Decision-Theoretic Planning with Communication in Open and Typed Multiagent Systems

by

Anirudh Kumar Maurya Kakarlapudi

(Under the Direction of Prashant Doshi)

Abstract

In many real-world applications with multiple agents, the agents often take a time-out to leave and enter the system. In such open systems that allow agents to enter and leave, individual planning is a challenging task as the agents need to predict the presence of others accurately in addition to their actions. For example, in the wildfire suppression domain, the firefighting agents may run out of the suppressants while fighting the fires and leave the system to refill their resources before joining the fight again. This research focuses on planning in such open systems while allowing agents to model and communicate with only a subset of others in a restricted communication channel. Though the communicative acts make the prediction of the presence of others easy, predicting what actions they take is tricky as the agents can lie among themselves. In addition to that, the communicative acts directly impact the beliefs and mental states of the interacting agents. This research extends on the latest developments in the CIPOMDP framework in open and multi-agent systems. Planning in such systems is complex and computationally intractable. This research improves the scalability of the CIPOMDP framework by modeling, communicating with only a subset of neighboring agents, and extrapolating their behavior to the entire population. Besides, the proposed framework opts for the Monte Carlo tree search to plan for the single agent. Furthermore, the agents form strictly closed groups called cliques, restricting the modeling and communication to the group. The proposed method achieves parallelization of planning using a server-client architecture where each client runs single-agent planning for a step to improve the speed. Experiments in the wildfire suppression problems show that the proposed method uses communication to increase coordination among the agents and, in turn, achieve better rewards compared to the other baseline models. The results also exhibit the potency of the framework in systems that require high coordination.

Index words: POMDP, Monte Carlo, MCTS, Decision Planning, CPOMCP, IPOMDP, CIPOMDP, IPOMCP-$l1$, CIPOMCP-$l1$

Decision-Theoretic Planning with Communication in Open and Typed Multiagent Systems

by

Anirudh Kumar Maurya Kakarlapudi

B.E., Andhra University, INDIA, 2017

A Thesis Submitted to the Graduate Faculty of the
University of Georgia in Partial Fulfillment of the Requirements for the Degree.

Master of Science

Athens, Georgia

2021

Decision-Theoretic Planning with Communication in Open and Typed Multiagent Systems

by

Anirudh Kumar Maurya Kakarlapudi

Major Professor:	Prashant Doshi
Committee:	Frederick Maier
	Adam Eck

Acknowledgments

I want to thank my committee members Dr.Prashant Doshi, Dr.Adam Eck, and Dr. Frederick Maier, for their time and suggestions throughout my research. A special thanks to my major professor Dr. Prashant Doshi, Dr.Adam Eck from the Oberlin College, and Dr. Leen-kiat Soh from the University of Nebraska-Lincoln for helping me from time to time and bringing in newer perspectives.

I am thankful to the Institute of Artificial Intelligence at UGA and the CS Department at Oberlin College for providing the computational resources that helped me run the experiments. I am incredibly grateful to my family, friends, and colleagues at THINC Lab.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Autonomous agents can replace humans effectively and efficiently in many real-world applications like search and rescue, transportation, cyber-security, and wildfire suppression. Simply put, an autonomous agent is any entity that can make rational decisions on its own and receive observations from the environment to accomplish its goal. In addition, the agent may not have proper sensors, which leads to improper observations leading to uncertainty. In such scenarios that involve uncertainty, the agent's planning and its decision-making are of high importance, which garnered the attention of many AI researchers and, in turn, led to the development of various algorithms in the field of decision-planning.

Systems that have two or more autonomous agents are known as Multi-Agent systems(MAS). Figure 1.1 depicts the simple MAS with two agents. In a MAS, every agent takes a decision or action and receives an observation from the environment. In addition, the environment depends on the actions taken by some or all agents. So the agents need to factor in other agents' actions during the planning process to take the optimal actions. In other words, an agent in MAS benefits from reasoning about the behavior of other agents and by predicting the actions they would take. For example, in the autonomous ride-sharing cars environment, the autonomous car can maximize its revenue by predicting and avoiding its competitors' locations and moving to other places where it would expect high customer availability.

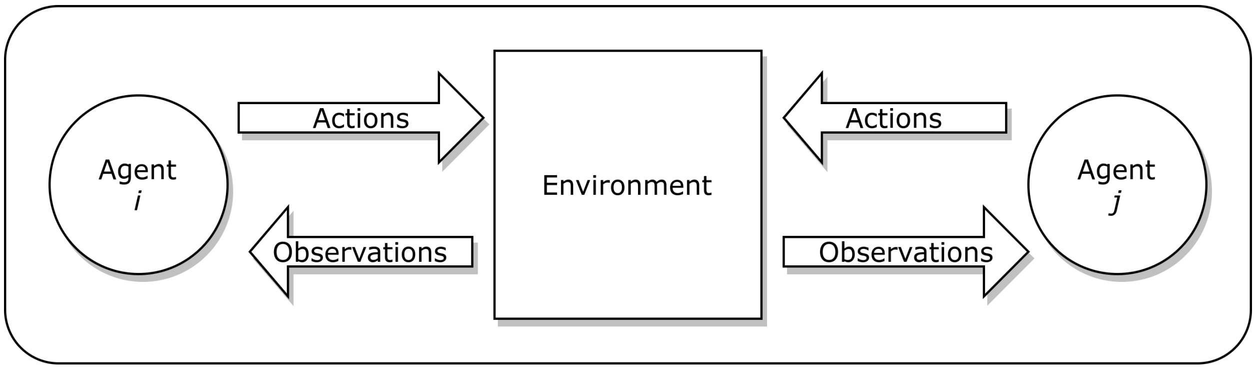


Figure 1.1: Multi-Agent System (Russell & Norvig, 2010).

Modeling other agents in multi-agent systems has its challenges when openness is involved, especially 1) Agent openness, 2) Type openness, and 3) Task openness. Type refers to the difference in properties or skills of the agents, and type openness exists in the system in which the agents can change their type at any time. Task openness refers to those systems in which the task or goal may change. An open-agent system is a system in which an agent can enter or leave at any time. The addition of new agents at any time increases the complexity of decision planning as the agents should plan for the new agents. For example, adding a new car might make the other cars lose a rider in the autonomous ride-sharing car domain. In this research, we limit our attention to agent openness and limit the openness with our assumption that the new agents cannot enter the system, but the existing agents can enter and leave the system at any time. Though this assumption brings down the complexity to a certain extent, the agent still needs to predict whether the other agents or neighbors are present or absent in the system. Inaccurate predictions about the presence of other agents in the system may cause the agent to pick actions that do not give the expected reward or utility. In a similar example, if an autonomous car predicts the presence of other cars in the system wrong, it might lose the riders and revenue.

The introduction of communicative acts improves the prediction of the absence and presence of agents in the system. The agents use the communicated messages to get a faster inference about the agent that sends the corresponding message. These communicative acts will influence the interacting agent's mental state, such as its belief. In other words, the communicative acts would change the receiving agent's belief and, in turn, its action. For example, in the autonomous ride-sharing domain, a car might send a message that it is leaving the system. The receiving cars use this message to infer that the sending agent is not competing in the environment for some time. Since the agents need not communicate honestly, the interacting agents might get the wrong inferences. We model such problems as decision planning problems with communication in open and typed multi-agent systems.

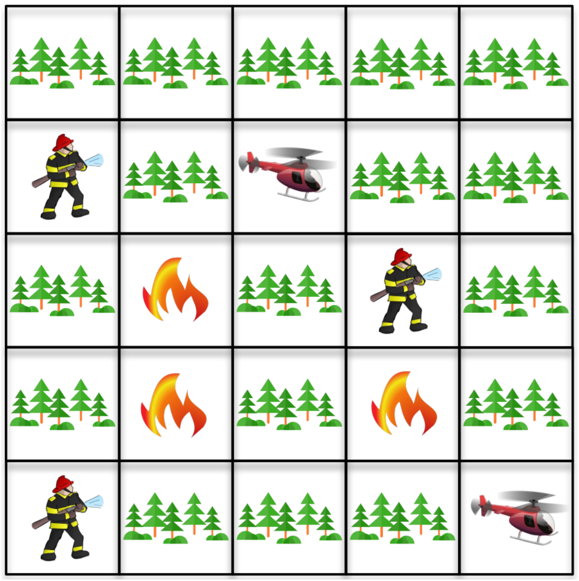


Figure 1.2: An example of wildfire suppression domain in a 5 x 5 grid of forest (Chandrasekaran et al., 2016)

To illustrate the above scenarios in a cooperative environment, consider the wildfire suppression domain with five agents in a 5x5 grid of forest area as shown in Figure 1.2[1]. Three-ground firefighters and two-helicopters are of two types as they can have different fighting power or different suppressant levels. Each agent needs to predict what other agents do in the system to put out the fires effectively. Furthermore, the agents may run out of suppressants fighting the fires, and it leaves the system to refill and may not join the fight for few steps. If any agent predicts the presence of its neighbor wrong and starts fighting the fire, it wastes valuable resources as not enough agents are present in the system to fight the particular fire. The

[1]Some of the icons from www.flaticon.com are used to create the figures and the use is licensed by CC BY 3.0

use of communicative acts like *'No Suppressant'* will signal the receiving agent that the sending agent is likely to be absent for some time steps.

1.1 Related Work

Partial observability is one of the critical aspects of many real-world applications. For example, a simple automated cleaning robot does not know where it is located in a system. It needs to find its way in cleaning the system while estimating its location. The POMDP is a formalization of such problems in a single agent system, and it generalizes the Markov Decision Process (Russell & Norvig, 2010). Solving a POMDP suffers from the curse of dimensionality and curse of history (Hsu et al., 2007; Kaelbling et al., 1998). The curse of dimensionality is because the belief space is exponential in the number of state variables. The curse of history is because the action observation histories are exponential in the planning horizon. So, in large multi-agent systems, solving a POMDP is intractable.

Many techniques have been developed to counter the high computational complexity of the above frameworks. One such technique is to employ a Monte Carlo Tree Search(MCTS) to find each state's utility in the search tree of each agent. The use of MCTS improves the system's scalability as it breaks the curse of dimensionality and the curse of history by intelligently exploring and exploiting the tree efficiently. The usage of MCTS made solving the POMDPs in large spaces possible (Katt et al., 2017; Lee et al., 2018; Silver & Veness, 2010; Sunberg & Kochenderfer, 2017).

