Go4RES: Goal-based Modeling for Reward Function Engineering and Shaping

Kenneth H. Chan
Department of Computer Science and Engineering
Michigan State University
East Lansing, U.S.
chanken1@msu.edu

Betty H.C. Cheng
Department of Computer Science and Engineering
Michigan State University
East Lansing, U.S.
chengb@msu.edu

***Abstract*—Search-based software engineering has demonstrated its potential at solving difficult problems in innovative ways across numerous and diverse applications, including reinforcement learning, evolutionary search, uncertainty exploration, software testing, game theory, etc. Foundational to search-based techniques are reward functions that describe the behavior or objective of the problem to be solved. Reward functions are the key means for developers to provide feedback for the system during the search process. Existing approaches for search-based techniques often rely on developer expertise and trial-and-error for reward engineering (i.e., defining the reward function) and reward shaping (i.e., fine-tuning the reward function). However, as high-level objectives are often abstract and declarative in nature, they cannot be easily mapped into mathematical expressions that faithfully reflect the true search objectives. As such, existing approaches often generate complex, nested, and error-prone reward functions for non-trivial search problems. This paper presents the Go4RES framework that uses goal models to support a systematic approach to reward engineering and reward shaping. By following a goal model decomposition strategy, Go4RES provides the structure that promotes modularity, traceability, and interpretability for reward engineering and reward shaping. We provide proof-of-concept demonstrations of Go4RES applied to different search-based application domains and construct the reward functions from the goal-based models of the optimization objectives for the respective search techniques.**

***Index Terms*—Goal models, Reinforcement Learning, Reward Functions, Search-based Software Engineering**

I. Introduction

Search-based approaches have been used to efficiently and effectively discover optimal or near-optimal solutions for complex problems, where the solution space may be vast and complicated. *Reward engineering* describes the process of developing a *reward function*[1] to guide the behavior of a computational search. Subsequently, *reward shaping* is the process of experimentally and empirically fine-tuning specific parameters and weights associated with these reward functions. Reward functions are used in a number of different computing subfields to define *objectives*, including Artificial Intelligence (AI) [1], robotic agent training with Reinforcement Learning (RL) [2], evolutionary search [3], [4], and game theory [5]. Existing research has expended significant effort towards improving the fundamental process and efficiency of algorithms in these fields; research has even shown that they can demonstrate better performance than humans in their designated tasks [6], [7]. While research to improve the efficiency of the algorithmic structure defining the search process is important to yield correct solutions for the search problems, equally important but largely unexplored in the literature is reward engineering and reward shaping [8], [9], [10], [11]. Traditional approaches often involve an ad-hoc approach to develop the reward functions, informed by simulation-specific constraints, domain expertise, empirical studies, and developer experience that is often subjective. This paper introduces a goal model-based approach to design and engineer reward functions for search-based approaches, thereby providing a systematic, reusable, and modular approach to define the objectives of a computational search and facilitate the fine-tuning of parameters.

In search-based approaches for optimization problems, a developer must define a reward function to guide the algorithm towards the desired behavior of the application. Search-based approaches include a wide range of applications across many disciplines, but share a commonality of optimizing towards a (specified) high-level objective, which is often declarative in nature. These high-level search objectives are typically defined using natural language. The corresponding "requirements" are then codified in a mathematical expression that represents an optima in the solution space. While an ad-hoc approach may be sufficient for simple optimization landscapes, such an approach may lead to overly complex and difficult to understand mathematical expressions in program code if the search objective requires a variety of updates over time, or if multiple subobjectives are involved. Another software engineering challenge for black-box machine learning approaches (e.g., deep learning, deep RL, etc.) in search-based settings is the lack of human-readable code. As the weights of the (AI) models are often uninterpretable [12] and determined automatically during training, the reward function is integral in shaping the behavior of the system and the correctness of the behavior exhibited when facing uncertainty. As such, there is a need for a rigorous approach to systematic reward engineering and reward shaping, which promotes better interpretability, modu-

[1]The reward function is also known as an *objective function*, a *fitness function*, a *cost function*, a *value function*, or a *reward signal* in different disciplines (e.g., reinforcement learning, evolutionary search, etc.). For discussion purposes, we use *reward function* as an umbrella term to capture the analogous terms across the different application domains.

larity, reusability, and traceability in search-based approaches for software assurance.

This paper introduces GO4RES (**G**oal-**O**riented requirements engineering for (**4**) **R**eward **E**ngineering and **S**haping), a goal model-based approach to systematic reward engineering and reward shaping. Specifically, we introduce several key insights and contributions in this work. First, our key insight is that the process of reward engineering shares many similarities and is analogous to goal-based requirements modeling, where the high-level abstract objective is often described in natural language and can be hierarchically decomposed into subobjectives. Second, goal-based models can decompose a high-level objective into functional and non-functional requirements/objectives, which supports separation of concerns and traceable reward functions. Functional subgoals can describe the corresponding task that the system is assigned, such as "drive to the destination". Non-functional subgoals describe properties for how the system should achieve the corresponding task, such as "drive as safely as possible". Third, by constructing the goal model through hierarchical decomposition, GO4RES enables the assessment of the behavior of the overall system relative to the contributions of individual subgoal evaluations, which can then be aggregated to construct the reward function for the overall system.

In order to apply a goal-model approach to reward function engineering and shaping, GO4RES uses the following process. First, high-level abstract objectives are decomposed hierarchically until leaf-level goals are reached (where a given goal can be concretely measured), thereby facilitating a top-down refinement of subtrees akin to component-based software engineering [13]. At the leaf-level goals, we associate *utility functions* [14] that can be used to evaluate a goal based on observable system properties, such as the "velocity of the rover'. The utility functions are discharged to *system components* (i.e., sensors) that measure the observable state and assign a scalar value to the leaf-level goal, thus providing a means to determine the degree of satisfaction for the given subgoal. Next, to facilitate reward shaping where developers fine-tune and experiment with a number of different configurations and priorities of subgoals, each edge between subgoals has a numerical weight associated with the decomposition. The weights can be adjusted and rebalanced by the developer to fine-tune the degree or level of emphasis placed on each subgoal in the resulting search solution. Weights can also be used to reflect the degree of confidence in a given sensor's measurement, particularly useful when the sensor's performance may be compromised (e.g., foggy weather hindering the capability of LiDAR sensors). Finally, the reward function for the computational search can be constructed in GO4RES through a traversal of the goal tree collating the aggregate utility functions.

We provide proof-of-concept demonstrations to illustrate the efficacy of GO4RES and its versatility by applying it to engineer reward functions for a number of different applications. While this paper includes examples to engineer reward functions for RL, evolutionary search, and game theory, GO4RES can be applied to search-based software engineering techniques that involve the refinement of a high-level objective into a mathematical expression to be used to guide the search process (e.g., search-based testing, machine learning, etc.). The remainder of this paper is organized as follows. Section II reviews the background material. Section III overviews the proposed framework and provides a running example. Section IV demonstrates an application of GO4RES to define an evolutionary search reward function. Section V demonstrates an application of GO4RES to define a game theory reward function. Section VI provides the related research for our work. Finally, Section VII concludes the paper.

II. BACKGROUND

This section overviews the background information and enabling technologies for GO4RES. First, we summarize reward engineering and reward shaping; we also discuss existing challenges associated with them. Second, we describe goal-based modeling in requirements engineering. Finally, we overview utility functions and their applications.

