Coordinated Self-Exploration for Self-Adaptive Systems in Contested Environments

Saad Sajid Hashmi[1], Hoa Khanh Dam[1], Alan Colman[2], Anton V. Uzunov[3], Quoc Bao Vo[2], Mohan Baruwal Chhetri[4] and James Dorevski[1]

[1]*University of Wollongong, Wollongong, Australia*

[2]*Swinburne University of Technology, Melbourne, Australia*

[3]*Defence Science and Technology Group, Adelaide, Australia*

[4]*CSIRO's Data61, Melbourne, Australia*

{*shashmi, hoa, jdorevski*}*@uow.edu.au*, {*acolman, bvo*}*@swin.edu.au, anton.uzunov@defence.gov.au,*

Keywords: Incompatible States, Goal Dependency, Adversarial Search, Multi-Agent Systems.

Abstract: Enhancing the resilience and flexibility of distributed software systems is critical in challenging environments where adversaries can actively undermine performance and system operations. One approach for achieving resilience is to employ collections of intelligent software agents that can autonomously execute management actions and adapt a target system according to pre-defined goals, thereby realising various self-* properties. Self-exploration is one such self-* property, relating to a system's ability to compute resilient responses through an adversarial game-tree search process that takes into account an adversary's action-effects on goals. Unlike the current realisation of self-exploration that assumes goal independence, we propose a novel approach that addresses goal inter-dependencies through agent coordination, ensuring more realistic and effective counter-responses. We provide a correctness proof and evaluate the performance of our algorithm.

1 INTRODUCTION

Cyber attacks are becoming increasingly sophisticated in recent years, causing significant damage to businesses, organisations and critical infrastructures worldwide (IC3, 2023). To address this challenge, there is an urgent need for new methods and techniques to develop systems that are increasingly resilient to external attacks and disruptions. Self-adaptation and self-management are two essential properties for enhancing the resilience and flexibility of distributed software systems in complex, contested environments, where an adversary can actively reduce performance and impede general system operation (Atighetchi and Pal, 2009; Baruwal Chhetri et al., 2019; Linkov and Kott, 2019). These properties imply that a system is able to continuously monitor its environment and adjust its structure, behaviour, and configuration in response to changing conditions.

Self-adaptation and self-management can be realised in various ways. One way is to design a system as a collection of intelligent (AI-powered) software agents that autonomously manage and adapt a target (henceforth *domain*) system to maintain and enhance performance (Tesauro et al., 2004; Weyns and Georgeff, 2009; Florio, 2015; Baruwal Chhetri et al., 2018; Uzunov et al., 2023). Compared to a central agent managing the system, a multi-agent system enhances robustness and fault tolerance by reducing the risk of a single point of failure. In a multi-agent system, each agent is responsible for managing different parts or components of the domain system, aiming to satisfy specific system requirements or goals. In scenarios where goals are dependent, some goal state configurations cannot coexist. Therefore, agents must communicate and coordinate to achieve their goals while avoiding conflicting configurations (Ismail et al., 2018; Ponniah and Dantsker, 2022).

To be resilient in contested environments, the aforementioned AI-powered agents must continually reason ahead, consider adversaries' actions and their effects, and evaluate and deploy countermeasures to achieve their system goals. Recent work (Hashmi et al., 2022) has proposed a new property for self-adaptive systems called *self-exploration* to describe this capability. Inspired by the breakthrough perfor-

Hashmi, S. S., Dam, H. K., Colman, A., Uzunov, A. V., Vo, Q. B., Baruwal Chhetri, M. and Dorevski, J.
Coordinated Self-Exploration for Self-Adaptive Systems in Contested Environments.
DOI: 10.5220/0013140900003890
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 17th International Conference on Agents and Artificial Intelligence (ICAART 2025) - Volume 1*, pages 318-332
ISBN: 978-989-758-737-5; ISSN: 2184-433X
Proceedings Copyright © 2025 by SCITEPRESS – Science and Technology Publications, Lda.

mance of DeepMind's AlphaGo (Silver et al., 2016), self-exploration empowers agents to leverage adversarial search to anticipate the potential effects of adversarial attacks on the system and compute counter-responses, ensuring system resilience. Each agent independently explores the possibility of achieving or maintaining their assigned goal(s) in the face of adversarial attacks.

However, the work by (Hashmi et al., 2022) has a major limitation: it assumes independence between system goals. This oversimplification does not reflect real-world scenarios, where system goals are often dependent. For example, providing accurate predictions depends on the quality of the processed data. Specifically, if the data is significantly compressed and degraded, high accuracy in predictions cannot be expected. The existing self-exploration approach fails to account for such dependencies, potentially resulting in responses or actions that agents cannot collectively perform to achieve their system goals.

In this paper, we aim to address this gap by considering the dependencies between system goals. We introduce a new concept called *incompatible goal state* into the self-exploration process to represent scenarios where achieving one goal with certain quality attribute values makes it impossible to achieve another goal with contradictory quality attribute values. In this paper, we focus on pairwise goal dependencies, as it allows us to isolate the impact of one goal state on another, enabling us to make precise adjustments. In pairwise goal dependencies, all the incompatible states in a system are broken down into incompatible states between pairs of goals. We propose a novel, coordinated approach to self-exploration in which agents account for these incompatible goal states during their self-exploration process. This ensures that the responses agents compute are feasible for achieving system goals, even in the face of adversarial attacks. Our approach to resolving incompatibilities in goal pairs is a step towards a general solution involving more than two goals in an incompatibility relationship. We have developed a prototype implementation of our approach using Python, Docker[1] and ZeroMQ[2]. We formally prove the correctness of our approach, and also demonstrate its efficient performance through a series of experiments.

The rest of the paper is organised as follows. In Section 2, we present a motivating example for the research problem. Section 3 provides an overview of self-exploration concepts. In Section 4, we present a rigorous problem formulation. Section 5 illustrates the methodology underlying our proposed approach.

[1]https://docker.com

[2]https://zeromq.org

In Section 6, we summarise our distributed implementation, and in Section 7, we evaluate the performance of our proposed approach. Finally, in Section 8, we discuss the related work, and in Section 9, we offer some closing remarks.

2 MOTIVATING EXAMPLE

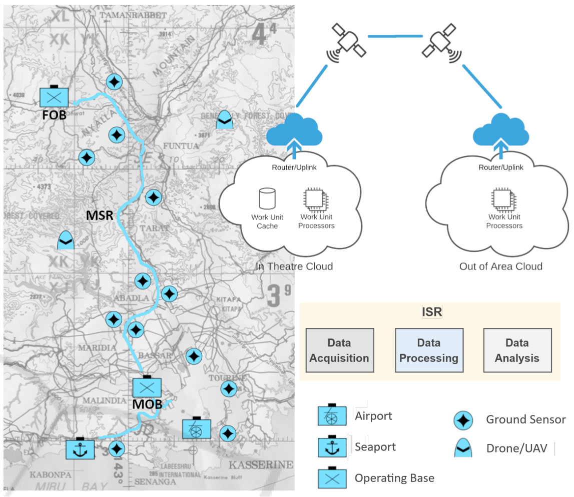


Figure 1: Overview of the `ISR` scenario along the main supply route (MSR) connecting the seaport and airport with the main and forward operating bases (MOB and FOB)[3]. Surveillance data is collected by ground sensors and UAVs. This data is subsequently transmitted to cloud-based services, where it undergoes further processing and analysis to extract valuable insights.

In this section, we present a running scenario of a cyber-physical system used for Intelligence, Surveillance and Reconnaissance (`ISR`) activities along supply routes (*see* Figure 1). The system provides services such as data acquisition, data processing, and data analysis, each managed by separate agents. The goals of the agents are to maintain the provisioning of these services. Various sensors, including ground sensors and surveillance drones, collect raw data along these routes (**Data Acquisition**, `DAQ`). This raw data undergoes various operations including (i) filtering to remove irrelevant data or noise, such as glitches and sensor malfunctions; (ii) aggregation to combine similar data points, summarising large amounts of information; and (iii) transformation to reformat the data into a consistent and structured format, making it easier to analyse trends and patterns (**Data Processing**, `DP`). The processed data is then analysed using various statistical methods and machine learning algorithms to identify anomalies, predict future events, and pro-

[3]Adapted from (Hashmi et al., 2023).

vide real-time situational awareness (**Data Analysis**, DA). This analysis informs decision-making by highlighting potential risks, opportunities, and areas requiring further investigation.

Each of these services—data acquisition, data processing, and data analysis—has critical attributes that influence their successful achievement. For example, the data acquisition service might be influenced by coverage area and video resolution. The data processing service has attributes such as compression ratio and processing time. Additionally, the data analysis service might be influenced by (classification) accuracy and F1 score of the chosen algorithm(s) and available computational resources.

In a contested environment, adversaries may disrupt ISR services and affect their functionality and quality through actions such as network flooding, false data injection and jamming. In the event that an adverse incident results in the degradation of a service attribute, the managing agent has the capability to execute actions that alter system attributes, thereby ameliorating the affected service attribute. For instance, to enhance the video resolution of DAQ service, an agent can allocate additional bandwidth, tweak sensor range, or utilise a spare battery with higher battery life. By reasoning ahead, an agent can compute the most optimal action to take in a given circumstance.