In a multi-agent system, the POMDPs can be generalized into Decentralized POMDPs and Interactive POMDPs. DEC-POMDP is a decentralized planning technique in a cooperative setting, where the agents share the initial beliefs, get a collaborative reward after each time step (Bernstein et al., 2013; Oliehoek & Amato, 2016). The IPOMDPs can be used in a cooperative or competitive setting. In IPOMDPs, each agent has its reward function, and it maintains mental models over the other agents while planning individually (P. Gmytrasiewicz, 2020). Like POMDPs, IPOMDPs suffer from the curse of history and the curse of dimensionality - the belief space in IPOMDPs is exponential in the interactive state space.

Besides, IPOMDPs also suffer from the curse of nested reasoning - the nested nature of IPOMDPs makes the beliefs of agents nested.

In many real-world applications, the system exhibits frame-action anonymity; that is, the state transitions in the system depend on the number of agents doing an action rather than the specifics of the agent (Sonu et al., 2017). The use of this property can increase the scalability of the system for the POMDP and IPOMDP planning greatly (Eck et al., 2020).

The CIPOMDP framework (P. Gmytrasiewicz, 2020) extends the IPOMDPs by introducing communication between the agents. This communication impacts the respective beliefs of both interacting agents. The addition of restricted communication brings newer perspectives that are discussed in detail in the next chapter.

1.2 Contributions

This research makes the following contributions

1. Use the methods like Frame-action Anonymity and MCTS to scale the CIPOMDP planning in open and multi-agent systems(MAS). The experiments are run with 2, 3, 4 agents to demonstrate the scalability in the number of agents.

2. Instead of sequentially planning for each agent, the server-client architecture is used to plan for each agent in parallel. From the results, we establish that using the server-client architecture, the time taken for the runs is significantly reduced with the increase in the number of agents and planning parameters.

3. Assess the behavior and benefit of communication with the benchmark simulations of the wildfire domain (Chandrasekaran et al., 2016) with 2,3,4 agents for CIPOMCP-$l1$. From the results, we observe that the CIPOMCP-$l1$ uses the communication to increase coordination among the agents, thereby putting out the high reward yielding shared fire more. In contrast, the baselines used their resources to put down individual fires.

1.3 Document Structure

This thesis is organized into five chapters.

- In Chapter 1, Introduction, we start by introducing the autonomous agents and multi-agent systems. Then, we summarize the thesis and review the related work.
- In Chapter 2, Background, we revisit some of the existing frameworks and concepts like POMDP, CIPOMDP, and Frame Action Anonymity.
- In Chapter 3, Methodology, we illustrate the concepts of communication, f-function, Monte Carlo tree search, and the algorithm used to perform the CIPOMCP planning.
- In Chapter 4, Experiments, we start by describing about the run time optimizations. Then we introduce the setups, baselines followed by an analysis of the experimental results and the run-time results using the server-client architecture.
- In Chapter 5, Conclusion, we give the deductions from the results, framework limitations, and possible lines of future work.

Chapter 2

Background

In this chapter, we present a background on several fundamental frameworks and concepts that form the foundation for methods used in the later chapters. This chapter is organized as follows:

- In the section 2.1, we first define the framework of Partially Observable Markov Decision Processes(POMDPs).
- In section 2.2, we describe the extension of Interactive Partially Observable Markov Decision Processes(IPOMDP) with communication, called the communicative IPOMDP(CIPOMDP) framework.
- In section 2.3, we introduce the concept of frame action anonymity and its application in the CIPOMDP framework.

2.1 Partially Observable Markov Decision Processes(POMDP) framework

A partial observable Markov decision process (POMDP) (Åström, 1965; Cassandra et al., 1994; Kaelbling et al., 1998) is a generalization of the Markov decision process (MDP) in which agents cannot directly observe the environment or system. The following tuple defines a POMDP

$$\text{POMDP} \triangleq \langle S, A, \Omega, T, O, R, \gamma \rangle$$

- S is the physical state space of the environment. Here, S is the set of states of the decision-making problem, possibly factored into multiple variables $F_1 \times F_2 \times \cdots F_k$, such as the intensities of k wildfires in the wildfire domain.
- A is the set of actions an agent can perform.
- Ω is the set of observations.
- $T : S \times A \times S' \rightarrow [0, 1]$ is a stochastic state transition function which provides a distribution over the next state(S') after performing an Action (A) on the current state (S).
- $O : S' \times A \times \Omega \rightarrow [0, 1]$ is a stochastic observation function that provides the probability with which an agent receives an observation(o) given the resulting state (S') and the performed action (A).
- $R : S \times A \rightarrow \mathbb{R}$ is a reward function that provides the reward for the agent given its state and action from the state.
- $\gamma \in (0, 1]$ s a discount factor used for weighting the reward values over the range of future time steps.

As partial observability prevents the agent from knowing its current state, it maintains a probability distribution over possible current states, referred to as an agent's belief or a belief state (b). The agent uses the previous belief(b) over the states, the agent's action at a previous time step(a), and the received observation(o) after the transition to update its belief at each time step using the Bayes rule. The new belief, b' for a current possible state(s') is given by

$$b'(s') = Pr(s'|o, s, a) = \alpha O(o, s', a) \sum_{s \in S} b(s) T(s, a, s') \tag{2.1}$$

where α is a normalization constant.

The solution to a POMDP is a policy (π), which is a function that maps the agent's beliefs to a distribution over actions, $\pi : B \times A \to [0, 1]$. Simply put, a policy is a strategy for an agent to act in the given state. A value function $(V^h(b))$ in the form of a Bellman equation is defined to get a long-term expected long-term reward as the utility of action over the given belief. The bellman equation is defined as the

$$V^h(b) = \max_{a \in A} \sum_{s \in S} b(s) \left(R(s, a) + \gamma \sum_{s'} T(s, a, s') \sum_{o \in \Omega} O(o, a, s') V^{h-1}(\tau(b, o, a)) \right) \tag{2.2}$$

where τ is the belief update as in equation 2.1 and h is the horizon.

2.2 Communicative Interactive POMDP (CIPOMDP) framework

Communicative Interactive POMDP (CIPOMDP) (P. Gmytrasiewicz, 2020) builds on the well-known finitely-nested I-POMDP (P. J. Gmytrasiewicz & Doshi, 2005) framework in a straightforward way to include an additional action of sending a message, an additional observation of receiving a message, and a set of messages that are sent or received. Formally,

$$\text{CI-POMDP}_{i,l} \triangleq \langle Ag, IS_{i,l}, A, \Omega_i, M, T_i, O_i, R_i, \gamma, b^0_{i,l} \rangle$$

- Ag is a finite set of agents, consisting of a subject agent$i$ using the CI-POMDP to decide how to act and communicate with other agents $j, \ldots, z$ modeled by subject agent i.

- $IS_{i,l}$ is the set of level l interactive states, $IS_{i,l} = S \times \Theta_{j,l-1} \times \Theta_{k,l-1} \times \ldots \times \Theta_{z,l-1}$ for $l > 0$. Here, S is the set of states of the decision-making problem, possibly factored into variables $F_1 \times F_2 \times \ldots \times F_N$, such as the intensities of the N wildfires the agents need to suppress. Agent j in Ag is ascribed a model $\theta_{j,l-1} = \langle b_{j,l-1}, \hat{\theta}_j \rangle$ from the set of computable models $\Theta_{j,l-1}$ where $b_{j,l-1}$ is the agent's belief over its level $l-1$ interactive state and $\hat{\theta}_j$ denotes the agent's frame. A frame represents the agent's capabilities and preferences. The level 0 interactive states $IS_{i,0} = S$.

- $A = A_i \times A_j \times \ldots \times A_z$ is the set of possible joint actions of the agents; e.g., the individual fires that each agent chooses to fight. For notational convenience, $\mathbf{a_{-i}} \in A_j \times \ldots \times A_z$ denotes the joint action by agents in $Ag \setminus \{i\}$.

- Ω_i is the set of observations of agent i.

- M is the set of messages that are sent and received by any agent. Let $m_{i\rightarrow j} \in M$ denote a message that is sent to an agent j and $m_{i\leftarrow j} \in M$ denote a message that is received from j. Let $\mathbf{m}_{i\rightarrow -i}$ denote the vector of messages sent to all other agents, and analogously for received messages $\mathbf{m}_{i\leftarrow -i}$.

- $T_i(s, a_i, \mathbf{a_{-i}}, s') = P(s'|s, a_i, \mathbf{a_{-i}})$ gives the probabilities of stochastic state transitions caused by the actions of Ag.

- $O_i(s', a_i, \mathbf{a}_{-i}, o_i) = P(o_i|a_i, \mathbf{a_{-i}}, s')$ models the probabilities of stochastic observations revealed to subject agent i after joint action $(a_i, \mathbf{a_{-i}})$.

- $R_i(s, a_i, \mathbf{a}_{-i}, \mathbf{m}_{i\rightarrow -i}) \in \mathbb{R}$ is the reward function of agent i dependent on the state, joint actions, and messages sent to the other agents. While there is a cost of sending messages, there is no cost to receiving (and processing) messages.

- $\gamma \in (0, 1]$ and $b^0_{i,l}$ are the discount factor and initial belief state of i over its level-l interactive state space, respectively.

An agent with $l > 0$ in the CI-POMDP framework updates its belief on performing an action and/or sending a message at the previous time step followed by receiving an observation and/or a message at the current time step. The belief update shown below yields the new belief($b^t_{i,l}$).