A. Reward functions and challenges

Reward functions are mathematical expressions that are used to specify an *objective* for system behaviors or search objectives. Specifically, they are used to define an optima or optimum for a search space for a number of distinct search-based techniques [15], [16], [17]. In RL, an agent may be trained to perform a given task based on a reward function (e.g., participating in a race, parking a car in a dense parking lot, etc.). Another rapidly developing application of RL is to customize machine learning models (e.g., large language models) via model fine-tuning, which uses reward functions to tailor a machine learning model to a given task using a feedback signal [18]. In evolutionary search, the reward is used to select for individuals that exhibit the best performance to iteratively evolve the population towards an optima (e.g., evolving an antenna to optimize for signal reception in space missions [19]). In game theory, the objective(s) of the game for the competing players may be defined using a reward function that the players seek to maximize (e.g., define the reward for a zero-sum game and the strategies players may employ). In many cases across these applications, researchers may have some prior domain knowledge or constraints about the search space described using natural language, but may require significant effort to transcribe them into a consistent mathematical expression due to subjectivity in the interpretation of the requirements and the abstract nature of the high-level objectives.

Reward engineering describes the process where developers design and engineer the corresponding mathematical reward function to guide the search process. Existing approaches often use an ad-hoc approach based on heuristics and prior knowledge [2], [8], [20], where the reward function is defined and tuned using trial-and-error, largely subjective with respect to developer expertise. The ad-hoc approach often leads to several key challenges with the generated reward function.

First, the desired objective of the reward function might be difficult to reflect in mathematical notations, especially when the high-level objective is abstract or complex [20], [21]. As such, there is a need for a systematic and rigorous approach to refine the objective into concrete and analyzable components. Second, reward functions in mathematical notations are often convoluted, nested, and increase in length and complexity as the system objectives become increasingly sophisticated [2], [22]. This raises challenges associated with the interpretability of the reward function, often relying on separate documentation to maintain traceability and explain how different components contribute to the high-level objective(s) of the reward. Third, existing approaches for reward engineering are largely not extendable [2], [22]. As the number of mathematical terms increases due to the addition of new requirements or objectives, revising and introducing new terms requires low-level code editing and rigor from developers to avoid disrupting the balance of existing reward function components. Finally, once the reward function has been engineered, the weights associated with each of the components are typically fine-tuned experimentally to achieve the desired output (i.e., reward shaping). Traditional approaches to reward shaping also require low-level code editing, where the changes may be difficult to track and can lead to incorrectly specified rewards due to the complex nested nature of the reward components.

B. Goal-based modeling

Goal-based modeling is used in requirements engineering to provide a hierarchical decomposition-based approach for developers and stakeholders to formally specify a software system's high-level goals and describe how those high-level goals should be satisfied. Specifically, high-level abstract goals are refined, typically using AND/OR decomposition, into subgoals, each of which describes and corresponds to a specific subobjective that addresses a part of the high-level goal. By iteratively refining the subgoals into low-level requirements, goal-based modeling provides a hierarchical approach to decompose seemingly complex or abstract high-level goals down to requirements. Goal models provide several advantages. First, they enable stakeholders to better understand, establish, and describe achievable goals by decomposing complex, abstract objectives into more concrete objectives and design solutions to satisfy the concrete objectives. These concrete objectives in turn contribute to the satisfaction of the overarching abstract objective. Second, they impose a hierarchical decomposition as a means to organize the goals by levels of abstractions. Third, goal models provide traceability, describing how a single low-level requirement may contribute to the high-level objective of the system.

KAOS [23] is a popular approach for goal-oriented requirements modeling to decompose high-level objectives down to leaf-level system requirements. KAOS goal models support a goal-oriented analysis of a system's high-level objectives, and use hierarchical AND/OR decompositions to refine them into low-level requirements, forming a tree-like structure. *Goals* represent the objectives of the system at different levels of abstraction. *Edges* between goals refine a goal into several subgoals, each of which contributes to the parent goal. In KAOS, goals can be refined through AND or OR decompositions. For AND-decomposed goals, all the children subgoals have to be satisfied in order for the parent to be considered satisfied. In contrast, for OR-decomposed goals, one or more children subgoal(s) have to be satisfied for the parent to be considered satisfied. At the leaf-level, *system components*[2] are assigned responsibility to satisfy the requirements associated with the leaf-level goals.

C. Utility functions

Utility functions have been used to monitor self-adaptive systems during run time to measure desired properties or performance of their given task [24], [25], [26], [27], [28]. Expression (1) shows the structure of a utility function, where u is a scalar value normalized between $[0, 1]$ and the function f is used to calculate the utility value from a system state vector v. For example, a utility function may capture the current *velocity* of an autonomous rover, which may be used to assess whether the rover is exceeding the operational speed limit constraints.

$$u = f(v) \tag{1}$$

Previous work has demonstrated that utility functions can also be used with goal models for the run-time assessment of goals for satisfaction or satisficement by attaching them to leaf-level nodes in KAOS goal models [27], [28]. For example, the requirement "move to the destination" for an autonomous rover can be quantified by the utility function shown in Expression (2), where u calculates the Euclidean distance between the rover's current coordinates estimated by its GPS unit and the target destination's coordinates. By traversing the goal model tree and aggregating the values measured by the utility functions, utility functions have also been used to monitor the goal satisfaction of self-adaptive systems during run time [28].

$$u = \text{dist}(gps.getCoordinate(), target.coordinate) \tag{2}$$

III. Methodology

This section describes how modeling is applied at design time in Go4Res to systematically engineer reward functions, which can be used for the aforementioned search-based applications or areas. Our approach also facilitates reward shaping, the fine-tuning process that provides optimization as part of reward engineering. For illustration purposes, we include a running example of Go4Res, where we demonstrate how our framework can be used to engineer, design, and refine the objectives of an autonomous rover whose task is to park itself in a parking lot using RL. In RL, the reward function is used to indicate to the model in training how well it is currently achieving its intended task [21], [29]. Figure 1 shows the RL

[2]This paper uses the term system components to describe *agents* in KAOS goal models to avoid confusion with overloaded use of the term *agents* in RL and other agent behavior training domains.

training loop, where the agent performs some action in the environment, observes the resulting state, calculates a reward value based on the corresponding reward function, and updates its instructions. Through trial-and-error and iteratively taking actions, the agent learns to associate "good" actions with high rewards. Since the reward function is the only means for the agent to obtain feedback on its performance during training, reward engineering is vital in RL to ensure that the reward function correctly reflects the desired behaviors for the agent in training. Figure 2a shows the goal model for the autonomous rover. Parallelograms denote goals, edges with yellow circles denote AND decompositions, split edges denote OR decompositions, yellow roundtangles denote utility functions, and hexagons denote system components. We next describe the elements of our framework in turn and show how the goal models can be automatically processed to generate the corresponding reward function, shown in Figure 2b.

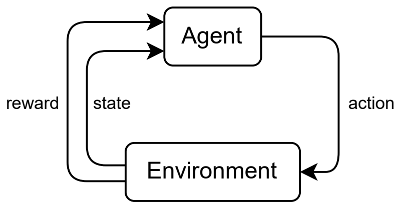


Fig. 1: The standard training loop for RL algorithms.