While achieving these three services (or goals) is crucial, their objectives can sometimes become incompatible. This inherent conflict is further amplified when each service is managed by separate agents, creating challenges in ensuring the overall effectiveness of the system. For example, a conflict may arise in the following scenarios:

- **Data Acquisition vs. Data Processing.** In data acquisition, capturing a large volume of data from various sources ensures comprehensive coverage of the operational area. However, this vast amount of data can overwhelm processing capabilities, leading to delays in handling critical events or anomalies that require a real-time response. This highlights the tradeoff between data comprehensiveness and processing speed.
- **Data Processing vs. Data Analysis.** Data processing may involve applying data compression techniques to reduce data size and complexity, enabling efficient storage and transmission. However, compression can introduce information loss, potentially masking subtle features crucial for in-depth analysis. Conversely, data analysis might require high-fidelity, uncompressed data to extract these fine-grained details.

These examples illustrate the inherent challenges in managing a cyber-physical ISR system and the need for an agent to reason ahead to compute the best response in managing their service while avoiding incompatibility with another managed service. Balancing the needs of data acquisition, processing, and analysis requires careful consideration of their often-conflicting goals. Finding strategies to manage these incompatibilities is crucial for optimising the overall effectiveness of the ISR system.

3 CONCEPTS IN SELF-EXPLORATION

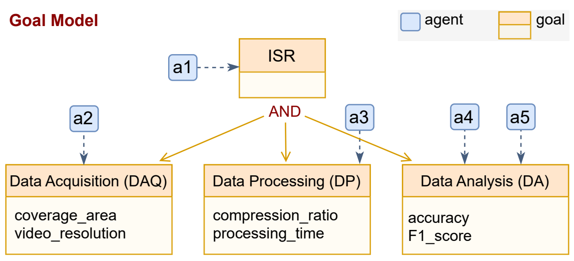
Self-exploration is a self-* property that enables agents in a multi-agent system to compute optimal, resilient responses to adverse events through adversarial search (lookahead with adversaries, as an extensive-form game). Each agent uses self-exploration to identify the best action to take in the current situation to achieve its goals, considering its capabilities, those of its teammates, and the actions of adversaries. Some actions may be infeasible, as they can transition the system into an incompatible state (discussed in Section IV). The focus of this paper is on collaboration among agent-teammates such that they avoid reaching an incompatible state during self-exploration. In this subsection, we provide a somewhat more rigorous (but not formal) definition of the key concepts used in self-exploration.

3.1 Goals and Goal Models

Central to self-exploration are the concepts of goals and goal models, which are defined as follows.

- **Goal.** Declarative representation of a system requirement that must be satisfied (achieved or maintained) (Franch et al., 2016). A goal can either be a leaf goal (e.g., DAQ goal in Figure 2) or a non-leaf goal[4] (e.g., ISR goal in Figure 2).
  - **Leaf Goal.** Associated with a set of attributes and conditions.
  - **Non-leaf Goal.** Decomposed into one or more sub-goals (e.g., ISR goal is decomposed into DAQ, DP, and DA subgoals).
- **Goal Attribute.** Measurable or observable property of the system that contributes to goal fulfilment (e.g., accuracy in DA goal).

[4]While we do not adhere to a particular goal modelling framework (such as *i*∗, KAOS), we follow the generally accepted design of goal models as tree-like structures, where higher-level goals are decomposed into lower-level goals using AND/OR relationships (cf. (Uzunov et al., 2021)).



goal satisfaction conditions	
ISR	DAQ and DP and DA
DAQ	DAQ.coverage_area > MEDIUM and DAQ.video_resolution > MEDIUM_HIGH
DP	DP.compression_ratio > MEDIUM and DP.processing_time < MEDIUM_LOW
DA	DA.accuracy > MEDIUM_HIGH and DA.F1_score > MEDIUM_HIGH

goal failure conditions	
DAQ	DAQ.coverage_area < MEDIUM_LOW and DAQ.video_resolution < MEDIUM_LOW
DP	DP.compression_ratio < MEDIUM_LOW and DP.processing_time > MEDIUM_HIGH
DA	DA.accuracy < MEDIUM and DA.F1_score < MEDIUM_LOW

incompatible goal states			
α / β	goals	incompatibility conditions	agents
α	DP	DP.processing_time = VERY_LOW	a3
β	DAQ	DAQ.coverage_area = VERY_HIGH and DAQ.video_resolution = VERY_HIGH	a2
α	DA	DA.accuracy = VERY_HIGH	a4, a5
β	DP	DP.compression_ratio = VERY_HIGH	a3

Figure 2: Goal Model of the `ISR` scenario.

- **Goal Condition.** Establishes criteria for the value of a goal attribute that must be met for the goal to be considered satisfied or failed (e.g., if both `DA.accuracy` and `DA.F1_score` is greater than `MEDIUM_HIGH`, then the `DA` goal will be satisfied).
- **Subgoal.** A more specific goal that contributes to the satisfaction (or failure) of the given goal (e.g., `ISR` is satisfied if the subgoals `DAQ`, `DP`, and `DA` are satisfied).
- **Goal State.** Defined by the values of the goal attributes at a given time. The goal state indicates the satisfaction status of a goal (e.g., if at a given timestamp, both `DA.accuracy` and `DA.F1_score` are `VERY_HIGH`, then `DA` is deemed to be satisfied in that goal state).
- **Goal Model.** Structured representation of the requirements in a system comprising goals, goal attributes, subgoals, goal conditions, and agents (Van Lamsweerde, 2001).
- **Root Goal.** The highest-level goal within a goal model (e.g., `ISR` goal in Figure 2).

Figure 2 illustrates the goal model for the `ISR` scenario from the motivating example (*see* Section 2), showing the goal decomposition of the `ISR` goal into `DAQ`, `DP`, and `DA` subgoals, along with their respective satisfaction and failure conditions. Additionally, it identifies the agents that can manage each goal based on their specific capabilities. Incompatibility conditions, indicated at the bottom of Figure 2, are discussed in Section 4.2.

3.2 Self-Exploration and Game Trees

Self-exploration consists of two parts: (i) global and (ii) local. **Global self-exploration** manages interactions between agents and is initiated by the **root agent**, i.e., the agent handling the root goal (e.g., $a1$ in Figure 2). **Local self-exploration** identifies the best action(s) for a given agent at the current state of a leaf goal. It combines the precision of a tree search with the power of statistical sampling to evaluate the potential outcomes of different actions.

To satisfy the root goal, its subgoals must be satisfied. A leaf goal can be managed by one or more agents (e.g., the `DA` goal can be managed by either $a4$ or $a5$). These agents are called the **candidate agents** for that goal. When an agent comes across a subgoal that it cannot handle (e.g., due to lacking required capabilities), it sends a delegation request for the subgoal to its candidate agents (e.g., $a1$ sends a delegation request for `DAQ` to $a2$). The agent sending the delegation request (e.g., $a1$) is called a delegator, and the agent receiving the request (e.g., $a2$) is called a delegatee. Once a candidate agent receives a delegation request for a leaf goal, it performs local self-exploration. It then returns the output (covered in the next sub-section) to the delegator agent. The delegator agent receives the output from all the candidate agents and identifies the most suitable/capable agent for a given goal to establish a contract. A **contract** is a relationship between two agents that specifies the responsibility of one agent towards the other in achieving/maintaining a goal (e.g., $a1$ establishes a contract with $a2$ for `DAQ`).

As indicated in the previous sub-section, agents are responsible for achieving or maintaining one or more goals, which are adopted at runtime. During self-exploration, these agents construct individual, goal-based **game trees**. These trees are graphs that project each agent's future goal states, resulting from their own planned actions or those anticipated from adversaries Specifically, a **node** in a game tree represents the collective state of all goals of an agent (e.g., a node in the game tree, generated by $a2$, represents the state of `DAQ`). Therefore, a node is defined by the values of an agent's goal attributes. If an agent is responsible for only one goal, then the node is equivalent to a goal state. As the number of nodes increases exponentially with each depth of the game tree, the unlikely branches need to be pruned to improve the efficiency of the adversarial search (Hashmi et al., 2025). If an agent is responsible for only one goal, then the node is equivalent to a goal state. An **edge** in a game tree represents a goal state transition, occurring sequentially through an agent's action in one round followed by an adversary's action in the next round. The action space of an agent represents the combination of all the actions that an agent can perform. An **effect** is a node (or goal state) that is reached as an outcome of an agent or adversary's action (e.g., $a2$'s action that increases the video resolution to a specific value is an edge, and the node representing the increased video resolution value is an effect of that action). The **effect space** of a given node is the set of all possible nodes that can be reached from that node.