$b^t_{i,l} = Pr(IS^t_{i,l}|b^{t-1}_{i,l}, a^{t-1}_i, \mathbf{m}^{t-1}_{i\to -i}, o^t_i, \mathbf{m}^t_{i\leftarrow -i})$

$$\begin{aligned} b^t_{i,l}(is^t) = \alpha \sum_{is^{t-1}} b_{i,l}(is^{t-1}) \\ \times \prod_{j\in Ag/\{i\}} \left(\sum_{a^{t-1}_j} Pr(a^{t-1}_j, m^{t-1}_{j\to i}|\theta^{t-1}_{j,l-1}) \right) T_i(s^{t-1}, a^{t-1}_i, \mathbf{a}^{t-1}_{-i}, s^t) \\ \times O_i(s^t, a^{t-1}_i, \mathbf{a}^{t-1}_{-i}, o^t_i) \prod_{j\in Ag/\{i\}} \Bigg(\sum_{o^t_j} \tau_{\hat{\theta}_j}(b^{t-1}_{j,l-1}, a^{t-1}_j, m^{t-1}_{j\to i}, o^t_j, m^t_{j\leftarrow i}, b^t_{j,l-1}) \\ \times O_j(s^t, a^{t-1}_j, \mathbf{a}^{t-1}_{-j}, o^t_j) \Bigg) \end{aligned} \tag{2.3}$$

Here, $m^{t-1}_{j\to i}$ is the message sent by agent j to i at timestep $t-1$, which is same as the message received by agent i from j at timestep t, $m^t_{i\leftarrow j}$, as the framework assumes a perfect communication channel. Therefore, the term $Pr(a^{t-1}_j, m^{t-1}_{j\to i}|\theta^{t-1}_{j,l-1})$ makes those models of j that support sending this message more probable. $\tau_{\hat{\theta}_j}(b^{t-1}_{j,l-1}, a^{t-1}_j, m^{t-1}_{j\to i}, o^t_j, m^t_{j\leftarrow i}, b^t_{j,l-1})$ is 1 if agent j's belief in is^{t-1} on performing its predicted action a^{t-1}_j and sending message to i $m^{t-1}_{j\to i}$ (which is same as the message received by agent i from j) followed by receiving possible observation o^{t-1}_j and i's sent message to j, $m^t_{j\leftarrow i}$, updates to $b^t_{j,l-1}$ in is^t. A level-0 agent updates its belief using the POMDP belief update by first marginalizing the other agent from the transition and observation functions using a fixed probability distribution.

Analogously to I-POMDPs, subject agent i assigns a value to each level l belief, which is the expected cumulative, discounted rewards over a finite or infinite horizon H, $r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots + \gamma^{H-1} r_{H-1}$, by maximizing over the Bellman equation for each belief and action-message pair:

$$Q^t_i(b^t_{i,l}, a^t_i, \mathbf{m}^t_{i\to -i}) = \rho_i(b^t_{i,l}, a^t_i, \mathbf{m}^t_{i\to -i}) + \gamma \sum_{o^{t+1}_i, \mathbf{m}^{t+1}_{i\leftarrow -i}} Pr(o^{t+1}_i, \mathbf{m}^{t+1}_{i\leftarrow -i}|b^t_{i,l}, a^t_i, \mathbf{m}^t_{i\to -i})\, V^{t+1}_i(b^{t+1}_{i,l}) \tag{2.4}$$

$$V_i^t(b_{i,l}^t) = \max_{a_i \in A_i, \mathbf{m}_{i\to -i}^t} Q_i^t(b_{i,l}^t, a_i^t, \mathbf{m}_{i\to -i}^t) \tag{2.5}$$

where

$$\rho_i(b_{i,l}^t, a_i^t, \mathbf{m}_{i\to -i}^t) = \sum_{is^t \in IS_{i,l}^t} b_{i,l-1}^t(is^t) \sum_{\mathbf{a}_{-i} \in A_{-i}} \prod_{j \in Ag} \sum_{\mathbf{m}_{j\to -j}^t} Pr(a_j^t, \mathbf{m}_{j\to -j}^t | \theta_{j,l-1}^t)\, R_i(s, a_i, \mathbf{a}_{-i}, \mathbf{m}_{i\to -i}^t)$$

and $b_{i,l}^{t+1}$ is the updated belief on performing action a_i^t and sending messages $\mathbf{m}_{i\to -i}^t$ followed by receiving observation o_i^{t+1} and messages $\mathbf{m}_{i\leftarrow -i}^{t+1}$.

Policy $\pi_{i,l}$ is then the distribution of those action and message pairs that maximize the Q-value:

$$OPT(b_{i,l}^t) = \arg\max_{a_i, \mathbf{m}_{i\to -i}} Q_i^t(b_{i,l}^t, a_i, \mathbf{m}_{i\to -i}) \tag{2.6}$$

and $\pi_{i,l}(a_i^*, \mathbf{m}_{i\to -i}^* | b_{i,l}^t) = \frac{1}{|OPT|}$ where $(a_i^*, \mathbf{m}_{i\to -i}^*) \in OPT$.

2.3 Agent Anonymity in IPOMDP

In a Multi-agent system, the transitions (T), observations (O), and rewards (R) depend on the joint action taken by all the agents. As the number of agents in the system increase, the IPOMDP planning becomes intractable since the representation of T, O, R is exponential in the number of possible joint actions. For such systems, a simple exploit called Frame-action anonymity (Sonu et al., 2017) can be implemented to reduce the above exponential dependency provided that the T, O, and R depend on the number of agents performing each action rather than the which agent performs actions. Simply put, Frame action anonymity relaxes the agent identities so that multiple joint actions can lead to the same transitions, observations, and rewards if they yield the same action count vector. The Action count vector, called Configuration is represented as $C = \langle n_{a_1,\theta_1}, \cdots, n_{a_{|A|},|\Theta|} \rangle$ where n_{a_1,θ_1} is the count of agents of frame or type θ_1 that perform the action a_1 at the current time step (Eck et al., 2020). Now the representation of T, O, R is greatly reduced as they depend on the number of possible configurations $\binom{|Ag|+m+1}{m+1}$ which inturn is polynomial in the number of agents where $m = \max\{|A_i|, |A_j|, \cdots, |A_z|\}$.

In addition to that, the state in an IPOMDP is an interactive state which contains the state and mental model of the neighbors of a subject agent. As the number of agents increases, the interactive state-space grows exponentially in the number of agents. The frame action anonymity property allows the subject agent to maintain beliefs for a small random subset of agents from each frame and extrapolate their predicted behavior to all the agents in corresponding frames with some bounded error. However, the introduction of communication inhibits the random sampling of agents from each frame which is explained in section 3.2, in detail.

Frame action anonymity also facilitates the construction and use of a single policy for all level-0 agents in a frame. Since the level-0 agent does not model others in the environment, maintaining a policy table consisting of a policy for every possible message set, frame, state, internal state, and planning horizon is possible. Using the policy table in level-1planner saves the computation time could refer to the table to get the possible actions for neighbors instead of constructing trees for the neighbors. However, the higher-level planning is entirely online because constructing a policy table should for every possible belief of the subject agent (itself and its neighbors), frame, state, internal state, and planning horizon for higher-level planners is intractable in space and time.

Chapter 3

Methodology

In this chapter, we introduce several concepts that are employed in this research before laying out the algorithm. The rest of the chapter is organized as follows:

- In section 3.1, we describe how communication impacts the mental models of the agents.
- Next in section 3.2, we introduce the complication in belief update caused by nested modeling that inhibits communication and the need for f- function and g-function.
- In section 3.3, we define the concept of cliques and how communication and modeling of agents occur inside the cliques.
- In section 3.4, we start by discussing the Monte Carlo Tree Search and then the changes introduced in it to use the entire particle filter to update the tree.
- Finally, in section 3.5, we discuss the CIPOMCP algorithm used in this research in detail.

3.1 Communication

By maintaining a new state variable $present \in \{true, false\}$ for each neighbor in its neighborhood($N(i) \subseteq Ag$, let$N(i) = \{j, k, l, \cdots, z\}$), the subject agent (say i) can reason about the presence and absence of the corresponding neighbor. However, the state space increases exponentially due to the addition of a new state variable. The new state variable $present$ is placed inside the mental model of the neighbor to eliminate the exponential bloat up in state-space.

In general, the planning agent senses its neighbors' presence by observing the changes in state-space variables. The use of communication in the CIPOMDP framework allows the agents to infer the presence of their neighbors' faster. However, the inference due to the communication might not be accurate as the agents can always lie among themselves. In any case, the subject agent who receives the messages from its interacting neighbor will update the mental models corresponding to that neighbor using the belief update equation 2.3. The mental models likely to result in the message received will receive a higher weight than the other models that will not likely give the same message. For example, in the firefighting domain, the message the agent could send is among the set of messages, M=*{'Full suppressant', 'Half suppressant', 'No suppressant', and 'No Message'}*. So if the subject agent receives a *"No Suppressant"* message from its neighbor, it would presume that the neighbor is likely to miss the next step in refueling and update the mental models accordingly.

3.2 Nested Modeling Complicates Communication and use of F,G-Functions

Though the CIPOMDP framework offers a way to integrate communication, the nested modeling of others inhibits communication. A level-l CIPOMDP agent (say i) receives the set of messages ($\mathbf{m_{i \leftarrow -i}}$) from an agent at level-($l+1$). It uses this set of messages to update its belief and model the level-($l-1$) neighbors. This nested modeling and communication continue down to level-0. As the level-0 agent is a POMDP agent, it does not ascribe intentional models to others in the environment. It will consider others in the environment as the random agents and assumes a flat distribution over their actions. As a result, the messages received from others will not influence the level-o agent's belief. Because the level-0 agent does not model anyone, it may choose not to communicate as it does not think its message will impact others in the system. An unintended consequence of this is that level-1 agent may decide not to communicate with its neighbors because it reasons that any message that it sends may not influence its level-0 neighbor's belief. A level-1 agent modeling these level-0 neighbors won't expect to receive any messages as it thinks that its neighbors will not communicate, resulting in undesired behavior.

We introduce a simple way to incentivize the level-1 agent to communicate. Similar to the level-0 agents being literal listeners and literal speakers in the original CIPOMDP framework(P. Gmytrasiewicz, 2020), we consider that the level-0 agents can listen and send messages using the f and g functions, respectively. Let $f_j : m_{r,j} \longrightarrow a_j$ which maps a message from agent j's message to its action $a_j \in A_j$. Similarly let $g_j : s_j \longrightarrow m_j$ maps the agent j's state to its message. Consequently, a level-0 POMDP agent models others using an f-function instead of considering them as random agents and maps their messages to actions. For more than one other agent, denote the vector of maps, one for each other agent, as $\mathbf{f_{-i}}$.

