in another and the potential contribution of each new cue to a final decision falls off rapidly (Gigerenzer and Brighton, 2009; Gigerenzer et al., 1999). In non-compensatory environments there is no need to integrate information across cues or to form an overall impression of an alternative before it is eliminated, so heuristics like *take-the-best* and *elimination-by-aspects* avoid spending extra effort evaluating poor alternatives (Shah and Oppenheimer, 2008). Poisson processes and power laws commonly encountered in nature produce J-distributions where there are many more objects at one end of a criteria range than the other. The *QuickEst* heuristic can use this to quickly estimate population sizes and *categorization by elimination* heuristic can rule out categories quickly (Gigerenzer et al., 1999).

Appendix B. Toolbox components

We describe a number of specific implementations of heuristic components detailed in Section 3, based on heuristics from the decision-making literature. Similar to the heuristic components themselves, we give each of these specific implementations functional names rather than operational ones. Source code for this demonstration is available upon email request to the author.

B.1 Assess

Because all cues that an agent has committed to for the given iteration are assumed to be used, there is only one natural implementation of ASSESS. It simply populates an assessment matrix from each cue in the working set on each alternative in the choice set.

B.2 Aggregate

Cognitive structures implementing AGGREGATE combine the assessments of each alternative to produce independent judgments of each alternative along (potentially) multiple axes of the agent's value. Generally speaking, implementations of these components that produce a one-dimensional judgment are called "utility functions" and they assume cues are commensurable.

Table 5. Aggregate Components

Name	**Description**	**Related heuristics**
`Pass Preferences`	Passes assessments along as judgments, assuming that each cue directly measures a value of the agent. As judgment dimensions are incommensurable, the assessed cues are assumed to individually measure incommensurable axes of each alternative's value.	one-good-reason heuristics, arbitration-based action selection mechanisms
`Utility - Boolean`	Interprets assessments as boolean preference vectors and combines them all using a boolean operator. This could be generalized to combine preferences according to any complex predicate logic rule, but for simplicity we assume the same operator (either `AND` or `OR`) is used to combine all scores.	suggested by Kirsch (2019)
`Utility - Signed`	Interprets assessment scores as positive or negative and counts the number of each. In the *Tallying* heuristic only the total number of positive cues is counted. In *Dawes' Rule*, the result is the difference between the number of positive and negative scores.	Tallying (Gigerenzer and Brighton, 2009), Dawes' Rule (Dawes, 1979)
`Utility - Sum`	Takes a sum of the assessment scores for each alternative, potentially with some weight according to the cue that produced it, as used in the *Weighted Additive Rule*. When cues are one-hot, this is equivalent to *Weighted Pros*. The unweighted sum can also be taken when there is no difference in importance between the cues. Depending on the cues used, in some implementations the range of scores can first be normalized to [0, 1] within each assessment.	Weighted Additive Rule (Payne et al., 1993), Weighted Pros (Shah and Oppenheimer, 2008), scoring cues
`Multi-value Utility`	Uses some combination of the other utility-based aggregation methods to produce a multi-dimensional judgment.	Command fusion action selection mechanisms, multiple criteria decision making

B.3 Order

Ordering is how an agent takes a stance on incommensurable value judgments to determine its preferences of alternatives. Kirsch (2019) lumps AGGREGATE and ORDER together into one function, but separating them reduces the total number of different functional implementations an agent needs to accomplish the same tasks (e.g. 4 aggregate + 3 order = 7 total, vs. 4*3 = 12 possible combinations). We remove the interdependency Kirsch (2019) identifies between ACCEPT and their implementation combined subroutine AGGREGATEANDORDER. Because AGGREGATEANDORDER could output ranks or scores, their version of ACCEPT needs to be able to interpret them correctly to properly evaluate stopping criteria. Because we assume AGGREGATE always produces value judgments, which are scores, this interdependency is removed and ACCEPT always can assume that evaluations contain scores.

Table 6. Order components

Name	**Description**	**Related heuristics**
`Lexicographical`	Judgment axes are assumed to have some total ordering, where alternatives are ranked by their scores along the first axis first, with subsequent axes breaking ties.	Priority (Brandstätter et al., 2006), Default (Gigerenzer and Brighton, 2009), Ranking cues
`Dominating`	Alternatives are ranked based on how they dominate others. *Majority Rule* measures in how many ways they dominate others where *Pareto Fronts* measures how many others they dominate.	one-good-reason heuristics
`Condorcet`	Condorcet extensions are social choice functions that produce a condorcet winner (an alternative that defeats every other alternative in a pairwise majority) if one exists. The *Copeland* method ranks each alternative by the net preference: the difference in the number of others that are worse than it and better than it in each feature. *Sequential Majority Comparison* gets a single winning alternative by making pairwise comparisons of the net preferences of all alternatives two at a time, keeping the better one. *Majority of Confirming Dimensions* does the same, but only using confirming dimensions (those with where the alternative wins) instead of net preference. *Confirming Dimensions* and *Net Preference* are analogous to *Tallying* vs. *Dawes' Rule*, potentially representing another way to reuse the same underlying machinery.	Majority of Confirming Dimensions, examples by Kirsch (2019)

B.4 Take

Cognitive structures implementing TAKE simply collect the best alternatives from the choice set in some way each iteration to make the final decision. How an agent takes these alternatives depends on the decision to be solved and if they are modifying the size of the choice set each iteration. Kirsch (2019) defines only one take function, FIRST, which always takes the single best alternative (or randomly selects among those tied for best) and assumes that any modification to the choice set for subsequent iterations is evaluated by an UPDATEALTERNATIVES function.