A. Goal Decomposition

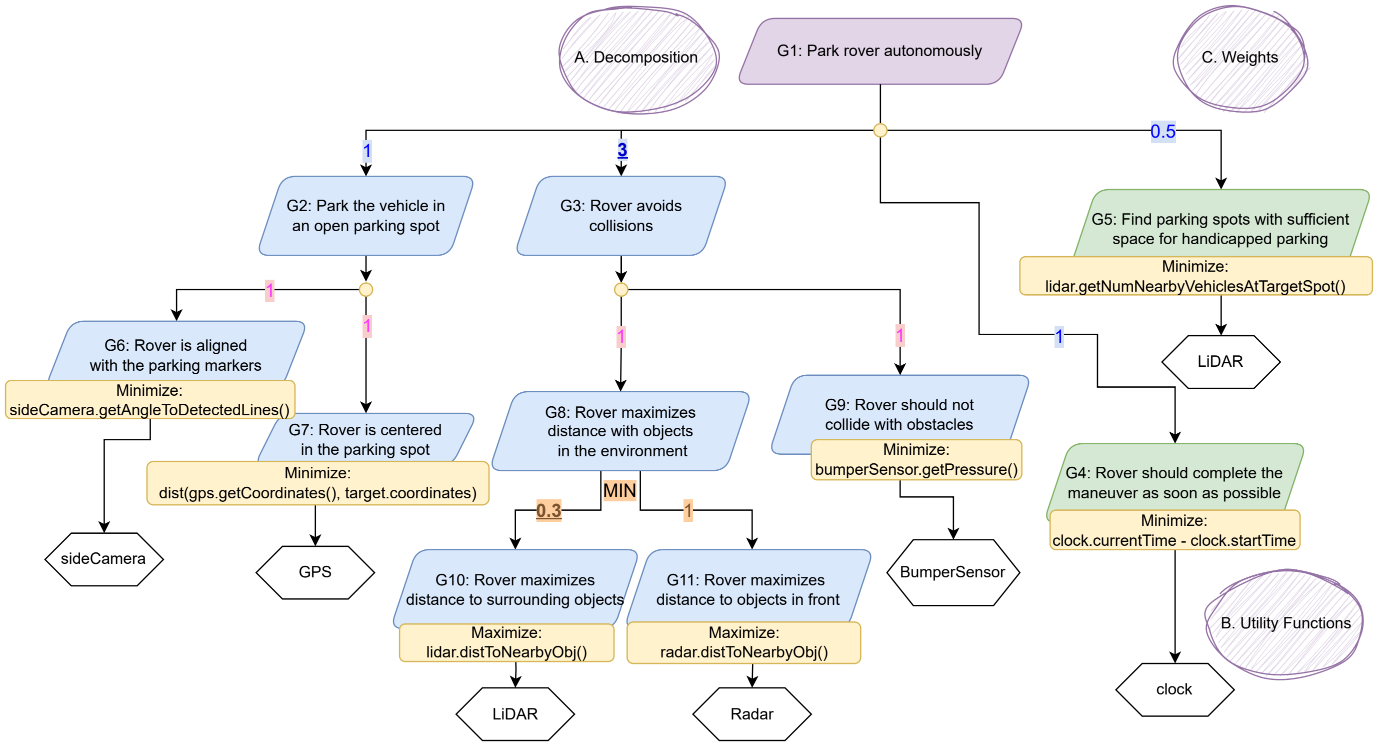
GO4RES uses an analogous approach to goal-based requirements modeling to support reward function engineering by decomposing high-level or complex optimization problems using traditional goal modeling techniques down to low-level requirements that can be monitored and assessed by system components. By taking a goal-based approach, GO4RES supports an inherently modular approach to reward engineering and provides a platform for developers to reason how an abstract objective can be further decomposed into concrete subobjectives. Similarly, traditional search optimization problems may have a primary objective while attempting to optimize another dimension (e.g., optimizing to find the minima in a search space and also to minimize computation time). In search-based engineering, a developer can specify the optimization problem as a minimization problem or a maximization problem.[3] By default, GO4RES supports a minimization problem where the objective is to minimize the reward at the root goal. Developers may also use GO4RES to describe and engineer the rewards for a maximization problem, but they must maintain consistency in the description of the utility functions of the subgoals accordingly. Next, we describe each of the supporting goal decomposition strategies in GO4RES in turn.

[3]In search-based settings, minimization problems can be converted to their equivalent maximization problems by maximizing their negation (i.e., Minimizing the function $f(x)$ is equivalent to maximizing $-f(x)$) [30].

a) Hierarchical Decomposition: During requirements specification, a high-level objective is defined using natural language, such as '`G1: Rover parks autonomously`' (denoted by the purple parallelogram root goal in Figure 2a). The high-level objective is then refined into subgoals, each contributing to the root goal. For example, our root Goal G1 is decomposed into four subobjectives: G2, G3, G4, and G5. These subgoals can then be further refined iteratively, forming a hierarchical tree until we reach low-level requirements that are measurable and testable. For instance, Goal '`G2: Park the vehicle in an open parking spot`' might be further decomposed into two subgoals: '`G6: Rover is aligned with parking markers`' and '`G7: Rover is centered in the parking spot`'. At the leaf-level, individual requirements can be measured in the observable environment with a corresponding system component, such as the LiDAR unit that measures distance to nearby objects (e.g., Goal '`G5: Find parking spots with sufficient space for handicapped parking`' and Goal '`G10: Rover maintains a minimal distance to surrounding objects`'). As such, the hierarchical decomposition provides a well-defined structure for developers to systematically refine an abstract or complex high-level goal into modular and quantifiable requirements (i.e., measurable in scalar value).

b) Functional and non-functional objectives: The second strategy supported by GO4RES is the explicit decomposition of the parent root goal into either functional or non-functional objectives. Specifically, functional objectives (indicated by blue) are those that contribute to the intended task of the system, such as Goal '`G3: Rover avoids collisions`'. Non-functional objectives (indicated in green) are those that specify properties for how the system should achieve those tasks, such as '`G4: Rover should complete the maneuver as soon as possible`'. As such, this decomposition strategy supports a "separation of concerns" approach to goal modeling, thereby enabling modular specification of functional objectives and non-functional (e.g., performance-related) objectives.

c) AND / OR refinements: When goals are decomposed into multiple subgoals, developers can use either an AND or an OR refinement. In AND refinement, each subgoal contributes to the parent goal (i.e., parents are a summation of the children subgoals' rewards). The decomposition of Goal '`G2: Park the vehicle in an open parking spot`' shows an example of an AND refinement, denoted by the yellow circle connecting the edges. Both goals, '`G6: Rover is aligned with the parking markers`' and '`G7: Rover is centered in the parking spot`', contribute equally to the parent goal and reward the rover to park into the designated spot correctly. In OR refinement, one or more subgoals may contribute to the parent goal. As such, the developer must specify whether the minimum or the maximum (depending on the context used for decomposition) utility value from the children subgoals should be used for the value of the parent goal. The decomposition of


A. Decomposition
G1: Park rover autonomously
C. Weights
1
3
0.5
G2: Park the vehicle in an open parking spot
G3: Rover avoids collisions
G5: Find parking spots with sufficient space for handicapped parking
Minimize: lidar.getNumNearbyVehiclesAtTargetSpot()
1
1
1
1
1
G6: Rover is aligned with the parking markers
Minimize: sideCamera.getAngleToDetectedLines()
G7: Rover is centered in the parking spot
Minimize: dist(gps.getCoordinates(), target.coordinates)
G8: Rover maximizes distance with objects in the environment
G9: Rover should not collide with obstacles
Minimize: bumperSensor.getPressure()
LiDAR
G4: Rover should complete the maneuver as soon as possible
Minimize: clock.currentTime - clock.startTime
MIN
0.3
1
sideCamera
GPS
G10: Rover maximizes distance to surrounding objects
Maximize: lidar.distToNearbyObj()
G11: Rover maximizes distance to objects in front
Maximize: radar.distToNearbyObj()
BumperSensor
B. Utility Functions
LiDAR
Radar
clock

(a) The goal model used to decompose the abstract root objective of the rover into single requirements.

```
Rover_reward_fn = (1/5.5) * [(1/2) * sideCamera.getAngleToDetectedLines() + (1/2)* dist(gps.getCoordinates(), target.coordinate())]      // G2
                + (3/5.5) * [(1/2) * min( (0.3 * (1 - lidar.getNearestObjDist())), (1 * (1 - radar.getNearestObjDist())) ) + (1/2) * bumperSensor.getPressure()]   // G3
                + (1/5.5) * (clock.currentTime - clock.startTime)      // G4
                + (0.5/5.5) * (lidar.getNumNearbyVehicleAtTargetSpot())      // G5
```

D. Compiled Utility Function

(b) The constructed reward function used in the RL training of the rover, where highlighting colors correlate to nesting level of edge weights in the goal model (highlighted in the respective colors).

Fig. 2: Goal model and the corresponding reward function for an autonomous rover learning to park using RL.