Then, i's updated belief $b^t_{i,0}(s^t)$is:

$$\begin{aligned} b^t_{i,0}(s^t) &= Pr(s^t | b^{t-1}_{i,0}, a^{t-1}_i, g^{t-1}_i(s^{t-1}), o^t_i, \mathbf{m^t_{i\leftarrow -i}}) \\ &= \alpha O_i(s^t, a^{t-1}_i, \mathbf{f_{-i}}(\mathbf{m^t_{i\leftarrow -i}}), o^t_i) \\ &\times \sum_{s^{t-1}} b^{t-1}_{i,0}(s^{t-1}) T_i(s^{t-1}, a^{t-1}_i, g^{t-1}_i(s^{t-1}), \mathbf{f_{-i}}(\mathbf{m^t_{i\leftarrow -i}}), s^t) \end{aligned} \tag{3.1}$$

Let this update be common knowledge among the agents. The update above is analogous to a POMDP belief update with a slight modification that allows messages from others in the neighborhood to impact the updated belief. As the level-1 agent is aware that an agent modeled at level-0 updates its belief using the Equation 3.1, it may reason to communicate with its others because the messages sent by it may impact others' beliefs over the state. This impact may change their behavior and, in turn, affects their action choice. Consequently, the agent at level 2 or higher may choose to communicate with others and expects to receive messages from others, which now enables the use of the CIPOMDP framework for its decision-making.

3.3 Communication Cliques

In the absence of communication, the agents randomly sample their neighbors, model them, and use their behavior to extrapolate to all the agents in the system. When communication is possible between the agents, the subject agent's modeled neighbors are the ones with whom the subject agent can communicate. In other words, the subject agent cannot benefit from communication with a neighbor that it does not explicitly model. To illustrate this, in a three agent system with agents $\{i, j, k\}$, let agent i models only j. So it maintains a belief of j. If agent i receives a message from k, it does not know how to use this obtained information as it does not model k that is, it does not maintain the belief of agent k. So it cannot update its belief about the environment or update the k's model using Equation 3.1. In systems where new agents can enter, the communication with neighbors that the subject agent does not model can occur. As we focus on the systems where new agents cannot join, the agents will only communicate with the sampled neighbors and vice versa. Each such set of agents model and communicate with everyone in the set form a clique as shown in Figure3.1.

In a scalable system with a high number of agents, the behavior of all the agents can be predicted by sampling a few neighbors randomly and extrapolating their behavior. Let $\hat{n}_{a,\hat{N}_{\hat{\theta}}(i)}$ be the number of sampled agents whose predicted action is a, where $a \in \{A_i \cup A_j \cup \ldots \cup A_z\}$, from the sampled neighborhood $\hat{N}_{\hat{\theta}}(i)$. Eck et al., 2020, proposed that the agents can extrapolate the behavior of sampled agents into all the agents in the respective frame with the prediction error less than some given error ϵ provided that agents sample the minimum($n_{\hat{\theta}}$) number of agents in each frame.

In small agent systems, each clique should have at least two agents from each frame. In such scenarios, the extrapolation would not give an accurate estimate of the behavior of all agents. If there is only one agent per frame, there can be only a single clique, and each agent inside the clique models every one where the extrapolation error would be zero.

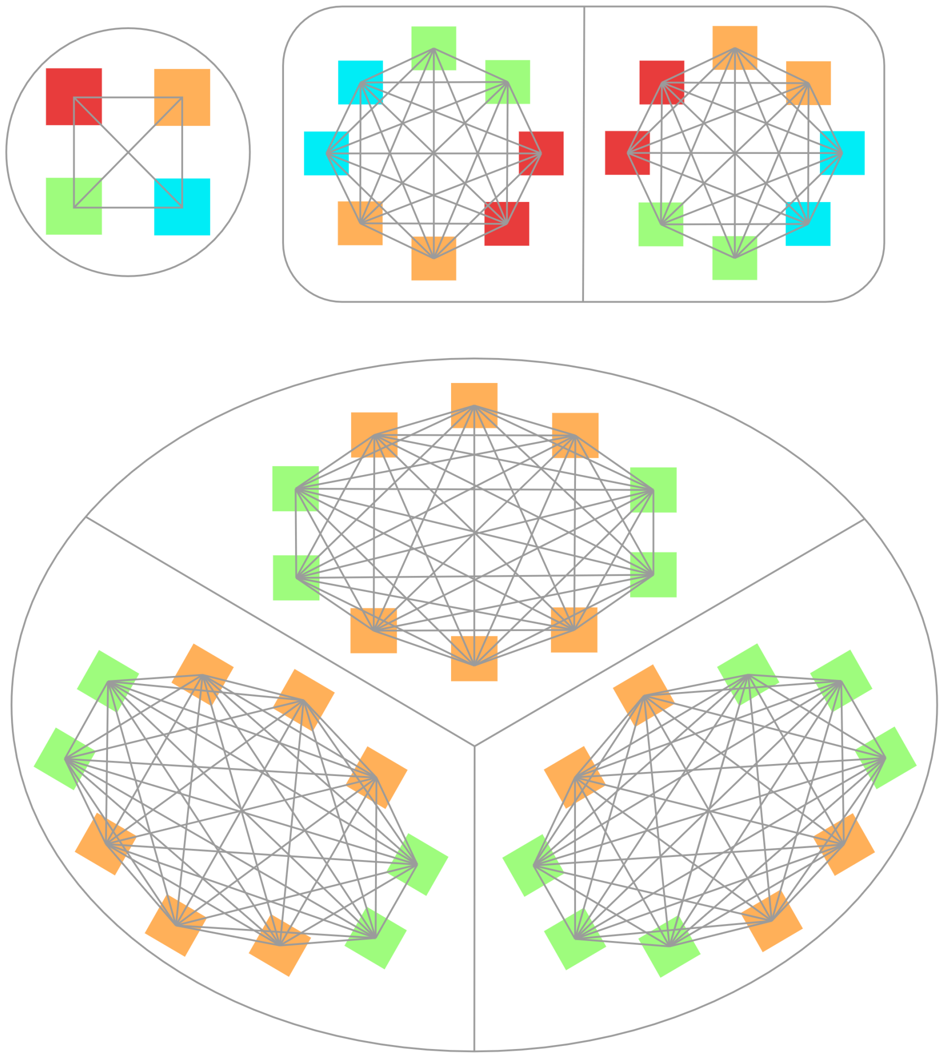


Figure 3.1: Different types of cliques - a) Single clique(Top left) b) Two Cliques(Top right) c) Three cliques(Bottom). Here the colored nodes represent different types of agents. Each connected edge represents the agents modeling each other. Here each agent models the equal number of agents, and so the cliques are with the same number of agents.

3.4 Monte Carlo Tree Search in CIPOMCP

MCTS is a well-known online decision planning framework for solving large state spaces like in MAS. The UCB heuristic in the MCTS algorithm balances the exploration and exploitation of the nodes in the search tree, thus saving time in exploring the most promising nodes. It also breaks the curse of dimensionality and the curse of history. to solve POMDPs and IPOMDPs. Unlike POMDPs, the belief space is interactive for IPOMDP, and hence the belief update is nested.

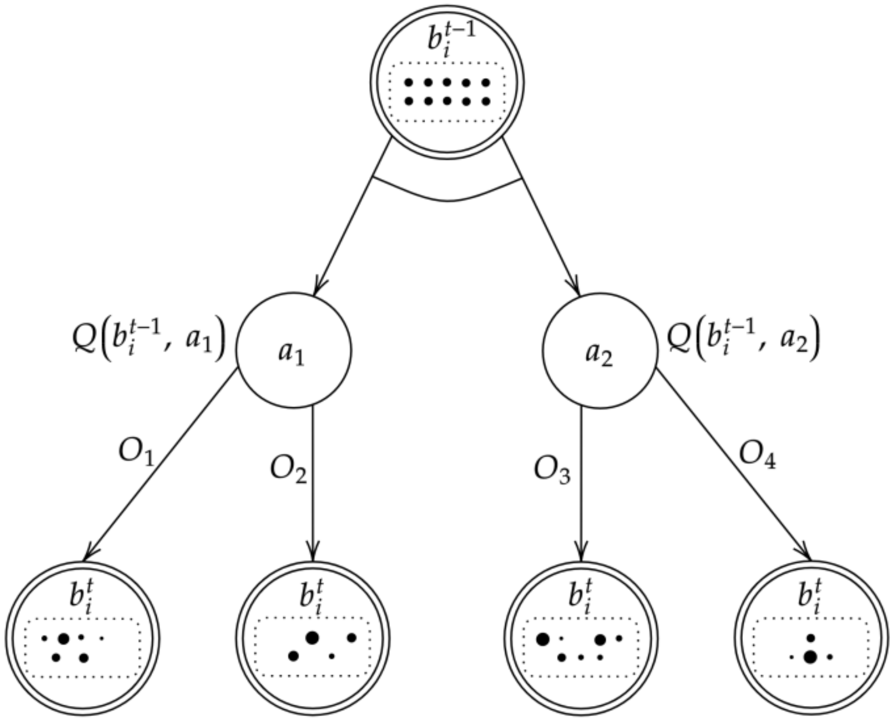


Figure 3.2: Monte Carlo Tree Search in CIPOMCP(Garg et al., 2019). Initial belief: b_i^{t-1}. Next belief: b_i^t. $Q(b_i^{t-1}, a_1)$: Q-value of the action node a_1 from the initial belief. O_1, O_2, O_3, O_4: Different Observations. The black filled circles represent the particles inside the particle filter, and the size of the black filled circles: weight of the corresponding particle

The normal MCTS samples the belief from the distribution of beliefs and propagates the sampled belief inside the tree. However, the entire particle filter inside the tree can give a better estimate of belief and improve the algorithm's scalability (Somani et al., 2013; Thomas et al., 2021). We use the particle filter version of MCTS in this research to solve POMDP with communication, referred to as CPOMCP and CIPOMDP, referred to as CIPOMCP.

The MCTS employs a UCB heuristic to explore the search tree efficiently provided a sufficient planning horizon (h), and a decent ucb constant is selected. Figure 3.2. shows how the MCTS simulation for a single timestep. In simple words, the simulation starts with the initial belief i.e.,particle filter (b_i^{t-1}). The subject agent (say i) picks the best possible action given by the ucb heuristic. The joint action is formed by planning for the neighbors. Different possible observation_comm ($\langle$observation, $\mathbf{m}_{i \leftarrow -i}\rangle$) leads to different next beliefs (b_i^t) formed by updating the initial belief (b_i^{t-1}). The planning continues for the next horizon with the most likely belief from the set of beliefs. The corresponding visits at various nodes are updated during the planning. Also, the discounted reward is used to update the Q-values of the nodes.