3.3 Key Elements of Local Self-Exploration

Local self-exploration is based on Monte Carlo Tree Search (MCTS) (Kocsis and Szepesvári, 2006), which consists of four phases:

- **Selection.** The tree is traversed from the root node (representing the current state of the system) to a leaf node (e.g., $a2$ traverses the tree from the current state of `DAQ` to the leaf node of the explored tree). All nodes except the root represent future states.
- **Expansion.** The tree is expanded by a single depth at the leaf node via generation of successor (or child) nodes (e.g., $a2$ expands the tree by generating all possible states that can be reached by $a2$ through its action space at the leaf node).
- **Simulation (or Rollout).** A simulation or rollout is performed from the leaf node to a **terminal node** i.e., a node where an agent wins, loses, or a cut-off depth is reached. Simulations are performed to evaluate potential future states by estimating the outcomes (wins, losses) from the leaf node. A win occurs at the node where the goal is satisfied (e.g., if `DAQ.coverage_area` > `MEDIUM` and `DAQ.video_resolution` > `MEDIUM_HIGH`, then `DAQ` is satisfied at that state and it is considered a win). A loss occurs when the failure condition of the goal is true. The cut-off depth d is reached if by depth d (i) the goal remains unsatisfied and (ii) no failure conditions are met.
- **Backpropagation.** The evaluation of the terminal node (along with the visit count of the node) is backpropagated to the root node to reflect the new information gained from the simulation. By aggregating evaluations from multiple simulations, the tree builds a more accurate statistical representation of the expected outcomes from different states and actions. This guides the selection phase in future iterations by directing the search towards the most promising nodes (e.g., future iterations by $a2$ leads the search towards states where `DAQ` is satisfied).

These four phases are repeated for each new simulation, building a more accurate statistical representation of expected outcomes and guiding the search towards the most promising nodes.

In local self-exploration, an agent operates with a specific reasoning budget (e.g., a fixed number of simulations or simulation time). Once the reasoning budget is exhausted, the node at depth 1 with the highest visit count is determined as the (**transitioning state**), indicating the most suitable node to transition to. The edge leading to this node represents the agent's most optimal action. Additionally, the agent's root node accumulates the sum of evaluations for all the simulations. This sum, divided by the total number of simulations, yields the agent's **win ratio**. The win ratio serves as a measure of the agent's utility in adverse conditions and is returned to the delegator agent upon completion of local self-exploration. In scenarios where multiple agents are capable of handling a goal, the win ratio acts as the determining factor for the selection of the most capable agent.

4 PROBLEM FORMULATION

Previous work on self-exploration (Hashmi et al., 2022; Hashmi et al., 2023), simplifies the problem by assuming that each goal operates independently from other goals. However, this assumption does not reflect real-world scenarios where goals can be dependent. Understanding these dependencies is crucial as they can significantly impact how agents perform self-exploration. In this work, we examine how self-exploration can be performed when goals have dependencies. Dependencies between goals can manifest in various forms, such as temporal dependencies where one goal must be satisfied before another, or resource dependencies, where agents assigned to different goals share a common resource. In this paper, we specifically focus on incompatible goal states as an illustrative example of goal dependencies. In this context, the ability to reach a particular goal state (e.g., `DP.compression_ratio = VERY_HIGH` and `DP.processing_time = LOW`) is dependent on the state of another goal (e.g., `DA`). In our scenario, these dependencies between goal states, and potential incompatibilities, arise because there are input/output performance dependencies between agents in a data processing pipeline. Our work, however, can be extended to other types of goal dependencies too.

4.1 Incompatible Goal States

As discussed previously, a goal state is defined by specific attribute values. Incompatible goal states between two goals are states that cannot co-exist within the system. They represent situations where achieving one goal with its particular attribute values makes it impossible to achieve another goal with specific attribute values, simultaneously. For example, a goal state of *Data_Processing* with a `VERY_HIGH` *compression_ratio* is incompatible with a goal state of *Data_Analysis* with a `VERY_HIGH` *accuracy*, as high compression ratio lowers the data fidelity, thereby degrading the classification accuracy of objects in the compressed sensor data. Similarly, a goal state of *Data_Acquisition* with a `VERY_HIGH` *video_resolution* and a `VERY_HIGH` *coverage_area* is incompatible with a goal state of *Data_Processing* with a `VERY_LOW` *processing_time*. In the next sub-section, we discuss the classification of goals, which helps determine priority among goal states and resolves incompatibilities.

4.2 Goal Classification

To understand how goal dependencies manifest as incompatible states, we define three categories of goals based on their involvement in incompatible states. In a pair of incompatible goal states, the goal with priority is called an **α** goal, while the non-priority goal is called the **β** goal. The incompatibility conditions associated with an α goal and a β goal are called the α condition and the β condition, respectively. An α goal can transition to any state regardless of whether the α condition is true or false in the transitioning state. In contrast, there are restrictions on the transitioning state for a β goal. A β goal can transition to a state where the (i) β condition is false, or the (ii) β condition is true but the α condition for an α goal's transitioning state is false.

Furthermore, in the collection of all the pairs of incompatible goal states, a goal can either be $\mathcal{A}$, $\mathcal{B}$, or $\mathcal{A}\_\mathcal{B}$.

- A goal is classified as $\mathcal{A}$ if it appears as an α goal in one or more pairs of incompatible goal states and none as a β goal.
- A goal is classified as $\mathcal{B}$ if it appears as a β goal in one or more pairs of incompatible goal states and none as an α goal.
- A goal is classified as $\mathcal{A}\_\mathcal{B}$ if it appears as an α goal in one or more pairs of incompatible goal states and as a β goal in one or more pairs as well.

In Figure 2, `DA`, `DAQ`, and `DP` serve as examples of $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{A}\_\mathcal{B}$ goals, respectively. A pair of states are deemed incompatible if and only if the α condition is true in one state and the corresponding β condition is true in the other state.

4.3 Agent Awareness of Incompatible Goal States

We assume that the agents have partial observability of the system they are managing. In our approach, each delegator agent has knowledge of the incompatible goal state for the immediate sub-goals. This implies that the delegator understands how the goals it delegates might conflict. For example, in Figure 3, $a1$ knows the incompatible goal states for sub-goals $g2$ and $g3$, while $a6$ knows those for $g9$, $g10$ and $g11$. Because an agent can have multiple candidate agents, and each agent can have a unique best-transitioning state, this implies that some of the candidate agents for an α goal can have a best-transitioning state where the α condition is true, whereas other candidate agents' best-transitioning state can have α condition false. If the candidate agents for the α goals directly communicate with the candidate agents for the β goals, the latter can receive conflicting communications regarding whether the α condition is true or not. Therefore, in our approach, the communication only happens between the delegator and the delegatee agents, and the best-transitioning state for each agent is identified by the delegator after it has identified the best candidate agent for each subgoal based on the win ratios received from the candidate agents. In Fig-

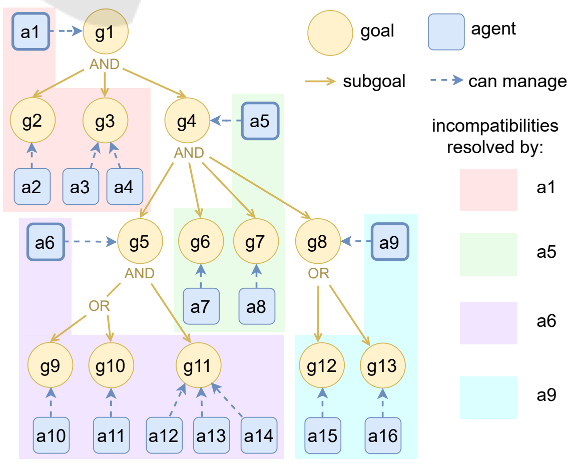


Figure 3: An illustrative goal model of depth 3, with shaded regions representing the knowledge of the delegator agents (bold outline) and their communication with the delegatee agents to resolve incompatible goal states.