Goal '`G8: Rover maximizes distance with objects in the environment`' shows an example of an OR refinement, denoted by two OR edges where either Subgoal '`G10: Rover maximizes distance to surrounding objects`' measured by the LiDAR unit or Subgoal '`G11: Rover maximizes distance to objects in front`' measured by the radar unit can contribute to the parent goal. In this case, we use the minimum value between the measured utility values from the subgoals (denoted by the annotated "MIN" label in the OR edge), since we want to promote the maximum distance to the nearest detected object by either the LiDAR unit or the radar unit in order to avoid collisions.

B. Utility functions

In GO4RES, the utility functions serve as the "interface" between the goal models and the observable environment. Specifically, leaf-level goals are assigned a corresponding system component that can measure observable properties of the environment to assess how well the system's behavior is contributing to the system's objectives. For example, Goal '`G7: Rover is centered in the parking spot`' uses the rover's GPS unit to measure the Euclidean distance between the rover and the center of the desired parking spot. In the utility function, the developer provides the corresponding code excerpt that can measure and evaluate the goal by monitoring observable properties of the environment using the system components (e.g., GPS). For Goal '`G7: Rover is centered in the parking spot`', the utility function may be provided as follows: dist($gps.getCoordinate()$, $target.coordinate$). Finally, the utility values are normalized between the scalar values $[0, 1]$ for each system component input, indicating how well a leaf-level goal contributes to the objective of the parent goal. The normalization of the utility functions also ensures that the values associated with subgoals are within the same distribution, preventing utility values with large units of measurement from dominating the reward function.

C. Weights

The final modeling strategy used in GO4RES is associating weights with edges in the subtree decomposition, shown as numbers annotating the edges. In GO4RES, edges are assigned a default weighting of 1.0. Subtrees with important objectives, such as those pertaining to safety, may be assigned a higher weight to indicate priority (e.g., Goal G3 with a weight of 3.0, bolded and underlined in Figure 2). In contrast, subtrees with less critical objectives may be assigned lower weights. By associating weights with each branch of the tree, developers can systematically adjust the emphasis on specific subobjectives instead of editing the reward function at a code level, thus facilitating rapid but systematic reconfiguration of the reward function during reward shaping. For example, the Goal '**`G3: Rover avoids collisions`**' is AND-decomposed to the two subgoals Goal '**`G8: Rover maximizes distance with objects in the environment`**' and Goal '**`G9: Rover should not collide with obstacles`**', with weights distributed equally for both. If the rover is being trained in a dense or tightly parked parking lot, then the developer may wish to adjust the weights associated with this decomposition to allow the rover to be in closer proximity to obstacles but associate a higher penalty to collisions. For the leaf-level nodes, weights enable developers to increase or decrease the trust (i.e., confidence) placed on some sensors. For example, the Goal '**`G10: Rover maximizes distance to surrounding objects`**' has a weight of 0.3 associated with the goal (bolded and underlined). This could reflect that the rover is being trained in foggy conditions, where the LiDAR's performance is often degraded.

D. *Constructing the reward function*

Once the high-level objective has been refined into low-level requirements, developers can use the goal model to construct the corresponding reward function by traversing the tree and aggregating the utility functions. Specifically, GO4RES uses a post-order traversal of the goal tree when constructing the reward function. When a leaf-level node is visited, we include its utility function in the mathematical expression along with its associated edge weight. For some utility functions, developers may want to maximize a property associated with the observable environment in a minimization problem (i.e., '**`G10: Rover maximizes distance to surrounding objects`**'). To remain consistent with the nature of the minimization problem, GO4RES uses the complement of the utility function for these leaf-level goals instead. The complement of the utility value will remain normalized between $[0, 1]$, since the original utility values are normalized. For example, GO4RES uses $[1 - lidar.getNearestObjDist()]$ for Goal G10's utility function: *maximize distance to nearby objects*. At parent nodes, the reward function components that are associated with the subobjective of the goal are aggregated. For an AND-decomposed node, we construct a linear weighted sum of its aggregate children nodes, with their respective weights normalized to a value of 1.0. For example, the total weight of Goal G1's decomposition is 5.5. As such, Subgoal G2 (with edge weight 1) is multiplied by a factor of 1.0/5.5, while Subgoal G3 (with edge weight 3) is multiplied by a factor of 3.0/5.5. The values are then aggregated through a linear weighted sum. For OR-decompositions, GO4RES first applies the edge weights associated with the decomposition to their respective children, then uses either the minimum or the maximum value (abbreviated as "min/max" for brevity) of the children based on the type of decomposition strategy specified by the developer in the goal model (i.e., $min/max(w_1 * g_1, w_2 * g_2, \cdots, w_n * g_n), \forall g_i \in G, w_n \in W$, where g_i are subgoals of parent G and w_i are weights associated with the respective edges). For example, Goal G8's OR-decomposition uses the minimum utility value from Goal G10 and Goal G11, denoted by the "MIN" annotation in the goal model. By traversing the tree and propagating the reward function components of each subtree recursively up to the root goal, the full reward function for the goal tree can be constructed and collected at the root node at the end of the traversal.

Figure 2b shows the final reward function expression obtained from the goal model for the autonomous rover in Figure 2a. While the same reward function may be obtained by developer expertise in a traditional ad-hoc approach, manual reward engineering can be complex, tedious, and error-prone due to the nested values. In addition, reward functions can be difficult to interpret due to the increasingly complex variables and logic. By using a goal model approach and constructing the corresponding mathematical expression, GO4RES enables developers to update reward functions at the goal model level, rather than editing low-level code. During reward shaping, a developer can also adjust weights associated with individual nodes or subtrees, rather than editing the weights at the low-level code (an error-prone process). Finally, a goal model approach facilitates the reuse and extension of the reward function, as developers can capture changes to requirements by revising the model to include additional constraints, functional goals, and non-functional goals. By revising at a model level, developers can edit weights, reconstruct, and execute the experiments to achieve rapid yet systematic reconfigurations of reward functions.

In summary, GO4RES facilitates the development, fine-tuning, exploration, and optimization for both reward engineering and reward shaping. In reward engineering, developers can explore different goal refinements, decompositions, and hierarchies of goals to discover the reward function that leads to the best solution in the search space for the primary functional objective. In reward shaping, developers can adjust the weights associated with the decomposition to explore various search paths in the search space to achieve the primary objective (i.e., how we achieve the primary objective).

IV. GO4RES FOR EVOLUTIONARY SEARCH

This section describes how GO4RES can be applied to systematically engineer the reward function for evolution-inspired optimization algorithms. Evolutionary search is a powerful optimization technique that is used to solve or approximate solutions for complex problems when there exists no exact method to solve the problem [31]. In evolutionary search, the complex problem is typically mapped to a solution space, then a reward function is used to guide the evolution of a population of individuals (each encoding different information) to discover solutions that are near optimal [31], [32]. Research has demonstrated that different types of evolutionary search algorithms can be strategically applied to optimize different search spaces, such as particle swarm optimization [33], ant colony optimization [34], genetic algorithms [32], etc., all of which involve a reward function. When applied to reward engineering in search optimization strategies, GO4RES enables the systematic decomposition of high-level search objectives into different subobjectives, facilitating traceability and interpretation of how each subobjective may contribute to the evolutionary search.