3.5 CIPOMCP Algorithm

The Monte-Carlo Tree Search is used to solve the CIPOMDP framework and is referred to as CIPOMCP. The algorithm 1 explains the primary CIPOMCP algorithm of this research for MAS. In algorithm1, the CIPOMCP procedure for the subject agent at level-l iteratively constructs the MCTS tree using the UpdateTree procedure for a given number of trajectories. This process creates the MCTS trees for its neighbors at level-($l-1$) using the CIPOMCP for the neighbor agent, which creates the MCTS trees similarly for its neighbors at level-($l-2$) and so-on till the level-1 agent is planning reached where it uses CPOMCP to plan for its neighbors. During each of these trajectories, the algorithm would pass the entire particle filter, the set of messages the subject agent received from its neighbors at level -($l+1$), into the tree as illustrated in line 3. At the highest level of planning, the agent starts with the set of messages $\mathbf{m}_{i \leftarrow -i}$ containing all no message. After updating the tree for given trajectories, the algorithm would return the action, the message having the optimal q-value, and the observation particle filters constructed as part of the planning process as illustrated on line 5.

The update tree procedure passes the entire particle filter to every branch of the tree. Initially, it adds the particle filter to the root belief node. If the root belief node is already visited, it picks the best possible action using the UCB heuristic under that belief node, as illustrated in line 16. We construct the empty observation particle filters β_o, weight lists for these particle filters, the list of message sets for all possible observations, list of weights for the corresponding message sets for all weights. Now, for each particle in the particle filter, we get state from the particle and plan for the neighbors' actions, messages their set of observation particle filters using the procedure PlanNeighbors as illustrated in lines 23-26. Then we sample the configuration and simulate the next state and reward (r_k), which in turn is multiplied with the corresponding particle weight as illustrated in line 31. The Nested Belief Update procedure is used to update the get the new particle. Through lines 32-37, the new particle is added to the observation particle filter with the corresponding observation weight. The simulated messages from line 29 are saved in the list of message sets for corresponding observation, and the observation weight is saved as a weight for this message set to the list of weights for message lists. We sample the observation and message set by normalizing the weights and sampling as illustrated in steps 38-39. The observation_comm is then formed by combining the sampled observation and message set. Then the new particle filter is taken from the resampled observation particle filter that corresponds to the sampled observation. This new particle filter is used to recursively call the UpdateTree procedure with increased horizon each time to calculate the overall reward obtained, as illustrated in line 43. The visits for the action node, belief node, and the expected utility for the action node are updated, as shown in lines 44-47.

If the root belief node is not visited before, a leaf node is created, and then the RolloutAll procedure is used to perform the rollout for each particle in the particle filter. The Plan Neighbors procedure is used to trees nested with levels for the neighbors to plan for their optimal action and message. In addition to these values, the particle filters corresponding to the visited belief nodes as part of planning under the action node are returned as the observation particle filters for different observation _comm and internal state pairs. The NestedBeliefUpdate procedure updates the nested belief of the subject agent. To save the computation time, the observation_comm that the neighbor would receive from the domain is calculated and used to get the corresponding precomputed belief which is calculated when planning for neighbors in PlanNeighbors procedure, as illustrated in line 14. If the precomputed belief is not found for the corresponding observation_comm, a BayesianUpdate on the current belief for the new state s' is performed.

The difference between the POMCP and CPOMCP is that the POMCP agent assumes others in the system as the random agents, whereas the CPOMCP agent assumes that the others pick actions based on the f-function so that the messages the CPOMCP gets from level-1 neighbor influences its decision as discussed in section 3.2. Also, the CPOMCP agent does not maintain the beliefs of its neighbors. Algorithm2 shows the pseudocode for the CPOMCP. The CPOMCP is similar to the CIPOMCP except that it simulates the messages that it would receive with *NextMessages* as it estimates the behavior of other agents using f-function.

Algorithm 1 CIPOMCP$_{\mathcal{O}}$

Constants: τ:number of trajectories to perform. $N(i)$:neighborhood of subject agent i. $\hat{N}(i)$:$\{j,\dots,z\}$ are the subset of neighbors modeled by subject agent i. $N_\theta(i)$ and $\hat{N}_\theta(i)$: neighbors of subject agent i with frame θ. c: is the constant from UCB-1.
Variables: $T_{i,l}$- agent i's level-l tree (initially empty). h:history of actions and observations. $B_{i,l}(h)$:particle filter representing agent i's level:l belief. $n_{i,l}$:the count of the number of visits to each node. $Q_{i,l}$: agent i's level-l approximated Q function. $\beta_{i,l}$: particle filter of agent i at level-l. w: weight associated with particle p in the particle filter(β) and w_o is weight of observation. $\overline{\beta}_i$: set of observation particle filters. oc: observation-communication.

```
1: procedure CIPOMCP($\beta_{i,l}, \mathbf{m}_{i\leftarrow -i}, l$)
2:     for $traj \in 1, 2, \dots, \tau$ do
3:         UpdateTree($\beta_{i,l}, \mathbf{m}_{i\leftarrow -i}, 0, \varepsilon, l$)
4:     $a_i^*, m_i^* \leftarrow \underset{a\in A_i, m\in M}{\text{argmax}}\ Q_{i,l}(\varepsilon, a, m)$
5:     return $a_i^*, m_i^*, \overline{\beta}_\mathbf{i}$

1: procedure RolloutAll($\beta, t$)
2:     $R \leftarrow 0$
3:     for $(s, w) \in \beta$ do
4:         $R \leftarrow R + w\cdot$ Rollout$(s, t)$
5:     return $R$
```

```
6:  procedure PlanNeighbors($s, p, l, m_i, \mathbf{m}_{\mathbf{i}\leftarrow-\mathbf{i}}$)
7:      $\bar{\beta}_{\mathbf{j}} \leftarrow \emptyset$
8:      $\mathbf{m'}_{i\leftarrow -i} \leftarrow \emptyset$
9:      for $j \in \hat{N}(i)$ do
10:         $\beta_{j,l-1} \leftarrow$ getBelief$(j, p, l-1)$
11:         $\mathbf{m}_{\mathbf{j}\leftarrow-\mathbf{j}} \leftarrow \mathbf{m}_{\mathbf{i}\leftarrow-\mathbf{i}} \cup \{m_i\} \setminus \{m_j\}$
12:         if $l - 1 \geq 1$ then
13:             $a_j, m_j, \bar{\beta}_{\mathbf{j}} \leftarrow$ CIPOMCP$(B_{j,l-1}, \mathbf{m}_{\mathbf{j}\leftarrow-\mathbf{j}}, l-1)$
14:         else
15:             $a_j, m_j, \bar{\beta}_{\mathbf{j}} \leftarrow$ CPOMCP$(B_{j,0}, \mathbf{m}_{\mathbf{j}\leftarrow-\mathbf{j}})$
16:         $\mathbf{a}_{\hat{\mathbf{N}}(\mathbf{i})} \leftarrow \mathbf{a}_{\hat{\mathbf{N}}(\mathbf{i})} \cup \{a_j\}$
17:         $\mathbf{m'}_{i\leftarrow -i} \leftarrow \mathbf{m'}_{i\leftarrow -i} \cup \{m_j\}$
18:         $\bar{\beta}_{-\mathbf{i}} \leftarrow \bar{\beta}_{-\mathbf{i}} \cup \bar{\beta}_{\mathbf{j}}$
19:     return $\mathbf{a}_{\hat{\mathbf{N}}(\mathbf{i})}, \mathbf{m'}_{i\leftarrow -i}, \bar{\beta}_{-\mathbf{i}}$

1:  procedure SampleConfiguration($\mathbf{a}_{\hat{\mathbf{N}}(\mathbf{i})}$)
2:      $C(a,\theta) \leftarrow 0, \hat{n}_{\pi(s)=a,\hat{N}_\theta(i)} \leftarrow 0 \quad \forall a, \theta$
3:      for $\mathcal{M}_{j,l-1} \in \mathbf{M}_{\mathbf{l-1}}$ do
4:          $a_j \leftarrow \mathbf{a}_{\hat{\mathbf{N}}(\mathbf{i})_j}$
5:          $\theta_j \leftarrow$ frame$(\mathcal{M}_{j,l-1})$
6:          $\hat{n}_{\pi(s)=a_j,\hat{N}_{\theta_j}(i)} \leftarrow \hat{n}_{\pi(s)=a_j,\hat{N}_{\theta_j}(i)} + 1$
7:      for $\theta \in \Theta$ do
8:          for $a \in A$ do
9:              $\hat{p}_{a,\theta} \leftarrow \hat{n}_{\pi(s)=a,\hat{N}_\theta(i)} \,/\, |\hat{N}_\theta(i)|$
10:         for $j \in N_\theta(i)$ do
11:             $a \sim Cat(\hat{p}_{a_1,\theta}, \hat{p}_{a_2,\theta}, \ldots, \hat{p}_{a_{|A|},\theta})$
12:             $C(a,\theta) \leftarrow C(a,\theta) + 1$
13:     return $\mathbf{C}$

1:  procedure SampleRandomConfiguration($s$)
2:      $C(a,\theta) \leftarrow 0 \quad \forall a, \theta$
3:      for $\theta \in \Theta$ do
4:          for $j \in N_\theta(i)$ do
5:              $a \sim Uniform(A_j)$
6:              $C(a,\theta) \leftarrow C(a,\theta) + 1$
7:      return $\mathbf{C}$
```

```
8: procedure UpdateTree($\beta_{i,l}, \mathbf{m}_{i\leftarrow -i}, t, h, l$)
9:     if $t \geq H$ then
10:        return 0
11:    $B_{i,l}(h) \leftarrow B_{i,l}(h) \cup \beta_{i,l}$
12:    if $h \notin T_{i,l}$ then
13:        $T_{i,l} \leftarrow T_{i,l} +$ CreateLeaf$(h, l)$
14:        return RolloutAll$(\beta_{i,l}, t)$
15:    else
16:        $a_i, m_i \leftarrow \underset{a \in A_i, m \in M}{\text{argmax}}\ Q_{i,l}(h, a, m) + c\sqrt{\frac{\log n_{i,l}(h)}{n_{i,l}(ham)}}$
17:        $r \leftarrow 0$
18:        for $o \in Z$ do
19:            $\beta_o \leftarrow \emptyset$
20:            $\omega(o) \leftarrow 0$
21:            $\bar{\mathbf{m}}_{\mathbf{o,i}\leftarrow -\mathbf{i}} \leftarrow \emptyset$
22:            $\mathbf{w}_{\mathbf{m,o}} \leftarrow \emptyset$
23:        for $(p_{k,l}, w) \in \beta_{i,l}$ do
24:            $s \leftarrow p_{k,l}$
25:            $\mathbf{a}_{\hat{\mathbf{N}}(\mathbf{i})}, \mathbf{m'}_{i\leftarrow -i}, \bar{\beta}_{-\mathbf{i}} \leftarrow$ PlanNeighbors$(s, \beta_{\mathbf{i,l}}, l-1, \mathbf{m}_{\mathbf{i}\leftarrow -\mathbf{i}})$
26:            $\mathbf{C} \leftarrow$SampleConfiguration$\left(\mathbf{a}_{\hat{\mathbf{N}}(\mathbf{i})}\right)$
27:            $s', r_k \leftarrow$ Simulate$(s, a_i, \mathbf{C})$
28:            $\mathbf{a} \leftarrow a_i \cup \mathbf{a}_{\hat{\mathbf{N}}(\mathbf{i})}$
29:            $\mathbf{m} \leftarrow m_i \cup \mathbf{m}_{\mathbf{i}\leftarrow -\mathbf{i}}$
30:            $p' \leftarrow$NestedBeliefUpdate$(s', p_{k,l}, \beta_{i,l}, \mathbf{a}, \mathbf{m}, \bar{\beta}_{-\mathbf{i}})$
31:            $r \leftarrow r + w \cdot r_k$
32:            for $o \in Z$ do
33:                $w_o \leftarrow w \cdot O(s', a_i, o)$
34:                $\beta_o \leftarrow \beta_o \cup (p', w_o)$
35:                $\omega(o) \leftarrow \omega(o) + w_o$
36:                $\mathbf{m}_{\mathbf{o,i}\leftarrow -\mathbf{i}}(k) \leftarrow \mathbf{m}'_{\mathbf{i}\leftarrow -\mathbf{i}}$
37:                $\mathbf{w}_{\mathbf{m,o}}(k) \leftarrow w_o$
38:        $o_i \sim Norm(\omega)$
39:        $\mathbf{m}'_{\mathbf{i}\leftarrow -\mathbf{i}} \sim Norm(\mathbf{w}_{\mathbf{m,o}}, \mathbf{m}_{\mathbf{o_i,i}\leftarrow -\mathbf{i}})$
40:        $oc_i \leftarrow (o_i, m'_{i\leftarrow -i})$
41:        $\beta' \leftarrow Resample(\beta_{oc_i})$
42:        $h' \leftarrow h a_i m_i o_i \mathbf{m}_{i\leftarrow -i}$
43:        $R \leftarrow r + \gamma \cdot$ UpdateTree $(\beta', \mathbf{m'}_{i\leftarrow -i}, t+1, h', l)$
44:        $n_{i,l}(h) \leftarrow n_{i,l}(h) + |\beta_{i,l}|$
45:        $n_{i,l}(h a_i m_i) \leftarrow n_{i,l}(h a_i m_i) + |\beta_{i,l}|$
46:        $Q_{i,l}(h, a_i, m_i) \leftarrow Q_{i,l}(h, a_i, m_i) + \frac{R - Q_{i,l}(h, a_i, m_i)}{n_{i,l}(h a_i m_i)}$
47:        return $R$
```

```
48: procedure NestedBeliefUpdate($s', p, \mathbf{a}, \mathbf{m}, o_i, \bar{\beta}_{-\mathbf{i}}$)
49:     $p'(i,l) \leftarrow belief(s')$
50:     for $a_j \in Ag_{-i}$ do
51:         $p'(j, l-1) \leftarrow \emptyset$
52:     for $j \in \hat{N}(i)$ do
53:         $\mathbf{m}'_{\mathbf{j} \leftarrow -\mathbf{j}} \leftarrow \mathbf{m}$
54:         $oc_j \leftarrow (o_j, \mathbf{m}'_{\mathbf{j} \leftarrow -\mathbf{j}})$
55:         if $oc_j \in \bar{\beta}_{-\mathbf{i}}$ then
56:             $\beta'_{j,l-1} \leftarrow \bar{\beta}_{-\mathbf{i}}$
57:         else
58:             $\beta_{j,l-1} \leftarrow p(j, l-1)$
59:             $\beta'_{j,l-1} \leftarrow BayesianUpdate(s', \beta_{j,l-1})$
60:         $p'(j, l-1) \leftarrow \beta'_{j,l-1}$
61:     return $p'$

1: procedure Rollout($s, t$)
2:     $R \leftarrow 0, t' \leftarrow t$
3:     while $t < H$ do
4:         $\mathbf{C} \leftarrow$ SampleRandomConfiguration($s$)
5:         $a_i \leftarrow$ SampleAction($A_i$)
6:         $s', o_i, r_i \leftarrow$ Simulate($s, a_i, \mathbf{C}$)
7:         $R \leftarrow R + \gamma^{t-t'} \cdot r_i, t \leftarrow t+1, s \leftarrow s'$
8:     return $R$

1: procedure CreateLeaf($h, l$)
2:     $B_{i,l}(h) \leftarrow \emptyset$
3:     $n_{i,l}(h) \leftarrow 0$
4:     for $a \in A_i$ do
5:         for $m \in M$ do
6:             $n_{i,l}(ham) \leftarrow 0$
7:             $Q_{i,l}(h, a, m) \leftarrow 0$
```