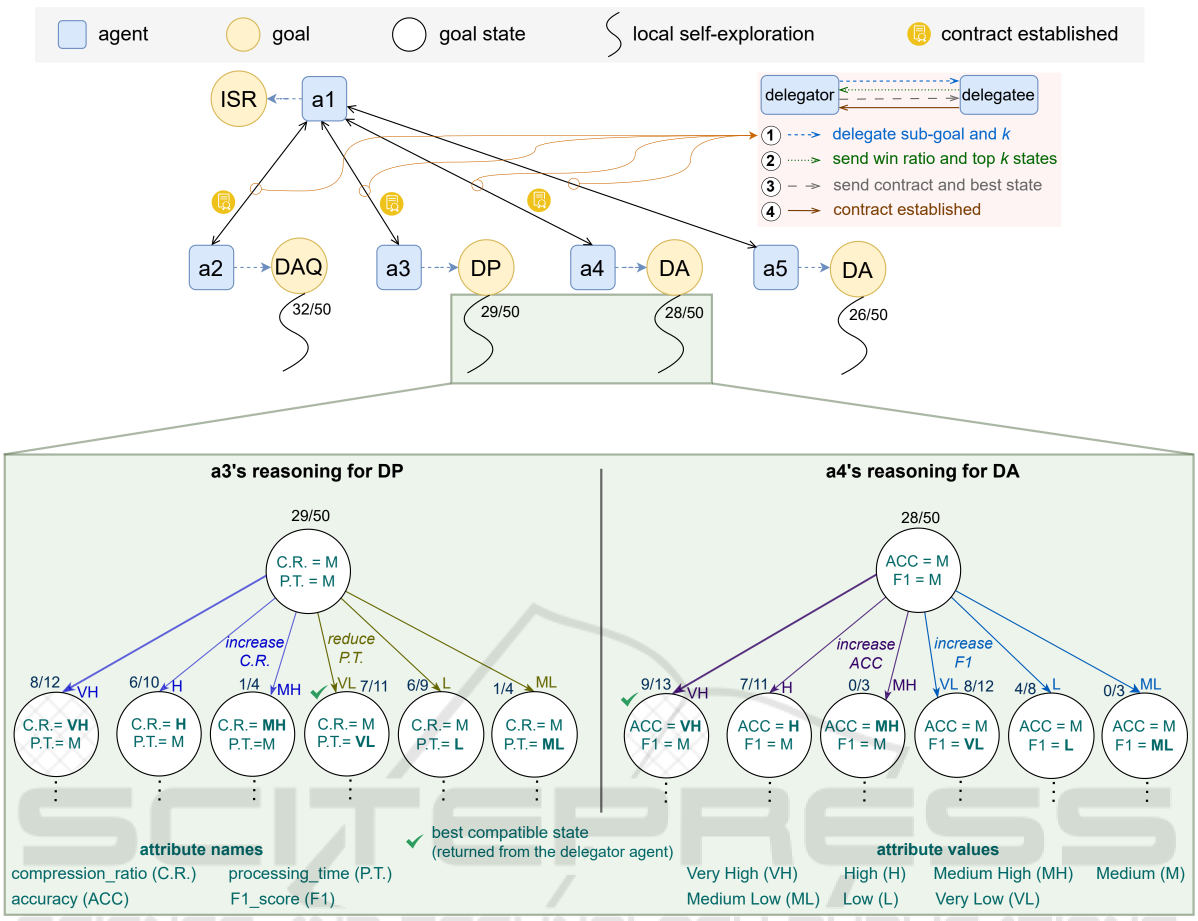


Figure 4: Overview of coordinated self-exploration w.r.t. `ISR` scenario. The four communication exchanges between the delegator and the delegatee agents are shown in the top-right corner. Agent $a1$ is the delegator, while the rest of the agents are delegatees. The local self-exploration of agents $a4$ and $a3$ is enlarged at the bottom of the figure. For simplicity, only depth 1 of the game trees are shown. The fractions written on top of the goal states represent the win ratios of simulations that pass through those states. The most visited branch is highlighted in bold. Cross-hatched states indicate incompatible states. In scenarios where the most visited state (in the simulations) is incompatible, the next most visited state is chosen as the best compatible state by the delegator agent.

ure 3, $a7$ and $a8$ communicate solely with $a5$, which identifies their best-transitioning states. We further assume that the actions of the agents are deterministic, i.e., given the current goal state, an agent can determine what the resultant goal state will be after performing a specific action.

5 COORDINATED SELF-EXPLORATION

In global self-exploration, a delegator agent delegates a subgoal to all the candidate agents for that subgoal. If the subgoal is a leaf goal, the candidate agents perform local self-exploration on it to obtain the most optimal action. Unlike scenarios where goals are independent and a candidate agent only reports the win ratio after simulations, here a candidate agent also coordinates with the delegator agent by sending the win ratio and the top-k ranked goal states based on the number of visits during simulations. The delegator agent then *(i)* selects the best candidate agent for each subgoal based on the win ratios, and *(ii)* selects the best state for each of the chosen candidate agents. For *(ii)*, because the delegator agent has knowledge of the incompatible goal states, it selects the highest ranked goal state (from the top-k) that does not have any incompatibility with the selected goal states of the other goals. The delegator agent then establishes a contract with the best candidate agent for each subgoal and returns the best (among k) goal states back to the contracted agents. The best goal states are those that have the highest utility and no incompatibilities.

We illustrate our approach in Figure 4. There are four communication exchanges between a delegator and a delegatee agent. First, the delegator agent delegates the subgoal and k (number of top states to consider) to the delegatee agent. Second, the delegatee agent returns the win ratio from local self-exploration and the top-k states to the delegator agent. Third, the delegator agent sends the contract and the best compatible state to the chosen delegatee agent for each subgoal. Finally, the delegatee agent acknowledges the receipt of contract and best state, and the contract between the delegator and delegatee agent is established. In Figure 4, agents $a2$, $a3$, $a4$, and $a5$ perform local self-exploration on the subgoals of `ISR` goal. For `DA` goal, $a1$ establishes contract with $a4$ instead of $a5$ due to the higher win ratio. For $a3$, the top ranked state is an incompatible state, because `DP` is the β goal and `DA` is an α goal (*see* Figure 2), and the α condition is true in the top ranked state of `DA`. Hence, $a1$ selects the next ranked state (based on the visit count) for `DP` where `DP.processing_time` is reduced to `VERY_LOW`.

Furthermore, to enhance scalability, our approach is based on parallel reasoning, i.e., candidate agents for multiple goals can perform self-exploration simultaneously. This implies that there is an overlap in the self-exploration timestamps between candidate agents of various goals, allowing concurrent reasoning processes.

5.1 Coordinated Self-Exploration Algorithm

We develop our algorithm based on the following assumptions:

- In our simulation of `DAQ`, `DP`, and `DA` services within the `ISR` scenario, we assume that sensors and all the potential connections, including those between sensors and the `DAQ` system, the `DAQ` system and `DP` units, `DP` units and `DA` systems, and `DA` systems and user interfaces, are operating without faults or malfunctions to mitigate external validity threats.
- In coordinated self-exploration, an agent handling a goal G communicates only with agents capable of handling the subgoals of G (delegating subgoals of G to the candidate agents) or the parent goal of G (sending win ratio of G). We assume that the incompatibility exists only between goals that are at the same depth and share the parent goal, i.e., both goals are immediate subgoals of the same parent goal.
- A circular dependency exists if a goal has no $\mathcal{A}$ and $\mathcal{B}$ subgoals, and all the subgoals are $\mathcal{A}$_$\mathcal{B}$.

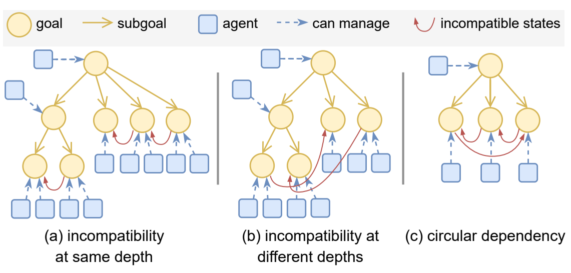

goal
subgoal
agent
can manage
incompatible states
(a) incompatibility at same depth
(b) incompatibility at different depths
(c) circular dependency

Figure 5: (a) Incompatibility between goals that are at the same depth. (b) Incompatibility between goals that are at different depths. (c) Circular dependency between goals implying that there are only $\mathcal{A}$_$\mathcal{B}$ (sub) goals. In the incompatible states symbol, the arrow head points to the β goal.

This implies that each subgoal is an α goal for one pair of incompatible states and a β goal for another pair of incompatible states. Consequently, the selection of the best-transitioning state for each subgoal depends on the selection of the best-transitioning state for another subgoal. Therefore, we assume that there is no circular dependency in the goal model.

In a circular dependency, each goal's involvement in multiple incompatibility relationships means that changing its best state can affect the compatibility of other relationships. In Figure 5, only the goal model in (a) aligns with our assumptions, while models like (b) and (c) are excluded from our approach.

The *agent* handling the root *goal* initiates global self-exploration with the knowledge of incompatible goal states for the subgoals and the number of top transitioning states (k) to request from each delegatee agent (line 1, Algorithm 1). The *results* variable will store a list of tuples containing the output of each local self-exploration for the immediate subgoals of *goal* (line 2). The *best_agents* variable will store a list of tuples containing the win ratios for the best agents for each (immediate) subgoal. A safety check (lines 4-6) is triggered if an agent encounters a goal and is not among the candidate agents. In lines 7-9, *agent* performs local self-exploration and sends the output to the delegator agent, where the delegator agent populates its results variable. In lines 10-17, *agent* iterates through each *subgoal* of *goal* and delegates the *subgoal* to each candidate agent of *subgoal*. The output of self-exploration is then appended to the *results* in the delegator agent. By line 18, the delegator agent has received the win ratios from all the delegatee agents, and the delegator agent therefore starts identifying the best candidate agent for each subgoal based on the highest win ratios (lines 18-20). The best (compatible) states for each subgoal are identified via the function call of *ident_states*() in line 21. This function is elaborated in Algorithm 2. Finally, Algorithm 1: Coordinated Self-Exploration.

```
1   Procedure self_exp(goal, agent, k)
2       $results \leftarrow$ list of tuples   // format: (goal, agent, win-ratio, [k states])
3       $best\_agents \leftarrow$ list of tuples   // format: (goal, agent, win-ratio)
4       if $agent \notin get\_candidate\_agents(goal)$ then
5           return $\varnothing, \varnothing$
6       end
7       if $goal.subgoals = \varnothing$ then
8           $winRatio, topK \leftarrow self\_exp\_local(goal, agent, k)$
9           $results.append(goal, agent, winRatio, topK)$
10      else
            /* goal is a non-leaf goal */
11          foreach $subgoal \in goal.subgoals$ do
12              foreach $cand\_agent \in get\_candidate\_agents(subgoal)$ do
                    /* delegating subgoal to candidate agent */
13                  $winRatio, topK \leftarrow self\_exp(subgoal, cand\_agent, k)$
14                  $results.append(subgoal, cand\_agent, winRatio, topK)$
15              end
16          end
17      end
        /* identifying best agents for each subgoal */
18      foreach unique $subgoal \in$ 1st col. of results do
19          $best\_agents.append(subgoal, cand\_agent, win\_ratio)$
              where cand_agent has highest win_ratio for subgoal
20      end
        /* identifying best compatible transitioning state for each agent in best_agents */
21      $best\_states \leftarrow$
          $ident\_states(agent, results, best\_agents, goal.incomp\_states, k)$
        /* root agent initiates establishing contracts with each agent in best_agents */
22      if agent is root agent then
23          $establish\_contracts(agent, best\_states)$
24      end
```

the root agent initiates the process of sending contracts and the best state to the best agent for each subgoal (lines 22-24). The value of the *best_states* variable is obtained from Algorithm 2, and it contains the required information. The process of establishing contracts continues in a top-down fashion until the contracts (and the best-transitioning state) for the leaf goals have been sent.