One interesting use case of evolutionary search has been proposed by Hornby *et al.*, where evolutionary search is used to evolve optimal antenna receivers for spacecrafts used on NASA's ST5 mission [19]. Specifically, the evolved antenna attempts to minimize the Voltage Standing Wave Ratio (VSWR) at different frequencies, along with two different gain components. Expression (3) gives the optimization reward function used in their work [19], where *rms* is the root-mean-square function for a target value t against the VSWR value v_f at various frequencies $f, \forall f \in F$, where F is the set of all frequencies under optimization.

$$\begin{aligned}\sum_f & rms(3, v_f)^5 + rms(1.5, v_f) + rms(1.0, v_f) \\ & + min(0, 15.25 - g_{f_0}) + min(0, 10.25 - g_{f_{20}})\end{aligned} \tag{3}$$

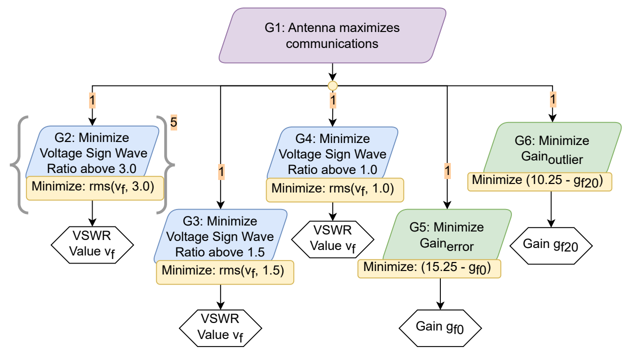
Next, we demonstrate how we can "reverse engineer" a GO4RES goal model from the information and reward function provided by Hornby *et al.*'s experiments in order to take advantage of the GO4RES platform and its reward engineering and shaping capabilities. Figure 3a shows the corresponding goal model that we developed for the expression described in Hornby *et al.*'s optimization problem. We first began with the domain knowledge provided in their work, including the reward function and the corresponding text that describes the components of the expression. Then we initialize the root goal of the tree based on the primary objective of the search shown in purple: '`G1: Antenna maximizes communications`'. Next, we create a subtree for each component in Expression (3) and create the corresponding succinct text description based on the original description. Utility functions are used to bind each of the terms in the expression to the goal model and describe how they should be evaluated. For example, the second term in Expression (3), *rms(1.5, v_f)*, is associated with the second goal subtree to minimize the VSWR with the utility function '`minimize: rms(1.5,` v_f`)`'. We used the default weighting of 1.0 for each subtree, since the original work does not apply any weighting to the terms. The developer may use GO4RES to adjust these weights to update the reward function to emphasize or de-emphasize specific terms. For example, if the last term to minimize the $gain_{outlier}$ is no longer important, then the weight associated with the corresponding subtree can be reduced to 0 (or any rational number between 0 and 1, forming a linear scale to denote the importance of the subtree from no effect to standard weighting). Finally, the first term in Expression (3), *rms(3.0, v_f)^5*, is raised to the power of 5 to assign a heavy penalty to any evolved antennas that have a higher VSWR than 3.0. We use the notation $(▱)^5$ to denote the corresponding penalty in the goal model. As a result, the GO4RES goal model that we developed for the evolved antenna's reward function provides ease of interpretability and visualizes the information used to construct the reward function when compared to the original text description and mathematical reward function.

By constructing the goal model that we developed into its reward function counterpart (see Figure 3c, where the functional components related to minimizing VSWR are highlighted in blue while the non-functional components related to minimizing gain are highlighted in green), GO4RES can obtain the same expression shown in Expression (3), which can be used as the reward function for the optimization algorithm to obtain the same result as the original work. In addition, GO4RES facilitates reward shaping in the reward function as developers can now adjust the weights associated with edges by revising the goal model, rather than editing the reward function directly. After the weights associated with the edges are updated in the model, developers can simply reprocess the updated goal model to obtain the revised reward function for new experiments.

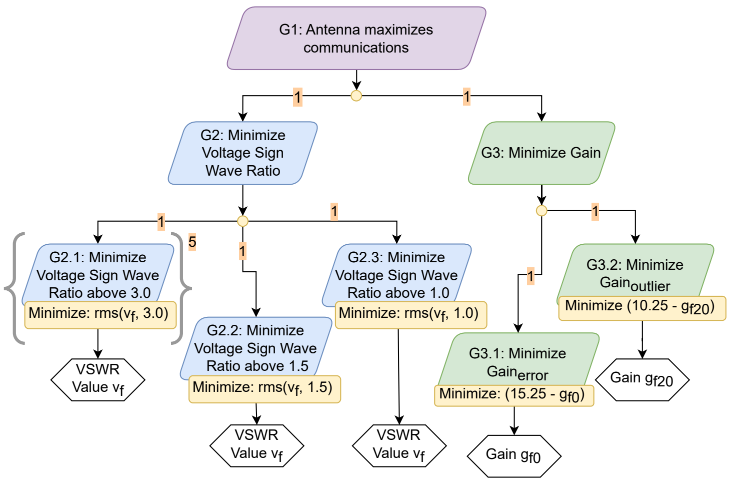
In order to take full advantage of GO4RES and its capabilities, we introduce a level of abstraction to the goal model to encapsulate the functional and non-functional subtrees in Figure 3b. In this example, each component associated with minimizing the VSWR is first aggregated to a parent functional goal (i.e., Goal '`G2: Minimize Voltage Sign Wave Ratio`'), while each component associated with minimizing gain is aggregated to a parent non-functional goal (i.e., Goal '`G3: Minimize Gain`'). Figure 3d shows the new reward function that is obtained after traversing the goal model with hierarchical decomposition. This reward function still faithfully reflects the original expression used in the evolutionary search and captures all the optimization components. In addition, this approach better facilitates reward shaping, as developers can adjust the emphasis between the functional and the non-functional subtrees by revising their respective weights (i.e., the weights associated with the decomposition of Goal '`G1: Anetnna maximizes communications`'), as opposed to editing each subobjectives iteratively to reflect the change. As part of the reverse engineering process, Figure 3b shows that abstraction can introduce a parent goal for the leaf-level goals, which enables legacy reward functions to better make use of the GO4RES framework to improve reward engineering, facilitate reward shaping, and improve interpretability and documentation.

V. GO4RES FOR GAME THEORY

We demonstrate the modularity and flexibility of GO4RES by applying it to a game theory application. In GO4RES, subtrees associated with the decomposition in the goal models are modular and self-contained, and thus can be reused or replaced to update the search objective. Specifically, the subtree decomposed from various system objectives and non-functional objectives can form "building block" components.



(a) The goal model engineered based on the reward function described in Hornby *et al.*'s work [19].


G1: Antenna maximizes communications
1
1
G2: Minimize Voltage Sign Wave Ratio
G3: Minimize Gain
1
1
1
1
1
5
G2.1: Minimize Voltage Sign Wave Ratio above 3.0
Minimize: rms(v_f, 3.0)
VSWR Value v_f
G2.2: Minimize Voltage Sign Wave Ratio above 1.5
Minimize: rms(v_f, 1.5)
VSWR Value v_f
G2.3: Minimize Voltage Sign Wave Ratio above 1.0
Minimize: rms(v_f, 1.0)
VSWR Value v_f
G3.1: Minimize Gain_error
Minimize: (15.25 - g_f0)
Gain g_f0
G3.2: Minimize Gain_outlier
Minimize (10.25 - g_f20)
Gain g_f20

(b) Abstraction is introduced to the goal tree to create functional and non-functional subtrees.

$$\text{optimization\_reward\_fn\_vf} = \text{rms}(3, v_f)^5 + \text{rms}(1.5, v_f) + \text{rms}(1.0, v_f) + \min(0, 15.25 - g_{f0}) + \min(0, 10.25 - g_{f20})$$

(c) The constructed reward function that reflects Expression (3).

$$\text{optimization\_reward\_fn\_vf} = 1 * (\,\text{rms}(3, v_f)^5 + \text{rms}(1.5, v_f) + \text{rms}(1.0, v_f)\,) + 1 * (\,\min(0, 15.25 - g_{f0}) + \min(0, 10.25 - g_{f20})\,)$$

(d) The equivalent reward function with edge weights from the newly introduced hierarchy level.

Fig. 3: The goal model for the reward function described in the EA configuration to optimize the antenna for the ST5 mission.

These building blocks can then be reused and composed in different combinations to support different searches, which can be used to explore uncertainty and in game-based testing [35], [36], [37].