Table 7. Take Components

Name	**Description**	**Related heuristics**
`Take Best`	Takes either a fixed number of the best alternatives or all of the alternatives tied for best. If the number is not fixed, an agent may elect to only consider the alternatives that were taken in the previous round to reduce effort.	Elimination, one-good-reason heuristics, fixed-size choice problems
`Eliminate Worst`	Eliminates either a fixed number of the worst alternatives or all tied for worst from the choice set. An agent may elect not to eliminate any alternatives if all are tied and it has some minimum required number.	Elimination

Using either `Take Best` or `Eliminate Worst`, with a fixed number of alternatives to take/eliminate could result in alternatives tied for worst that are not taken/eliminated. In these cases it may be beneficial for an agent to randomly break ties.

B.5 Accept

Cognitive structures implementing ACCEPT determine if computation on the decision problem should be stopped or not, enforcing the requirements of the decision to be made. If a choice is acceptable, a decision has been committed to, otherwise the decision process iterates.

Acceptability criteria based on the quality of alternatives are by definition value judgments, so they must be taken from the evaluation of the choice set. However, some requirements such as size of the chosen set, do not need access to the value judgments, so they can ignore this input.

Table 8. Accept Components

Name	**Description**	**Related heuristics**
`Always`	Always accept the taken choice. Heuristics using this function will not iterate. This accept function assumes that any requirements on the choice are either met by the other decision components (e.g. *Take-N-Best* enforces the size of the chosen set) or that there are no such requirements. Assumes all necessary cues and relevant alternatives have been identified before the decision process started because there will not be successive iterations to collect more.	Weighted Additive Rule (Payne et al., 1993; Shah and Oppenheimer, 2008)
`Size`	Accepts a choice based on the number of chosen alternatives. This accept function leverages heuristics that modify the size of the choice set between iterations. This could be a fixed number or some relation (e.g. at least 2 scoops of ice cream).	One-good-reason heuristics, Elimination heuristics
`Satisficing`	Accepts a choice if the scores of all chosen alternatives meet some threshold along some subset of the judgment axes. This assumes that some alternatives may not be good enough to meet the requirements for the decision task. Typically this is used when all cues are known beforehand but not all alternatives, however with our implementation it doesn't matter as long as the satisficing axes have been evaluated. For example in *Maes' behavior nets*, the combined activation of an action may need sufficient evidence before it is chosen to be executed.	Satisficing (Simon, 1956), Priority (Brandstätter et al., 2006), Maes Behavior Networks (Maes, 1989)
`Dominating`	Accepts a choice if the scores of all chosen alternatives are strictly better than the best unchosen alternative along some subset of the judgment axes. Kirsch (2019) surmises that an agent using this heuristic may elect to keep thinking if some eliminated alternatives are just as good (or better) in some ways as the taken ones, or if their perceived value is very close and the agent can afford more time to better differentiate between them.	examples in Kirsch (2019)

B.6 UpdateAlternatives

With the choice of which alternatives are worth keeping relegated to TAKE, and available alternative retrieval procedures left to other cognitive functions, we identify two important parameters for a general UPDATEALTERNATIVES component: the maximum size of the choice set to consider in each iteration and how the unaccepted choice from the previous iteration should be used. We assume that in successive iterations, the choice set is refilled up to its capacity with any available alternatives that the agent has discovered. We are unconcerned with how the agent discovers these alternatives or retrieves them from memory and assume their existence each iteration as given.

Table 9. UpdateAlternatives components

Name	**Description**	**Related heuristics**
`Keep All`	Keep all of the alternatives from the previous choice set. This assumes that after enough consideration, some evaluation will be able to differentiate between alternatives in an acceptable way.	Kirsch (2019) example heuristic, one-good-reason heuristics
`Keep None`	Keep none of the alternatives from the previous choice set. This assumes that once an alternative has been evaluated, no further consideration will make it acceptable.	QuickEst (Hertwig et al., 1999), Satisficing heuristics
`Keep Taken`	Keep only the alternatives taken in the last iteration. This assumes that the acceptability criteria for a choice could be satisfied with more consideration but that once an alternative has not been taken, it should not be considered further.	Elimination heuristics

B.7 UpdateCues

We identify two important parameters for a general UPDATECUES component: the maximum number of relevant cues that should be added to the working set in the next iteration, and if cues should be reused from the previous iteration or not. For example, all one-good-reason heuristics add one relevant cue per round and cues from the previous iteration are not reused. Satisficing heuristics typically assume all cues are known at the start of the decision, placing no limit on the number of relevant cues and reusing them all each iteration. These two classes rely on opposite mechanisms: one-good-reason heuristics modify the working set of cues each iteration while satisficing heuristics modify the choice set of alternatives.