The *ident_states*() is passed the *results*, *best_agents*, and k from Algorithm 1, along with the incompatibility states (line 1, Algorithm 2). The data structure of *incomp_states* is a list of dictionaries where each dictionary has four keys, namely, α goal, α condition, β goal, and β condition (*see* Figure 2). The *best_states* variable will store a dictionary of keys and values, where each key corresponds to a delegatee agent and value corresponds to the best-transitioning state for that delegatee agent (line 2). The *get_agents*() function on line 4 returns the agents that are candidate agents for the *subgoal* and where the *subgoal* is an α goal in the incompatibility states. In lines 6-8, best states are identified for the subgoals that do not have incompatibility with any goal. The $get\_state(results, subgoal, agent, k, 1)$ function on line 7 returns a state from *results* where

Algorithm 2: Identify the best-transitioning state.

```
1   Procedure ident_states(agent, results, best_agents,
      incomp_states, k)
2       $best\_states \leftarrow$ dictionary with keys and values   // agent:state
3       foreach $subgoal, agent$ in 1st, 2nd cols. of best_agents do
4           $\alpha\_agents \leftarrow get\_agents(\alpha, subgoal, incomp\_states)$
5           $\beta\_agents \leftarrow get\_agents(\beta, subgoal, incomp\_states)$
            // selection of states for leaf goals other than $\mathcal{A}$ and $\mathcal{B}$
6           if $\alpha\_agents = \varnothing$ and $\beta\_agents = \varnothing$ and
              $subgoal.subgoals = \varnothing$ then
                /* get highest ranking (1) state from results */
7               $best\_states[agent] \leftarrow$
                  $get\_state(results, subgoal, agent, k, 1)$
8           end
            /* selection of states for $\mathcal{A}$ goals */
9           if $agent \in \alpha\_agents$ and $agent \notin \beta\_agents$ then
10              $best\_states[agent] \leftarrow$
                  $get\_state(results, subgoal, agent, k, 1)$
11          end
12      end
        /* selection of states for $\mathcal{A}$_$\mathcal{B}$ goals */
13      foreach $subgoal, agent$ in 1st, 2nd cols. of best_agents do
14          $\alpha\_agents \leftarrow get\_agents(\alpha, subgoal, incomp\_states)$
15          $\beta\_agents \leftarrow get\_agents(\beta, subgoal, incomp\_states)$
16          if $agent \in \alpha\_agents$ and $agent \in \beta\_agents$ then
17              $other\_\alpha\_agent \leftarrow \varnothing$
18              foreach $\alpha\_agent \in get\_\alpha\_agents(subgoal)$ do
19                  if $\alpha\_agent \in$ 2nd col. of best_agents then
20                      $other\_\alpha\_agent \leftarrow \alpha\_agent$
21                  end
22              end
23              $\alpha\_goal \leftarrow get\_\alpha\_goal(subgoal)$
24              $\alpha\_cond \leftarrow get\_condition(\alpha, \alpha\_goal, subgoal)$
25              $\beta\_cond \leftarrow get\_condition(\beta, \alpha\_goal, subgoal)$
26              $top\_rank \leftarrow 1$   // top rank without incompatibility
                /* condition for incompatibility */
27              while $top\_rank \leq k$ and $\beta\_cond$ is True in
                  $get\_state(results, subgoal, agent, k, top\_rank)$
                  and (($other\_\alpha\_agent \in best\_states$ and $\alpha\_cond$
                  is True in $best\_states[other\_\alpha\_agent]$) or
                  ($other\_\alpha\_agent \notin best\_states$ and $\alpha\_cond$ is
                  True in
                  $get\_state(results, \alpha\_goal, other\_\alpha\_agent, k, top\_rank)$))
                  do
28                  $top\_rank \leftarrow top\_rank + 1$
29              end
30              $best\_states[agent] \leftarrow$
                  $get\_state(results, subgoal, agent, k, top\_rank)$
31          end
32      end
        /* selection of states for $\mathcal{B}$ goals */
33      foreach $subgoal, agent$ in 1st, 2nd cols. of best_agents do
34          $\alpha\_agents \leftarrow get\_agents(\alpha, subgoal, incomp\_states)$
35          $\alpha\_goal \leftarrow get\_\alpha\_goal(subgoal)$
36          $\beta\_agents \leftarrow get\_agents(\beta, subgoal, incomp\_states)$
37          if $\alpha\_agents = \varnothing$ and $agent \in \beta\_agents$ then
38              $other\_\alpha\_agent \leftarrow \varnothing$
39              foreach $\alpha\_agent \in get\_\alpha\_agents(subgoal)$ do
40                  if $\alpha\_agent \in$ 2nd col. of best_agents then
41                      $other\_\alpha\_agent \leftarrow \alpha\_agent$
42                  end
43              end
44              $\alpha\_cond \leftarrow get\_condition(\alpha, \alpha\_goal, subgoal)$
45              $\beta\_cond \leftarrow get\_condition(\beta, \alpha\_goal, subgoal)$
46              $top\_rank \leftarrow 1$
                /* condition for incompatibility */
47              while $top\_rank \leq k$ and $\beta\_cond$ is True in
                  $get\_state(results, subgoal, agent, k, top\_rank)$
                  and $\alpha\_cond$ is True in $best\_states[\alpha\_agent]$ do
48                  $top\_rank \leftarrow top\_rank + 1$
49              end
50              $best\_states[agent] \leftarrow$
                  $get\_state(results, subgoal, agent, k, top\_rank)$
51          end
52      end
53      return best_states
```

the goal and agent match with *subgoal* and *agent*, respectively, and the state to be fetched (among the k states) has a rank of 1. In lines 9-11, the best states are identified for the $\mathcal{A}$ subgoals. The rationale for identifying the best states of $\mathcal{A}$ subgoals is that once the best states for the $\mathcal{A}\_\mathcal{B}$ (or $\mathcal{B}$) subgoals are being computed, we already know whether the α condition is true in the best states of $\mathcal{A}$ subgoals and subsequently, decide whether the states of $\mathcal{A}\_\mathcal{B}$ (or $\mathcal{B}$) subgoals where β condition is true should be avoided or considered.

In lines 13-32, we compute the best states for the $\mathcal{A}\_\mathcal{B}$ subgoals. The conditional on line 16 is true for only an $\mathcal{A}\_\mathcal{B}$ goal. In lines 17-22, we identify the best agent (*other_α_agent*) for the α goal, from an entry in *incomp_states* where the β goal is *subgoal*. Note that because *subgoal* is an $\mathcal{A}\_\mathcal{B}$ goal, it will appear as both α and β in *incomp_states*. The function $get\_\alpha\_agents(subgoal)$ on line 18 returns a list of candidate agents for the α goal in *incomp_states* where the β goal is *subgoal*. The $get\_\alpha\_goal(subgoal)$ on line 23 fetches the α goal from *incomp_states* where the β goal is *subgoal*. The $get\_condition(\alpha, \alpha\_goal, subgoal)$ on line 24 returns the α condition from *incomp_states* where the α goal is *α_goal* and the β goal is *subgoal*. The while loop condition on line 27 checks for incompatibility, and increments the *top_rank* if the incompatibility condition is true (line 28).

In lines 33-52, we compute the best states for the $\mathcal{B}$ subgoals. The conditional on line 37 is true for only a $\mathcal{B}$ goal. The while loop condition on line 47 checks for incompatibility, and increments the *top_rank* if the incompatibility condition is true (line 48). The $get\_state(results, subgoal, agent, k, top\_rank)$ function returns a state from *results* with rank of *top_rank* provided that $top\_rank \leq k$, else it returns null. In situations where all the top-k ranked states are incompatible, then the value of *top_rank* will be $k+1$ at the end of the while loop on line 49, and $get\_state(results, subgoal, agent, k, top\_rank)$ returns null on line 50. Finally, the dictionary of *best_states* is returned on line 53.

5.2 Correctness Proof

To prove the correctness of our approach, we show that goal states (i.e., *best_states*) returned (line 53 in Algorithm 2) do *not* contain incompatible goal states.