Algorithm 2 CPOMCP$_{\mathcal{O}}$

```
1: procedure UpdateTree($\beta_{i,0}, \mathbf{m}_{i\leftarrow -i}, t, h$)
2:     if $t \geq H$ then
3:         return 0
4:     $B_{i,0}(h) \leftarrow B_{i,0}(h) \cup \beta_{i,0}$
5:     if $h \notin T_{i,0}$ then
6:         $T_{i,0} \leftarrow T_{i,0} + \text{CreateLeaf}(h)$
7:         return RolloutAll($\beta_{i,0}, t$)
8:     else
9:         $a_i \leftarrow \operatorname{argmax}_{a \in A_i} Q_{i,0}(h, a) + c\sqrt{\frac{\log n_{i,0}(h)}{n_{i,0}(ha)}}$
10:        $r \leftarrow 0$
11:        for $o \in Z$ do
12:            $\beta_o \leftarrow \emptyset$
13:            $\omega(o) \leftarrow 0$
14:        for $j \in \hat{N}(i)$ do
15:            $a_j \leftarrow f(m_{i\leftarrow j})$                    ▷ F-Function
16:            $\mathbf{a}_{\hat{\mathbf{N}}(\mathbf{i})} \leftarrow \mathbf{a}_{\hat{\mathbf{N}}(\mathbf{i})} \cup \{a_j\}$
17:        for $(s, w) \in \beta_{i,0}$ do
18:            $\mathbf{C} \leftarrow \text{SampleConfiguration}\left(\mathbf{a}_{\hat{\mathbf{N}}(\mathbf{i})}\right)$
19:            $\mathbf{m'}_{i\leftarrow -i} \leftarrow \text{NextMessages}(\mathbf{m}_{i\leftarrow -i})$
20:            $s', r_i \leftarrow \text{Simulate}(s, a_i, \mathbf{C})$
21:            $r \leftarrow r + w \cdot r_i$
22:            for $o \in Z$ do
23:                $w_o \leftarrow w \cdot O(s', a_i, o)$
24:                $\beta_o \leftarrow \beta_o \cup (s', w_o)$
25:                $\omega(o) \leftarrow \omega(o) + w_o$
26:        for $o \in Z$ do
27:            $oc \leftarrow (o, \mathbf{m'_{i\leftarrow -i}})$
28:            $\bar{\beta}_{\mathbf{i}} \leftarrow \bar{\beta}_{\mathbf{i}} \cup (oc, \beta_o)$
29:        $o_i \sim Norm(\omega)$
30:        $\beta' \leftarrow Resample(\beta_{o_i})$
31:        $h' \leftarrow h a_i o_i \mathbf{m'}_{i\leftarrow -i}$
32:        $R \leftarrow r_i + \gamma \cdot \text{UpdateTree}(s', \mathbf{m'}_{i\leftarrow -i}, t+1, h')$
33:        $n_{i,0}(h) \leftarrow n_{i,0}(h) + |\beta_{i,0}|$
34:        $n_{i,0}(ha_i) \leftarrow n_{i,0}(ha_i) + |\beta_{i,0}|$
35:        $Q_{i,0}(h, a_i) \leftarrow Q_{i,0}(h, a_i) + \frac{R - Q_{i,0}(h,a_i)}{n_{i,0}(ha_i)}$
36:        return $R$
```

```
37: procedure CPOMCP($\beta_{i,0}, \mathbf{m}_{i\leftarrow -i}$)
38:     for $traj \in 1, 2, \ldots, \tau$ do
39:         UpdateTree($\beta_{i,0}, \mathbf{m}_{i\leftarrow -i}, 0, \varepsilon$)
40:     $a_i^* \leftarrow \underset{a \in A_i}{\operatorname{argmax}}\, Q_{i,0}(\varepsilon, a)$
41:     $m_i \leftarrow$ EstimateMessage($s$)
42:     return $a_i^*, m_i, \bar{\beta}_{\mathbf{i}}$

1: procedure CreateLeaf($h$)
2:     $B_{i,0}(h) \leftarrow \emptyset$
3:     $n_{i,0}(h) \leftarrow 0$
4:     for $a \in A_i$ do
5:         $n_{i,0}(ha) \leftarrow 0$
6:         $Q_{i,0}(h, a) \leftarrow 0$
```

Chapter 4

Experiments

In this chapter, we start by describing about the runtime optimizations employed in this research. Then, we introduce the wildfire domain and different setups used for running the experiments. Next, we compare the results of the proposed framework with several baselines. Then, we show how the cost of communication impacts overall communication. Finally, we describe the runtime optimizations employed in this research along with the server-client architecture and compare the runtime results of sequential planning and planning using server-client architecture.

4.1 Runtime Optimizations

As mentioned in earlier chapters, the server-client architecture is used to run the level1 models to save computational time. In general, the server acts like a hub coordinating between the clients. Though the server can perform basic calculations, it divides the most computationally extensive tasks among its clients. Figure 4.1 illustrates the basic server-client architecture with n clients. The CIPOMCP is an individual planning framework where the planning for agents is done sequentially at every time step. As agent planning is the most time-consuming process, any optimizations implemented during the planning process save time.