Recently, researchers have demonstrated that game theory can be applied to model different scenarios as a game and explore interactions and strategies that players of the game may use to achieve their objectives [38], [39]. For example, security administrators may model themselves as a *defender* in a game against a malicious *attacker*, where the attacker's objective is to infiltrate the system's resources and disable nodes on the computer network, leading to a denial-of-service attack for legitimate users. The defender, in contrast, seeks to defend against the attack by deploying firewalls at a limited number of nodes to mitigate the attack. As such, this theoretical game examines the optimal nodes that an attacker should target and the optimal nodes that the defender should prioritize for protection. Next, we demonstrate how the objectives of the attacker, the defender, and the different types of strategies they may respectively use can be modeled using GO4RES.

First, we design and describe the goal tree for the defender. Figure 4a shows an example goal tree, where the fine-grained details of the subtrees are abstracted away. The blue subtree represents the defender's functional objective (e.g., protect its network asset), while the green subtree represents the non-functional objective (i.e., performance constraints) that the player may use. Specifically, this player may focus on a defensive strategy, where the emphasis is on mitigating the maximum number of attacks to ensure the lowest downtime of system resources. The expression below the goal model shows the corresponding reward function, where the components of each expression are highlighted in the respective subtree's colors. This goal model describes the most basic strategy that the defender player may use to protect its assets.

a) Modular and changeable subtrees: To explore different configurations and strategies, GO4RES facilitates the rapid reconfiguration of player objectives by allowing subtrees in the goal model to be swapped. For example, suppose the developer now wants to explore an aggressive strategy to identify the attacker's identity for the authorities to apprehend them. As such, the original green defensive non-functional subtree in Figure 4a is now replaced with a different, orange aggressive non-functional subtree in Figure 4b, where the new objective now rewards the defender for capturing the attacker (e.g., use a security node to observe and collect information from the attacker). By incorporating the new goal tree into the corresponding reward function, the developer may train the new player without the need to completely redefine the defender player's functional rewards. The new reward function below the goal model now shows that the green defensive component from Figure 4a's reward function has been replaced with the orange aggressive component associated with Goal G4.

Next, Figure 4c shows that the defender's non-functional objective has now been replaced by another *deceptive* objective in turquoise. This non-functional objective also demonstrates an OR-decomposition, where the defender is rewarded based on the maximum number of attacks mitigated (indicated by the annotation "MAX") by either a) masquerade a honey pot node as a vulnerable node to attract or b) divert the attacker's attention while moving valuable resources to a different obfuscated node. We use the maximum utility value between the utility functions *f5.1* and *f5.2* after applying weights *w5.1* and *w5.2* respectively. As a result, we demonstrated in this section how functional and non-functional objectives in GO4RES can essentially be used as building blocks to compose different combinations of system objectives.

b) Extendable goal models: GO4RES also supports the ability to extend subtrees with additional subobjectives to form systems with multiple non-functional objectives. The self-contained subtrees that describe functional or non-functional objectives of the system can be combined to form reward functions with multiple non-functional objectives. For example, a defender player may use an aggressive strategy and attempt to capture or monitor the attacker to gather information (denoted by the orange subtree). The defender player may also choose to use an additional *deceptive* strategy, where the defender also diverts the attacker's attention away from their valuable resources. Figure 5 shows the corresponding revised goal model that reflects the additional requirement. This goal model shows that the original defender function subtree is now extended with two different non-functional subtrees: aggressive (e.g., the use of honeypot nodes) and deceptive (e.g., masquerade the honey pot nodes to appear as vulnerable computation nodes to the attacker). We also show the corresponding reward function for this extended goal model, where the colors of each component reflect the respective subtree's functional or non-functional goal. By using a modular approach, GO4RES provides the ability to define systems with multiple objectives, thereby providing extensibility, reusability, and flexibility.

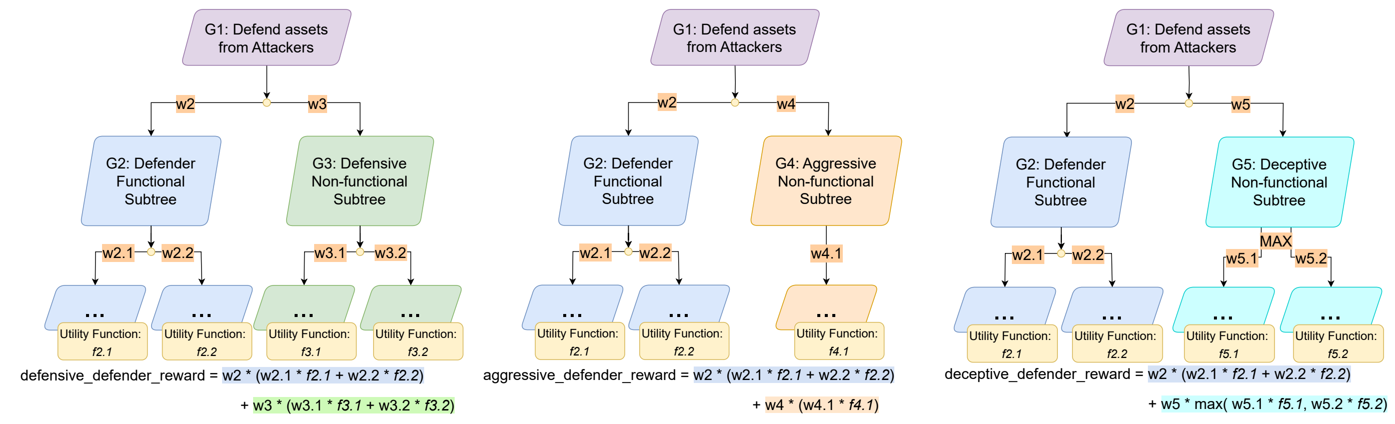
VI. RELATED WORK

This section overviews the related work for GO4RES. First, we discuss goal model decomposition in the context of search-based settings. Second, we overview existing efforts to improve reward function engineering and reward shaping. Third, we review existing approaches that use goal models for the run-time assessment of software systems.

A number of research studies have addressed goal decomposition of system objectives in requirements engineering to refine complex high-level objectives into small subgoals. van Lamsweerde [23] proposed the KAOS modeling language, originally intended to refine and capture high-level objectives into low-level requirements. Marthi [40], Rietz *et al.* [41], Juozapaitis *et al.* [42], and other researchers have addressed *reward decomposition*, where a system's task is broken up into corresponding subtasks. These works focus on creating different controllers for the system for each given subtask, rather than a systematic approach to reward function engineering and reward shaping. Li et al. [43] proposed a goal model-based approach to decompose tasks for curriculum learning in self-learning adaptive systems. Their work uses decomposition to describe different objectives for a given self-learning adaptive system (e.g., different tasks associated with a household cleaning robot, such as sweeping, mopping, doing the dishes, etc.). Then Q-learning and other approaches are used to learn the controller for each subtask, which are combined to form the controller for the system. Li *et al.* [44] proposed the use of pre-trained language models to provide suitable goal decomposition for multiagent RL, but focuses on automatic subgoal generation instead to better train agents on their given subtasks.

Other research directions have explored different methods to improve reward function engineering. However, these approaches do not provide a goal model-based framework that can be constructed into the corresponding reward function. Mallozzi *et al.* [9] proposed MoVEMo, an approach that combines finite state machines, program verification, and run time monitors as an early work to systematic reward function engineering. However, their work does not use a goal model-based approach to refine high-level objectives into low-level requirements, which are then compiled into a traditional reward function in GO4RES. Instead, their work produces a separate run-time entity that observes the defined property in the finite state automata and assigns a reward to the agent. Bahdanau *et al.* [45] proposed a framework that trains RL agents to understand task objectives specified in natural language. Their approach focuses on training "language-aware" agents. Hu *et al.* [46] proposed a bi-level optimization approach to refine weights in reward shaping for RL agents. Their approach focuses on adaptively using different components of an existing reward function by emphasizing components that positively contribute to the objective and suppressing components that negatively contribute to the objective. Yang *et al.* [47] studied reward functions and reward strategies of RL systems in continuous integration testing, but does not address reward function engineering. Cuayáhuitl *et al.* [48] explored a method to train multi-domain dialogue agents by using a decomposition strategy. Particularly, they describe and decompose dialogues at different levels of granularity and execute actions as either single actions or a composite action. However, their work focuses on modularity and decomposability of the (input) language level for natural language processing, and does not address goal model decomposition nor reward engineering.