Note that *best_states* contains the best goal state recommended for each individual delegated agent, stored in variable $best\_states[agent]$. This is updated on lines 7, 10, 30 and 50 of Algorithm 2. On lines 7 and 10, since the $\mathcal{A}\_\mathcal{B}$ and $\mathcal{B}$ goals have not yet been considered for the selection of best states, $get\_state(results, subgoal, agent, k, 1)$ always return compatible goal states. Hence, we only need to show that $get\_state(results, subgoal, agent, k, top\_rank)$ returns compatible goal states at lines 30 and 50.

We prove this by contradiction. Let us assume that in line 30 (or line 50) $top\_rank = j$ and an incompatible state is returned by $get\_state(results, subgoal, agent, k, j)$.

Given this assumption, the while loop condition in line 27 (or line 47) should be false for $top\_rank = j$. Let us examine the while loop condition in line 27. This condition (denoted as C) can be broken down into sub-conditions as follows:

- $C_1 \leftarrow j \leq k$
- $C_2 \leftarrow \beta\_cond$ is True in $get\_state(results, subgoal, agent, k, j)$
- $C_3 \leftarrow other\_\alpha\_agent \in best\_states$
- $C_4 \leftarrow \alpha\_cond$ is True in $best\_states[other\_\alpha\_agent]$
- $C_5 \leftarrow other\_\alpha\_agent \notin best\_states$
- $C_6 \leftarrow \alpha\_cond$ is True in $get\_state(results, \alpha\_goal, other\_\alpha\_agent, k, 1)$

Then, C can be expressed as:

$C \leftarrow C_1$ and C_2 and $((C_3$ and $C_4)$ or $(C_5$ and $C_6))$

Therefore, for C to be evaluated as false, one of the following must hold:

- (i) Either C_1 or C_2 should be false, or
- (ii) Either C_3 or C_4 should be false, and either C_5 or C_6 should be false.

For (i), since $j \leq k$ is always true, C_1 cannot be false. Besides, in a pair of incompatible states, the β condition must be true, therefore, C_2 cannot be false.

For (ii), C_3 and C_5 are complementary, i.e., $C_3 = \neg C_5$. This implies that both C_3 and C_5 cannot be false. If C_5 is false, then C_3 is true. In a pair of incompatible states, the α condition must be true, and therefore, C_4 must also be true. If C_3 is false, then C_5 is true, and to suffice the incompatibility requirement of Definition 1, the alpha condition must be true, and therefore, C_6 must also be true. Therefore, it is impossible for the conditions in (ii) to hold given the above assumption.

Since we cannot satisfy either (i) or (ii), C cannot be evaluated as false for *top_rank* equals j. This contradicts our assumption. Therefore, we conclude that in line 30 (or line 50[5]) $get\_state(results, subgoal, agent, k, j)$ cannot return an incompatible state. Hence, our algorithm is correct as it always returns compatible goal states.

[5]The proof is similar to what we have shown for line 30.

6 IMPLEMENTATION

In this section, we outline the Proof of Concept (PoC) application created to evaluate our coordinated self-exploration algorithm in an asynchronous environment, where each agent is provided with a subset of the input akin to a partially-observable environment.

We implement the self-exploration algorithm proposed in section 5 in self-contained agents (Python objects), which are individually containerised using Docker. Docker was chosen as it provides lightweight, isolated environments which are ideal for modelling distributed agent communication on a single host machine. Furthermore, Docker's Python SDK provides a programmatic interface suitable for building a test-bed that allowed us to swiftly test a large range of agent and goal permutations against the self-exploration algorithm.

To facilitate communication between the isolated Docker containers, ZeroMQ (an embedded networking library) was used to build an agent-to-agent messaging interface. We used ZeroMQ for agents to issue commands to each other, and to send the output of self-exploration to the delegator agent. ZeroMQ was chosen for its unique ability to function similarly to a traditional message queue service – but without the need for a centralised message broker that would otherwise be used to route messages to the intended recipient(s) – and also for its resilience features, such as dead letter queuing (DLQ). Our ZeroMQ-based distributed architecture allows us to mirror the communications real-world agents would undertake over a network, within a single host.

Through the use of Python's threading API and ZeroMQ's router sockets, the Docker containers employ data parallelism to run multiple instances of the self-exploration algorithm across threads, allowing an agent to process many goals at any point in time, without halting. To ensure that messages are routed to the appropriate thread, an identity frame (Hintjens et al., 2013) is prepended to messages, allowing the ZeroMQ router socket to forward the message to the intended recipient thread. This significantly reduces the execution time compared to a synchronous PoC implementation.

7 PERFORMANCE EVALUATION

In this section, we evaluate our algorithm's performance by measuring (i) reasoning time and (ii) the number of communication messages sent by agents, across a diverse range of goals and agents.

7.1 Experimental Settings

We measure the reasoning time as the total duration (in seconds) from the commencement of self-exploration by the root agent until the contracts have been established with all the required agents and they have been provided with the best state to transition to. We run the experiments with multiple input files, varying the number of goals and agents. The number of goals is increased in multiples of 4, up to a maximum of 48. The average branching factor for non-leaf goals in the input goal models is 4.7 (maximum is 6 and minimum is 3). Non-leaf goals can be decomposed into AND/OR subgoals, with a maximum of 3 subgoals for each type. In a type, when all three are leaf subgoals, they are divided into $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{A}\_\mathcal{B}$, each. For two leaf subgoals, they are divided into $\mathcal{A}$ and $\mathcal{B}$. The depth of the goal models varies from 1 to 3. The number of candidate agents per leaf goal varies from one to four, while each non-leaf goal is assigned a single agent. Each leaf goal has two attributes, which an agent aims to maximise while an adversary seeks to minimise.

An attribute can have a value from the following seven values: `VERY_LOW`, `LOW`, `MEDIUM_LOW`, `MEDIUM`, `MEDIUM_HIGH`, `HIGH`, and `VERY_HIGH`. The goal satisfied condition of a goal is true when both the attribute values are greater than `MEDIUM_HIGH`, and the goal failed condition is true when both the attribute values are less than `MEDIUM_LOW`. The default goal state has `MEDIUM` values for both the attributes. The incompatibility conditions for both the α and β goals is true if the value of the second attribute equals `VERY_HIGH`. For each incompatibility condition, there is only one incompatibility state; thus, k in the top-k states is set to 3, ensuring a solution with compatible states for each relationship. For each agent, the reasoning budget is set to 50 simulations, as multiple branches can be reasonably explored in this budget. The cut-off depth for each simulation is set to 25 so that computational resources can be effectively managed.

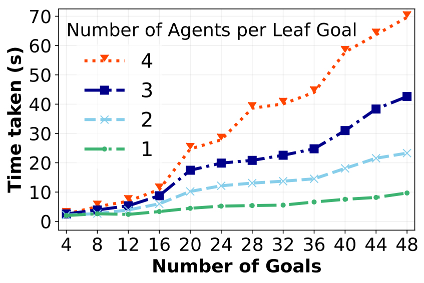
All experiments are run on a Ubuntu Linux system (version 22.04.1), using Docker (version 24.0.5) and Python (version 3.10.12), as indicated earlier. To enable parallel processing within a container, each container runs 7 threads (1 + maximum branching factor). The system specifications are 16GB RAM and a Ryzen 7 4800H processor with a base clock speed of 2.9GHz having 8 CPU cores and 16 threads.

To measure the number of communication messages, we sum the number of communications for: (i) delegate sub-goal requests from the delegator agents to the delegatee agents, (ii) sending win ratios along with top k goal states from the delegatee agents to the delegator agents, (iii) sending the contracts along with the best states from the delegator agents to the delegatee agents, and (iv) acknowledgments of contracts from the delegatee agents to the delegator agents.

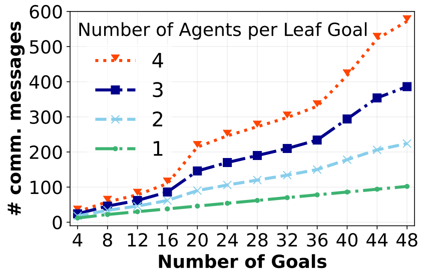
7.2 Results

Figure 6(a) shows the time taken (in seconds) to perform self-exploration when the number of goals is varied from 4 to 48. The number of agents is also varied from one agent per leaf goal to four agents per leaf goal. We observe a relatively linear increase in the time taken when the number of goals is increased. For example, the time taken increases from 2.0 seconds to 9.7 seconds when the number of goals is increased from 4 to 48 while keeping the number of agents per leaf goal fixed at 1. Similarly, there is a relatively linear increase in the time taken when the number of agents per leaf goal is increased. For example, the time taken increases from 9.7 seconds to 69.6 seconds when the number of agents per leaf goal is increased from 1 to 4, while keeping the number of goals fixed at 48.

Figure 6(b) shows the number of communication messages between agents when we vary the number of goals and the number of agents per leaf goal. Similarly to the time taken, we observe a relatively linear trend in the number of communication messages when the number of goals or the number of agents per leaf goal increases. For example, the number of communication messages increases from 12 to 102 when the number of goals is increased from 4 to 48 while keeping the number of agents per leaf goal fixed at 1. In addition, the number of communication messages increases from 102 to 572 when the number of agents per leaf goal increases from 1 to 4, while keeping the number of goals fixed at 48. The results demonstrate that our approach linearly scales with both the number of goals and the number of agents per leaf goal.