In the CIPOMCP framework, the decision planning can be executed in parallel since the agents plan individually once they receive the observations from the environment. So, if all agents plan in parallel at the same time, maximum time efficiency is achieved. The use of server-client architecture allows reducing the planning time significantly. At each timestep, the server simulates the environmental changes before sending the information to its clients. Then, each client performs the decision planning for the agents it has been assigned and sends the corresponding actions and messages to the server. The server then forms a joint action, which is used to simulate the environment, and the planning procedure continues to the next time step.

Also, the server-client architecture works best if the number of agents and the number of clients is the same. This way, multi-agent decision planning for each time step is completed at the same time as the sequential planning process completes planning for one agent. In the best case, the overall computational time is reduced in the number of agents. In addition, the server-client architecture scales well with the number of agents.

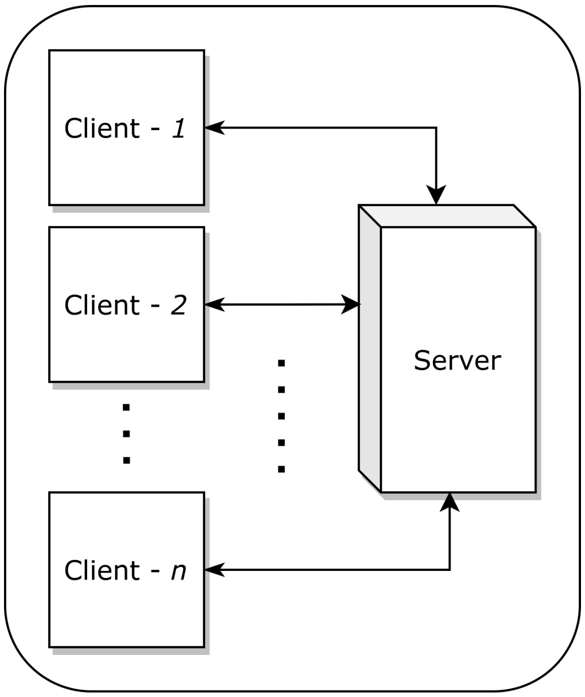


Figure 4.1: A server client architecture

4.2 Domain Description

In the wildfire domain (Chandrasekaran et al., 2016), an agent can have three suppressant levels - Empty, Half Suppressant, and Full Suppressant. The agents start with different suppressant levels based on a probability function. The agents can only fight the fires which are present adjacent to their location. The agents will leave the system to refill their suppressants, and it takes an average of two-time steps to complete the refill. The fires in the wildfire have four possible intensities starting from 0 (put out) to 4 (burned out). The agents cannot fight a burned-out fire to reduce its intensity. The putout fires can re-emerge again in the system with some probability if any fire is present in adjacent locations. The fires in the domain have three sizes-small, medium, and large. As the size increase, the firepower needed to put out the corresponding fire also increases, and so does the reward. The agents' rewards for putting out small, medium, and large fires are 20, 50, and 125, respectively. The agents get a cost of -1 for the burnout of the fires. The agents receive an enormous cost if they opt for an illegal action. If the firepower needed to put out a fire is not met, i.e., not enough agents are fighting that fire, and hence the fire intensities for that fire will not decrease. If the firepower needed is met, there is a probability for the fire intensity to decrease. In addition to that, the probability slightly increases with every added agent once the firepower is met.

As explained earlier, the agents can communicate one of these four messages - {no message, zero suppressants, half suppressant and, full suppressant}. Also, the communication in the domain can be costly, i.e., agents receive a cost if they decide to communicate.

Different reasoning methods, including the baselines, are used to compare the performance the

1. NOOP Reasoning: All the agents do a No-Operation(NOOP)

2. Heuristic Reasoning: The agents act at random if they are present in the system.

3. Coordination Reasoning: A coordination agent can be treated as a greedy heuristic agent that keeps track of the fire intensities at the previous step. It starts as a heuristic agent and remains heuristic if the fire intensities do not decrease across the time steps. If the agent sees any fire intensities

decrease, it starts fighting the same fire until it goes down. In the worst-case scenario, the agent with coordination reasoning is as bad as the heuristic agent.

4. POMCP Reasoning: A POMCP agent, while planning for itself, thinks that other agents in the system are heuristic and act at random.

5. IPOMCP-$l1$ Reasoning: A IPOMCP-$l1$ agent is a slightly sophisticated planning agent where it thinks that other agents in the system are POMCP agents.

6. CIPOMCP-$l1$ Reasoning.: A CIPOMCP-$l1$ agent considers others as the CPOMCP agents. It uses communication to influence the behavior of other agents in the system

The simulations in the domain are up to 7-time steps. The agents start with Full Suppressants. The decision planning algorithms - POMCP, IPOMCP-$l1$, and CIPOMCP-$l1$ are planned up to 500 trajectories, ucb constant of 50, a planning horizon of 5, and the particle filter of size 100. As mentioned in section 2.3, the level- 0 policies for POMCP and CPOMCP planners are constructed and used during the planning of IPOMCP-$l1$ and CIPOMCP-$l1$ respectively to save the computational time. In addition, the use of server-client architecture saves computational time. The server handles the domain simulations, whereas each client plans for a single agent during IPOMCP-$l1$ or CIPOMCP-$l1$ planning. For achieving the faster runtimes for IPOMCP-$l1$, CIPOMCP-$l1$ a server-client architecture is used. The computed results are shown in the figures and the tables. The tabulated results contain the average putout counts of the fires per run for different setups. The Figures contain the rewards obtained, suppressants used for respective models, and the impact of the cost of communication on the messages sent between agents, honest and overall total communication. The F-function is designed in such a way that the

- *Full Suppressant* is mapped to fight shared fire with 0.95 probability and 0.05 probability to pick a random action

- *Half Suppressant* is mapped to fight the individual fire with 0.95 probability and 0.05 probability to pick a random action

- *No Message* is mapped to fight a fire at random
- *No Suppressant* is mapped to a NOOP.

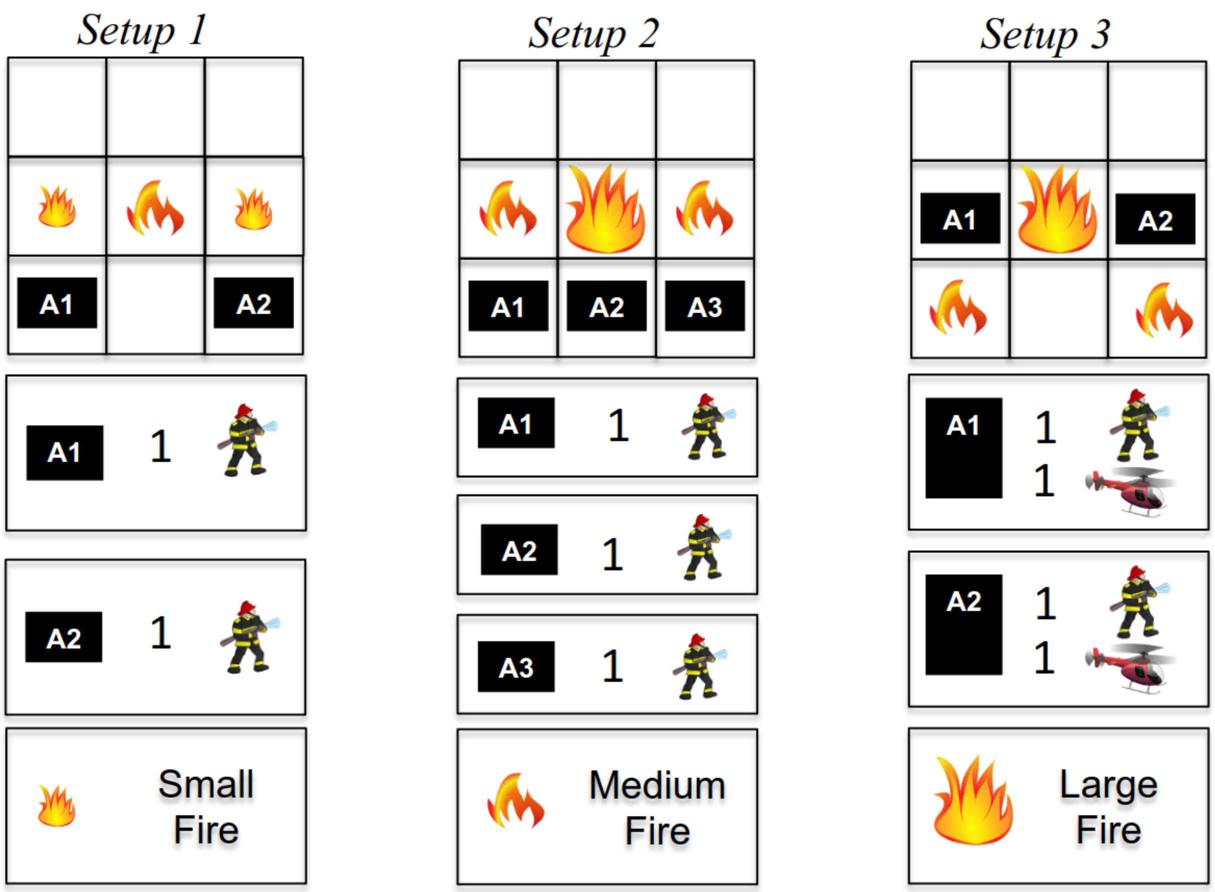


Figure 4.2: Wildfire Setups. Setups 1 and 2 have a single ground fire-fighter at each location. Setup 3 has 1 ground fire fighter and 1 helicopter at each location (Chandrasekaran et al., 2016; Eck et al., 2020).

4.3 Setup 1

There are two small fires and a medium fire present in the setup as in Figure 4.2. In locations A1 and A2, a single ground firefighter agent is present, and the agent has enough fire suppressants to put out the small fire adjacent to its location, whereas both the agents should fight the shared fire.

Table 4.1 consists of the average putout counts of the fires in setup 1 per single run. The NOOP method cannot put out any fires as the agents do not fight any fire. The Heuristic agents pick their actions randomly. Even if a fire's intensity decreases at any time step, the agents do not sense the change and may choose not to fight that fire. However, the Coordination agents sense the change and shift their focus to the respective fire till it is put out completely. Hence the Coordination reasoning has slightly more putout counts. The POMCP reasons that the other agents are random and cannot reason them to stay long enough to put out the shared fire, pushing them to focus their resources on individual fires. The IPOMCP-$l1$ sense the neighbor as the POMCP agents and are likely to reason that the neighbor concentrates on the individual fires, pushing them to focus on the individual fires. The CIPOMCP-$l1$ uses the communication to impact its neighbors. However, the agents will still be likely to put out the individual fire because they can put out these fires on their own, giving them an immediate reward.