Finally, a number of different researchers have applied goal models for the run-time assessment of self-adaptive systems. Ramirez *et al.* [25] introduced Loki, an automatic tool that facilitates the identification of goals that may have insufficient mitigations to prevent undesired behaviors. Fredericks *et al.* [26] proposed AutoRELAX, a fuzzy logic-based specification language to identify sources of environmental uncertainty for self-adaptive systems. Langford *et al.* [27] proposed the use of KAOS goal models and utility functions to assess the run-time satisfaction of system goals. Duran and Mussbacher [49] proposed to use goal models to determine run-time conflicts when reusing different software artifacts. Eteke *et al.* [50] proposed the use of hidden Markov models to monitor run-time executions for reward learning. However, these works do not address the use of goal models during design time to facilitate the design of reward functions.


G1: Defend assets from Attackers
w2
w3
w4
w5
G2: Defender Functional Subtree
G3: Defensive Non-functional Subtree
G4: Aggressive Non-functional Subtree
G5: Deceptive Non-functional Subtree
w2.1
w2.2
w3.1
w3.2
w4.1
MAX
w5.1
w5.2
Utility Function: f2.1
Utility Function: f2.2
Utility Function: f3.1
Utility Function: f3.2
Utility Function: f4.1
Utility Function: f5.1
Utility Function: f5.2
defensive_defender_reward = w2 * (w2.1 * f2.1 + w2.2 * f2.2) + w3 * (w3.1 * f3.1 + w3.2 * f3.2)
aggressive_defender_reward = w2 * (w2.1 * f2.1 + w2.2 * f2.2) + w4 * (w4.1 * f4.1)
deceptive_defender_reward = w2 * (w2.1 * f2.1 + w2.2 * f2.2) + w5 * max(w5.1 * f5.1, w5.2 * f5.2)

(a) A goal model with a functional and non-functional objective.

(b) Modular non-functional subtree(s) can be swapped for another non-functional subtree.

(c) The non-functional subtree has been swapped with another non-functional subtree that contains an OR decomposition.

Fig. 4: By building modular subtrees via functional and non-functional decomposition in GO4RES, subtrees form building blocks that can be reused and composed to form different agents with various objectives. The respective reward functions constructed from traversing the goal trees are shown at the bottom of each subfigure, highlighted by corresponding subtree coloring.

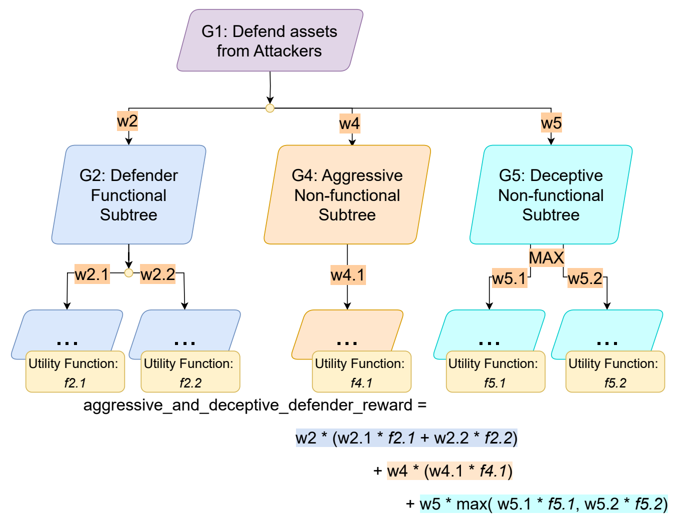

G1: Defend assets from Attackers
w2
w4
w5
G2: Defender Functional Subtree
G4: Aggressive Non-functional Subtree
G5: Deceptive Non-functional Subtree
w2.1
w2.2
w4.1
MAX
w5.1
w5.2
Utility Function: f2.1
Utility Function: f2.2
Utility Function: f4.1
Utility Function: f5.1
Utility Function: f5.2
aggressive_and_deceptive_defender_reward =
w2 * (w2.1 * f2.1 + w2.2 * f2.2)
+ w4 * (w4.1 * f4.1)
+ w5 * max(w5.1 * f5.1, w5.2 * f5.2)

Fig. 5: Multiple non-functional subtrees can form complex agents that exhibit both non-functional objectives.

VII. Conclusion

Due to their ability to discover innovative solutions for difficult to solve problems, search-based software engineering is becoming increasingly prevalent. However, a common challenge associated with search-based software engineering is reward engineering and reward shaping, where the developer must describe the search objectives using a mathematical expression. This paper introduced GO4RES, a systematic approach for refining high-level abstract objectives into low-level concrete objectives. Utility functions are associated with leaf-level goals, providing a mapping between the goal model and the observable state of the search environment. Preliminary results show that GO4RES can provide a systematic stepwise process to generate reward functions, leading to a modular and traceable model. We demonstrated the application of GO4RES to several distinct search-based techniques, including RL, game theory, and evolutionary search.

GO4RES provides a goal-based approach for developers to systematically design and engineer reward functions into modular reward function components. We note that domain expertise is needed to specify the utility functions that measure the utility value associated with the intended behavior for each of the low-level subobjective goals. We provide a process for developers to reason about the decomposition of their high-level system objectives into manageable, concrete subobjectives that can be associated with system components and measured. Our model-based approach provides versatility across different domains with search objectives, modularity in the decomposition of subobjectives, reuse of those subobjectives, and facilitates reward shaping by enabling adjustments to the reward function by revising the goal model instead of the complex reward function. Finally, GO4RES also provides interpretability of the reward function by associating reward function components with subgoals, thereby providing a means to visualize and trace how each reward component and to what degree do they contribute to the root goal.

Future work includes using GO4RES in search-based and game-based testing for uncertainty exploration, where GO4RES's support for rapid reconfiguration may be exploited to explore different types of uncertainty for AI-based software systems [35], [36], [37]. Other research directions include extending GO4RES to include objectives of other agents (i.e., players) in the environment for multi-agent zero-sum games.

References

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[2] S. Ibrahim, M. Mostafa, A. Jnadi, H. Salloum, and P. Osinenko, “Comprehensive overview of reward engineering and shaping in advancing reinforcement learning applications,” *IEEE Access*, 2024.

[3] K. Deb *et al.*, “Evolutionary algorithms for multi-criterion optimization in engineering design,” *Evolutionary algorithms in engineering and computer science*, vol. 2, pp. 135–161, 1999.

[4] C. A. C. Coello, *Evolutionary algorithms for solving multi-objective problems*. Springer, 2007.

[5] R. Gibbons, “An introduction to applicable game theory,” *Journal of Economic Perspectives*, vol. 11, no. 1, pp. 127–149, 1997.

[6] Y. C. Goh, X. Q. Cai, W. Theseira, G. Ko, and K. A. Khor, “Evaluating human versus machine learning performance in classifying research abstracts,” *Scientometrics*, vol. 125, pp. 1197–1212, 2020.

[7] J. M. Morrow and M. P. Sormani, “Machine learning outperforms human experts in mri pattern analysis of muscular dystrophies,” 2020.

[8] S. Booth *et al.*, “The perils of trial-and-error reward design: misdesign through overfitting and invalid task specifications,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, pp. 5920–5929, 2023.

[9] P. Mallozzi, R. Pardo, V. Duplessis, P. Pelliccione, and G. Schneider, “Movemo: a structured approach for engineering reward functions,” in *IEEE Intl. Conf. on Robotic Computing (IRC)*, pp. 250–257, IEEE, 2018.