(a)



(b)

Figure 6: The impact of the number of goals grouped by the number of agents per leaf goal on (a) time taken (in seconds), and (b) the number of communication messages.

7.3 Threats to Validity

We mitigate *construct validity threats* by measuring both the number of communication messages and the duration of time taken. We mitigate *internal validity threats* by varying (i) the ratio of leaf to non-leaf goals, and (ii) the number of subgoals in the input goal models. In our input goal models, the percentage of non-leaf goals varies from 17.5% to 25% with a median of 21.7%. Furthermore, 29.7%, 10.9%, 20.3% and 39.1% of non-leaf goals had three, four, five and six subgoals, respectively. We acknowledge the *external validity threats* inherent in our simulations and have proactively addressed these by explicitly stating our assumptions (*see* Section 5.1). Nevertheless, the primary focus of our approach remains the coordination mechanism between agents. We mitigate the *conclusion validity threats* by having 48 (12 × 4) data points each for measuring the time taken and the number of communication messages, and ensuring that linear trends were observed within those data points.

8 RELATED WORK

Explicit modelling of goals has been a theme in software engineering for many years. Goal-oriented requirements engineering (GORE) (Van Lamsweerde, 2001; Yu, 1997) attempts to capture the variability of requirements through AND and OR decompositions of ‘hard’ functional goals, and through contribution links to ‘soft’ goals that capture quality requirements. Subsequently such goal models have been extended to model and reason about many other types of goal relationships, such as temporal dependencies (Brown et al., 2006; Liaskos et al., 2010; Letier et al., 2008), fuzziness (Baresi et al., 2010), uncertainty (Cheng et al., 2009), context (Alrajeh et al., 2020; Aradea et al., 2023), etc. See (Horkoff et al., 2019) for a recent survey on goal models in requirements engineering. These functional goals eventually decompose into ‘leaf’ goals that can be mapped to system components and system specifications, e.g., (Yu et al., 2008; Van Lamsweerde, 2009). Beyond informing system design to better align with requirements and in common with the approach taken here, goals have been used as first-class runtime entities that have a goal state the system/agent seeks to achieve or maintain (Dalpiaz et al., 2013; Braubach et al., 2005; Morandini et al., 2017).

Castelfranchi (Castelfranchi, 2000) points out that conflicts between agents only exist in relation to goals – whereas there are “no real conflicts (but just oppositions, contrasts, etc.) between mere forces”. For Castelfranchi, agents make decisions that attempt to actively satisfy a goal. Conflicts occurring between agents are intrinsic and direct when the goals or beliefs that agents hold are logically inconsistent. This distinction between epistemic/knowledge and extrinsic/physical levels has also been adopted by others in discussions of conflict resolution, e.g., (Tessier et al., 2005; Barber et al., 2000; Tang and Basheer, 2017). In this paper, we propose an approach to making the states of goals compatible to enable agents to form effective responses (intentions) to adversarial events.

Other works such as (Wang et al., 2012; Zatelli et al., 2016) focus on conflicting intentions (actions and plans) rather than direct conflicts between the goals themselves. In some cases, the end goal might be shared, but there may be differing means for achieving that goal. Alternatively, while different goals held by agents may not in themselves be conflicting, the action for realising those respective goals may conflict. Such conflict can be characterised as “indirect” conflict in Castlefranchi’s terms. In particular, such conflicts may arise when there are temporal dependencies between the actions specified in plans are in coordination between agents, or where agents share resources in the execution of their goals (Thangarajah and Padgham, 2011).

In multi-agent systems, two primary challenges are task and motion planning (Verma and Ranga, 2021). Motion planning involves all agents working in a distributed manner to ensure collision-free movement (Panagou et al., 2015). However, this distributed approach faces scalability challenges, including increased coordination complexity and communication overhead as the number of agents grows, resulting in degraded response times. In task planning, conflict avoidance has been modelled as a decentralised constraint optimisation problem (DCOP), where concerns are represented as DCOP constraints (Dragan et al., 2023). However, this approach is less effective for complex coordination tasks, as it can result in cycles within the DCOP constraint graph.

There have been several modifications and extensions to MCTS over the last decade to suit different research needs (Świechowski et al., 2022), including parallelising (Cazenave and Jouandeau, 2007; Chaslot et al., 2008) and decentralising (Kurzer et al., 2018; Best et al., 2016; Hashmi et al., 2023) the search process. Cazenave and Jouandea’s (Cazenave and Jouandeau, 2007) approach uses a master/worker communication pattern where the master thread builds the tree and worker threads run simulations and report the win ratios to the master thread, which updates the tree accordingly, making the system relatively centralised. In contrast, Chaslot et al.’s (Chaslot et al., 2008) approach allows multiple threads to share and modify the game tree, using mutexes to manage simultaneous modifications, which can slow down the computation. Kurzer et al. (Kurzer et al., 2018) introduce a decentralised joint-action MCTS with macro-actions to minimise the search space, while Best et al. (Best et al., 2016) propose a decentralised MCTS for joint planning among agents, but do not account for adversaries in the environment. Hashmi et al. (Hashmi et al., 2023) present a decentralised MCTS for a multi-agent setting where each agent computes resilient responses while considering adversaries in the environment; however, the selection of an agent’s response involves no coordination with any other agent. Our approach differs because it enables agents to coordinate and compute resilient responses in an adversarial setting.

9 CONCLUSIONS AND FUTURE WORK

We proposed a coordinated self-exploration approach where agents coordinate with their delegator agents to identify compatible responses for their goals, focusing on the case of pair-wise goal dependencies. Our approach is applicable in partially-observable, contested environments where goals can have incompatible states. Our approach resolves incompatible goal states in the planning phase through coordination between agents, and remains linear when the number of goals and agents increase in amounts proportionate to the scenario type and size considered in this paper.

In terms of future work, several directions merit further investigation. First, to complement our approach, a purely distributed coordination mechanism needs to be developed, whereby a delegatee agent would coordinate with another delegatee agent (as opposed to the delegator agent) to reduce single points of failure. This will involve the most promising candidate agents for the given goals interacting, where the agent for one goal communicates with the agent for another goal and adapts its reasoning accordingly.

Second, settings involving more than two goals in an incompatibility relationship need to be explored as part of a more general framework for addressing goal inter-dependencies. Such a framework should, ideally, also address circular dependencies (e.g., through recursive checks for incompatibilities) and dependencies at varying depths or outside immediate subgoals/teams (*see* Section 5.1). Lastly, while this paper focuses on agents in a single team, complex systems often consist of multiple collaborating teams. Such a system-of-systems approach (cf. (Boardman and Sauser, 2006)) may introduce inter-team goal dependencies, i.e., dependencies between goals associated with agents belonging to different teams. Addressing such dependencies will require selective sharing to achieve coordinated agent actions.

ACKNOWLEDGEMENTS

This work was undertaken as part of the "Autonomic Cyber Resilience and Antifragility" Collaborative Research Project (CRP), supported by the Australian Department of Defence – Next Generation Technologies Fund initiative (Cyber theme).

REFERENCES

Alrajeh, D., Cailliau, A., and van Lamsweerde, A. (2020). Adapting requirements models to varying environments. In *Procs. of the ACM/IEEE 42nd Int. Conference on Software Engr.*, pages 50–61.

Aradea, Supriana, I., and Surendro, K. (2023). Aras: adaptation requirements for adaptive systems: Handling runtime uncertainty of contextual requirements. *Automated Software Engr.*, 30(1):2.

Atighetchi, M. and Pal, P. (2009). From auto-adaptive to survivable and self-regenerative systems successes, challenges, and future. In *8th IEEE Int. Symp. on Network Computing and Applications*, pages 98–101.

Barber, K. S., Liu, T. H., and Han, D. C. (2000). Strategic decision-making for conflict resolution in dynamic organized multi-agent systems. *A Special Issue of CERA Journal*.

Baresi, L., Pasquale, L., and Spoletini, P. (2010). Fuzzy goals for requirements-driven adaptation. In *2010 18th IEEE Int. Requirements Engr. Conference*, pages 125–134. IEEE.

Baruwal Chhetri, M., Uzunov, A. V., Vo, B., Nepal, S., and Kowalczyk, R. (2019). Self-improving autonomic systems for antifragile cyber defence: Challenges and opportunities. In *Procs. of 16th IEEE Int. Conference on Autonomic Computing (ICAC)*, pages 18–23. IEEE.

Baruwal Chhetri, M., Uzunov, A. V., Vo, Q. B., Kowalczyk, R., Docking, M., Luong, H., Rajapakse, I., and Nepal, S. (2018). AWaRE – towards distributed self-management for resilient cyber systems. In *Procs. of the 23rd Int. Conference on Engr. of Complex Computer Systems (ICECCS)*, pages 185–188. IEEE.

Best, G., Cliff, O. M., Patten, T., Mettu, R. R., and Fitch, R. (2016). Decentralised Monte Carlo Tree Search for active perception. In *Procs. of the 12th Int. Workshop on the Algorithmic Foundations of Robotics (WAFR)*. Springer.

Boardman, J. and Sauser, B. (2006). System of Systems - the meaning of *of*. In *2006 IEEE/SMC Int. Conference on System of Systems Engr.*, pages 118–123. IEEE.

Braubach, L., Pokahr, A., Moldt, D., and Lamersdorf, W. (2005). Goal representation for BDI agent systems. In *Programming Multi-Agent Systems: Second Int. Workshop ProMAS 2004, New York, NY, USA, July 20, 2004, Selected Revised and Invited Papers 2*, pages 44–65. Springer.

Brown, G., Cheng, B. H. C., Goldsby, H., and Zhang, J. (2006). Goal-oriented specification of adaptation requirements engineering in adaptive systems. In *Procs. of the 2006 Int. workshop on Self-adaptation and Self-managing Systems*, pages 23–29.

Castelfranchi, C. (2000). Conflict ontology. In *Computational Conflicts: Conflict Modeling for Distributed Intelligent Systems*, pages 21–40. Springer.

Cazenave, T. and Jouandeau, N. (2007). On the parallelization of UCT. In *Computer games workshop*.

Chaslot, G. M. J.-B., Winands, M. H. M., and van Den Herik, H. J. (2008). Parallel Monte-Carlo Tree Search. In *Int. Conference on Computers and Games*, pages 60–71. Springer.

Cheng, B. H. C., Sawyer, P., Bencomo, N., and Whittle, J. (2009). A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. In *Int. Conference on Model Driven Engr. Languages and Systems*, pages 468–483. Springer.

Dalpiaz, F., Borgida, A., Horkoff, J., and Mylopoulos, J. (2013). Runtime goal models. In *7th Int. Conference on Research Challenges in Info. Sci.(RCIS)*, pages 1–11. IEEE.

Dragan, P.-A., Metzger, A., and Pohl, K. (2023). Towards the decentralized coordination of multiple self-adaptive systems. In *2023 IEEE Int. Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, pages 107–116. IEEE.

Florio, L. (2015). Decentralized self-adaptation in large-scale distributed systems. In *Procs. of the 2015 10th Joint Meeting on Foundations of Software Engr.*, pages 1022–1025.

Franch, X., López, L., Cares, C., and Colomer, D. (2016). The i* framework for goal-oriented modeling. In *Domain-Specific Conceptual Modeling*, pages 485–506. Springer.

Hashmi, S. S., Dam, H. K., Baruwal Chhetri, M., Uzunov, A. V., Colman, A., and Vo, Q. B. (2025). Proactive self-exploration: Leveraging information sharing and predictive modelling for anticipating and countering adversaries. *Expert Systems with Applications*, 267:1–17, article 126118.

Hashmi, S. S., Dam, H. K., Smet, P., and Baruwal Chhetri, M. (2022). Towards Antifragility in Contested Environments: Using Adversarial Search to Learn, Predict, and Counter Open-Ended Threats. In *Procs. of the IEEE Int. Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, pages 141–146. IEEE.

Hashmi, S. S., Dam, H. K., Uzunov, A. V., Baruwal Chhetri, M., Ghose, A., and Colman, A. (2023). Goal-Driven Adversarial Search for Distributed Self-Adaptive Systems. In *Procs. of IEEE Int. Conference on Software Services Engr. (SSE)*, pages 198–209. IEEE.

Hintjens, P. et al. (2013). Advanced Request-Reply Patterns. In *ZeroMQ: messaging for many applications*, chapter 3. O'Reilly Media, Inc.

Horkoff, J., Aydemir, F. B., Cardoso, E., Li, T., Maté, A., Paja, E., Salnitri, M., Piras, L., Mylopoulos, J., and Giorgini, P. (2019). Goal-oriented requirements engineering: an extended systematic mapping study. *Requirements Engr.*, 24:133–160.

IC3 (2023). FBI Internet Crime Report 2023. https://www.ic3.gov/Media/PDF/AnnualReport/2023_IC3Report.pdf.

Ismail, Z. H., Sariff, N., and Hurtado, E. (2018). A survey and analysis of cooperative multi-agent robot systems: challenges and directions. *Applications of Mobile Robots*, 5:8–14.

Kocsis, L. and Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *European Conference on Machine Learning*, pages 282–293. Springer.

Kurzer, K., Zhou, C., and Zöllner, J. M. (2018). Decentralized cooperative planning for automated vehicles with hierarchical Monte Carlo Tree Search. In *2018 IEEE Intelligent Vehicles Symp. (IV)*, pages 529–536. IEEE.

Letier, E., Kramer, J., Magee, J., and Uchitel, S. (2008). Deriving event-based transition systems from goal-oriented requirements models. *Automated Software Engr.*, 15:175–206.

Liaskos, S., McIlraith, S. A., Sohrabi, S., and Mylopoulos, J. (2010). Integrating preferences into goal models for requirements engr. In *2010 18th IEEE Int. Requirements Engr. Conference*, pages 135–144. IEEE.

Linkov, I. and Kott, A. (2019). Fundamental concepts of cyber resilience: Introduction and overview. *Cyber Resilience of Systems and Networks*, pages 1–25.

Morandini, M., Penserini, L., Perini, A., and Marchetto, A. (2017). Engineering requirements for adaptive systems. *Requirements Engr.*, 22(1):77–103.

Panagou, D., Stipanović, D. M., and Voulgaris, P. G. (2015). Distributed coordination control for multi-robot networks using Lyapunov-like barrier functions. *IEEE Transactions on Automatic Control*, 61(3):617–632.

Ponniah, J. and Dantsker, O. D. (2022). Strategies for scaleable communication and coordination in multi-agent (uav) systems. *Aerospace*, 9(9):488.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Świechowski, M., Godlewski, K., Sawicki, B., and Mańdziuk, J. (2022). Monte Carlo Tree Search: A review of recent modifications and applications. *Artificial Intelligence Review*, pages 1–66.

Tang, A. Y. and Basheer, G. S. (2017). A conflict resolution strategy selection method (ConfRSSM) in multi-agent systems. *Int. Journal of Advanced Computer Science and Applications*, 8(5).

Tesauro, G., Chess, D. M., Walsh, W. E., Das, R., et al. (2004). A multi-agent systems approach to autonomic computing. In *Procs. of the Third Int. Joint Conference on Autonomous Agents and Multiagent Systems*, volume 1, pages 464–471.

Tessier, C., Chaudron, L., and Müller, H.-J. (2005). *Conflicting agents: conflict management in multi-agent systems*, volume 1. Springer Science & Business Media.

Thangarajah, J. and Padgham, L. (2011). Computationally effective reasoning about goal interactions. *Journal of Automated Reasoning*, 47:17–56.

Uzunov, A. V., Baruwal Chhetri, M., and Wondoh, J. (2021). GOURMET: A methodology for realizing goal-driven self-adaptation. In *Procs. of 2021 ACM/IEEE Int. Conference on Model Driven Engr. Languages and Systems Companion (MODELS-C)*, pages 197–202. IEEE.

Uzunov, A. V., Vo, Q. B., Dam, H. K., Harold, C., Baruwal Chhetri, M., Colman, A., and Hashmi, S. S. (2023). Adaptivity and Antifragility. In Kott, A., editor, *Autonomous Intelligent Cyber Defense Agent (AICA): A Comprehensive Guide*, volume 87 of *Advances in Information Security*, chapter 10. Springer Nature.

Van Lamsweerde, A. (2001). Goal-oriented requirements engineering: A guided tour. In *Procs. 5th IEEE Int. Symp. on Requirements Engr.*, pages 249–262. IEEE.

Van Lamsweerde, A. (2009). *Requirements Engineering: From system goals to UML models to software*, volume 10. Chichester, UK: John Wiley & Sons.

Verma, J. K. and Ranga, V. (2021). Multi-robot coordination analysis, taxonomy, challenges and future scope. *Journal of intelligent & robotic systems*, 102:1–36, article 10.

Wang, X., Cao, J., and Wang, J. (2012). A runtime goal conflict resolution model for agent systems. In *2012 IEEE/WIC/ACM Int. Conference on Web Intelligence and Intelligent Agent Technology*, volume 2, pages 340–347. IEEE.

Weyns, D. and Georgeff, M. (2009). Self-adaptation using multiagent systems. *IEEE software*, 27(1):86–91.

Yu, E. (1997). Towards modelling and reasoning support for early-phase requirements engineering. In *Procs. of ISRE'97: 3rd IEEE Int. Symp. on Requirements Engr.*, pages 226–235. IEEE.

Yu, Y., Lapouchnian, A., Liaskos, S., Mylopoulos, J., and Leite, J. C. (2008). From goals to high-variability software design. In *Foundations of Intelligent Systems: 17th Int. Symposium, ISMIS 2008 Toronto, Canada, May 20-23, 2008 Procs. 17*, pages 1–16. Springer.

Zatelli, M. R., Hübner, J. F., Ricci, A., and Bordini, R. H. (2016). Conflicting goals in agent-oriented programming. In *Procs. of the 6th Int. Workshop on Programming Based on Actors, Agents, and Decentralized Control*, pages 21–30.