[10] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: A survey,” *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[11] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, “Deep reinforcement learning: A brief survey,” *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[12] C. Molnar, G. Casalicchio, and B. Bischl, “Interpretable machine learning–a brief history, state-of-the-art and challenges,” in *Joint European conference on machine learning and knowledge discovery in databases*, pp. 417–431, Springer, 2020.

[13] G. T. Heineman and W. T. Councill, “Component-based software engineering,” *Addison-westley*, vol. 5, no. 1, 2001.

[14] P. DeGrandis and G. Valetto, “Elicitation and utilization of application-level utility functions,” 2009.

[15] P. McMinn, “Search-based software test data generation: a survey,” *Software testing, Verification and reliability*, vol. 14, no. 2, pp. 105–156, 2004.

[16] T. G. Kolda, R. M. Lewis, and V. Torczon, “Optimization by direct search: New perspectives on some classical and modern methods,” *SIAM review*, vol. 45, no. 3, pp. 385–482, 2003.

[17] D. Silver, S. Singh, D. Precup, and R. S. Sutton, “Reward is enough,” *Artificial intelligence*, vol. 299, p. 103535, 2021.

[18] openai, “Reinforcement fine-tuning.” https://platform.openai.com/docs/guides/reinforcement-fine-tuning, 2025.

[19] G. Hornby, A. Globus, D. Linden, and J. Lohn, “Automated antenna design with evolutionary algorithms,” in *Space 2006*, p. 7242, Aerospace Research Central, 2006.

[20] A. Gupta *et al.*, “Unpacking reward shaping: Understanding the benefits of reward engineering on sample complexity,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 15281–15295, 2022.

[21] A. K. Agogino and K. Tumer, “Unifying temporal and structural credit assignment problems,” in *Autonomous agents and multi-agent systems conference*, 2004.

[22] J. Leike, D. Krueger, T. Everitt, M. Martic, V. Maini, and S. Legg, “Scalable agent alignment via reward modeling: a research direction,” *arXiv preprint arXiv:1811.07871*, 2018.

[23] A. Van Lamsweerde, “Goal-oriented requirements engineering: A guided tour,” in *Proceedings fifth ieee international symposium on requirements engineering*, pp. 249–262, IEEE, 2001.

[24] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, “Rainbow: Architecture-based self-adaptation with reusable infrastructure,” *Computer*, vol. 37, no. 10, pp. 46–54, 2004.

[25] A. J. Ramirez, A. C. Jensen, B.H.C. Cheng, and D. B. Knoester, “Automatically exploring how uncertainty impacts behavior of dynamically adaptive systems,” in *2011 26th IEEE/ACM international conference on automated software engineering (ASE 2011)*, pp. 568–571, IEEE, 2011.

[26] E. M. Fredericks, B. DeVries, and B.H.C. Cheng, “Autorelax: automatically relaxing a goal model to address uncertainty,” *Empirical Software Engineering*, vol. 19, no. 5, pp. 1466–1501, 2014.

[27] M. A. Langford, K. H. Chan, J. E. Fleck, P. K. McKinley, and B.H.C. Cheng, “Modalas: addressing assurance for learning-enabled autonomous systems in the face of uncertainty,” *Software and Systems Modeling*, vol. 22, no. 5, pp. 1543–1563, 2023.

[28] M. A. Langford, S. Zilberman, and B.H.C. Cheng, “Anunnaki: A modular framework for developing trusted artificial intelligence,” *ACM Transactions on Autonomous and Adaptive Systems*, vol. 19, no. 3, pp. 1–34, 2024.

[29] R. S. Sutton, A. G. Barto, *et al.*, *Reinforcement learning: An introduction*, vol. 1. MIT press Cambridge, 1998.

[30] F. James, “Function minimization,” 1972.

[31] T. Back, *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

[32] J. H. Holland, “Genetic algorithms,” *Scientific american*, vol. 267, no. 1, pp. 66–73, 1992.

[33] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proceedings of ICNN’95-international conference on neural networks*, vol. 4, pp. 1942–1948, ieee, 1995.

[34] M. Dorigo, M. Birattari, and T. Stutzle, “Ant colony optimization,” *IEEE computational intelligence magazine*, vol. 1, no. 4, pp. 28–39, 2007.

[35] K. H. Chan, S. Zilberman, N. Polanco, J. E. Siegel, and B.H.C. Cheng, “SafeDriveRL: Combining Non-cooperative Game Theory with Reinforcement Learning to Explore and Mitigate Human-based Uncertainty for Autonomous Vehicles,” in *Proceedings of the 19th International Conference on Adaptive and Self-Managing Systems (SEAMS 2024)*, pp. 214–220, 2024. Short Paper.

[36] A. Wachi, “Failure-Scenario Maker for Rule-Based Agent using Multi-agent Adversarial Reinforcement Learning and its Application to Autonomous Driving,” May 2019.

[37] L. Weiwei, H. Wenxuan, J. Wei, L. Lanxin, G. Lingping, and L. Yong, “Learning to Model Diverse Driving Behaviors in Highly Interactive Autonomous Driving Scenarios with Multi-Agent Reinforcement Learning,” Feb. 2024.

[38] S. Roy, C. Ellis, S. Shiva, D. Dasgupta, V. Shandilya, and Q. Wu, “A survey of game theory as applied to network security,” in *2010 43rd Hawaii international conference on system sciences*, pp. 1–10, IEEE, 2010.

[39] A. Bhattacharya, S. D. Bopardikar, S. Chatterjee, and D. Vrabie, “Cyber threat screening using a queuing-based game-theoretic approach,” *Journal of Information Warfare*, vol. 18, no. 4, pp. 37–52, 2019.

[40] B. Marthi, “Automatic shaping and decomposition of reward functions,” in *Proceedings of the 24th International Conference on Machine learning*, pp. 601–608, 2007.

[41] F. Rietz, S. Magg, F. Heintz, T. Stoyanov, S. Wermter, and J. A. Stork, “Hierarchical goals contextualize local reward decomposition explanations,” *Neural Computing and Applications*, vol. 35, no. 23, pp. 16693–16704, 2023.

[42] Z. Juozapaitis, A. Koul, A. Fern, M. Erwig, and F. Doshi-Velez, “Explainable reinforcement learning via reward decomposition,” in *IJCAI/ECAI Workshop on explainable artificial intelligence*, 2019.

[43] J. Li *et al.*, “Goal-oriented knowledge reuse via curriculum evolution for reinforcement learning-based adaptation,” in *2022 29th Asia-Pacific Software Engineering Conference (APSEC)*, pp. 189–198, IEEE, 2022.

[44] W. Li, D. Qiao, B. Wang, X. Wang, B. Jin, and H. Zha, “Semantically aligned task decomposition in multi-agent reinforcement learning,” *arXiv preprint arXiv:2305.10865*, 2023.

[45] D. Bahdanau *et al.*, “Learning to understand goal specifications by modelling reward,” in *International Conference on Learning Representations 2019*, pp. 1–19, 2019.

[46] Y. Hu, W. Wang, H. Jia, Y. Wang, Y. Chen, J. Hao, F. Wu, and C. Fan, “Learning to utilize shaping rewards: A new approach of reward shaping,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 15931–15941, 2020.

[47] Y. Yang *et al.*, “A systematic study of reward for reinforcement learning based continuous integration testing,” *Journal of Systems and Software*, vol. 170, p. 110787, 2020.

[48] H. Cuayáhuitl, S. Yu, A. Williamson, and J. Carse, “Scaling up deep reinforcement learning for multi-domain dialogue systems,” in *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 3339–3346, IEEE, 2017.

[49] M. B. Duran and G. Mussbacher, “Investigation of feature run-time conflicts on goal model-based reuse,” *Information Systems Frontiers*, vol. 18, pp. 855–875, 2016.

[50] C. Eteke *et al.*, “Reward learning from very few demonstrations,” *IEEE Trans. on Robotics*, vol. 37, no. 3, pp. 893–904, 2020.