Table 4.1: The average putout counts of the fires in setup 1

Method	Setup 1		
	Individual fire at A1 location	Shared Fire	Individual fire at A2 location
Noop	0	0	0
Heuristic	**0.59**	0.2	0.49
Coordination	**0.8**	0.39	0.75
POMCP	1.06	0.3	**1.18**
IPOMCP-$l1$	1.02	0.28	**1.06**
CIPOMCP-$l1$	1.02	0.25	**1.11**

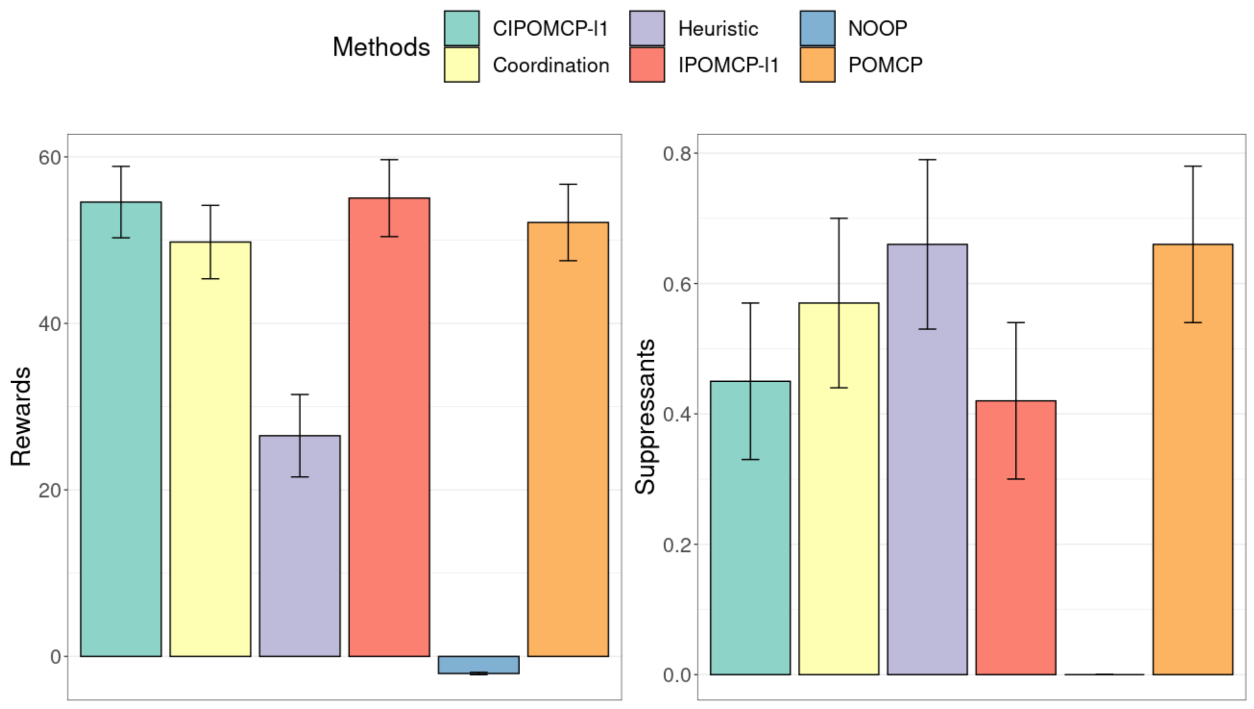


Figure 4.3: Models performance in setup 1: Rewards and Suppressants used.

Figure 4.3 shows the performance of models in setup 1. The NOOP reasoning agent does not use suppressants as it always chooses NOOP, and the fires eventually burn out, resulting in a negative reward. All models except Heuristic and NOOP have same almost the same rewards. From the average putout counts in the Table 4.1, as all the models except Heuristic and NOOP concentrate on putting out the individual fire, they are likely to use similar rewards and use of suppressants. However, the IPOMCP-$l1$ and CIPOMCP -$l1$ use comparatively fewer suppressants for similar rewards.

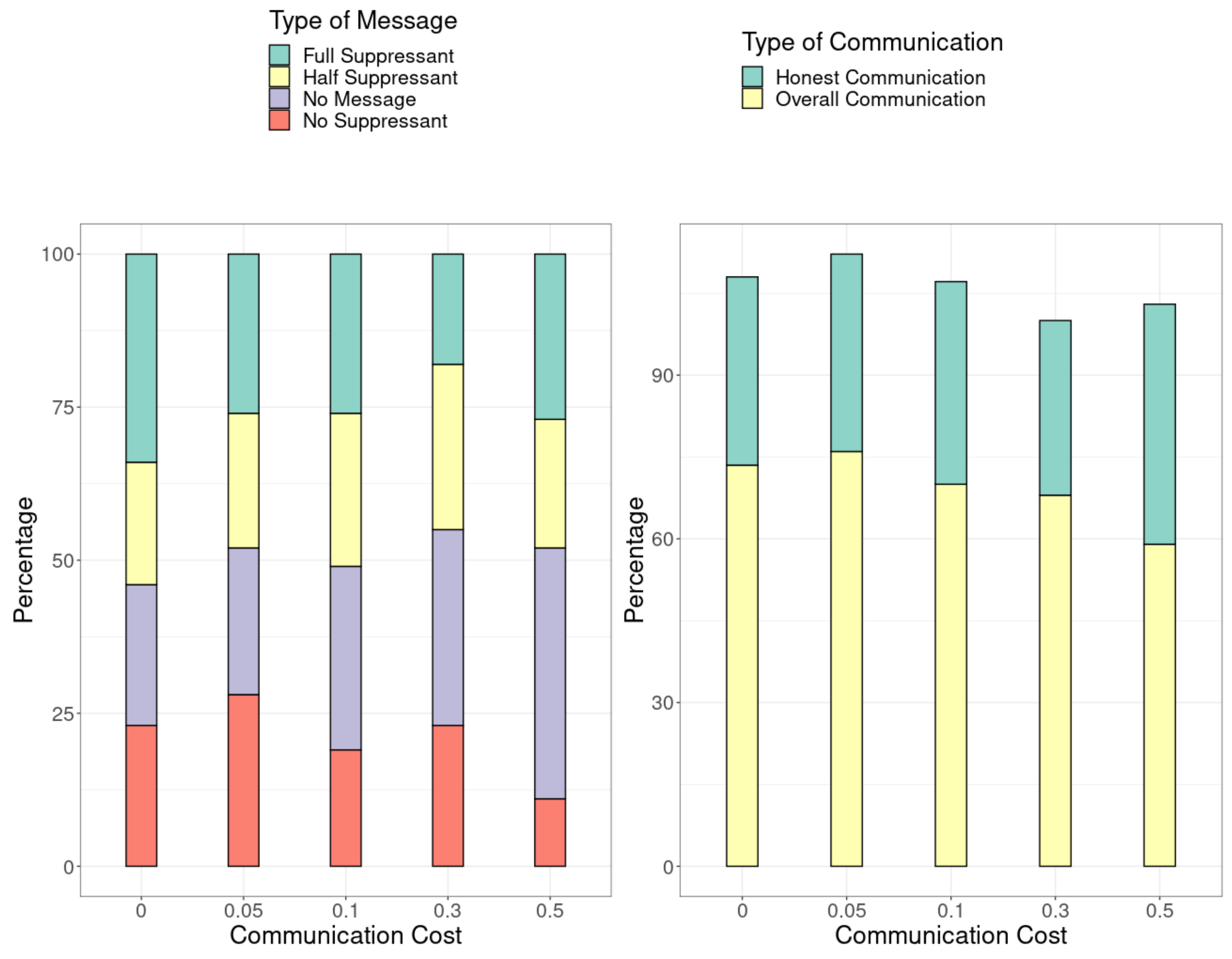


Figure 4.4: The impact of cost on the communication between the agents in Setup 1. a) (Left) Percentages of different messages vs costs of communication, b) (Right) Honest, Overall Communication percentages vs Cost of Communication

Figure 4.4 depicts the communication results. As the communication cost increases, the agents will receive higher negative rewards whenever they chose to communicate. As a result, the percentage of communication decreases as with the rise in communication cost. The *No Message* percentage increases with the rise of communication cost, which proves the trend mentioned above. Though the communication decreases with the rise in cost, the agents start to communicate more honestly, as shown in the Figure 4.4

4.4 Setup 2

There are two medium fires and one large fire in the system. The medium fires require two ground firefighters to fight the fire, whereas all three agents need to fight simultaneously for the large shared fire to be put out. In this setup, every agent needs other agents' help to put out a fire. That is, the agents need to coordinate among themselves to put down the fire. Also, the agent at location A2 has four actions (three fires and a NOOP) that it can take. The firefighting in setup 2 depends on the actions of the agent at A2. If the A2 agent fights the individual fire, the individual fires are likely to go down in intensity. If the A2 agent fights the shared fire, both the other agents need to fight in order to put it out.

Table 4.2: The average putout counts of the fires in setup 2

Method	Setup 2		
	Individual fire at A1 location	Shared Fire	Individual fire at A2 location
Noop	0	0	0
Heuristic	0.1	0.05	**0.14**
Coordination	0.29	0.12	**0.32**
POMCP	**0.6**	0.02	0.49
IPOMCP-$l1$	**0.63**	0.2	0.45
CIPOMCP-$l1$	0.21	**0.72**	0.06

Table 4.2 consists of the average putout counts of the fires in setup 1 per single run. Similar to setup 1, The NOOP method cannot put out any fires as the agents do not fight any fire. As the current setup requires coordination between agents to put out any fire, the heuristic reasoning finds it hard to put out any fire as the agents are acting randomly. Coordination reasoning is a modified heuristic method and is able to put out some more fires than the Heuristic reasoning. The IPOMCP-$l1$ sense the neighbor as the POMCP agents and are likely to reason that the neighbor concentrates on the individual fires, pushing them to focus on the individual fires. The CIPOMCP-$l1$ uses the communication to impact its neighbors. In this setup, the CIPOMCP-$l1$ focuses its resources or suppressants to fight and put out the shared fire.

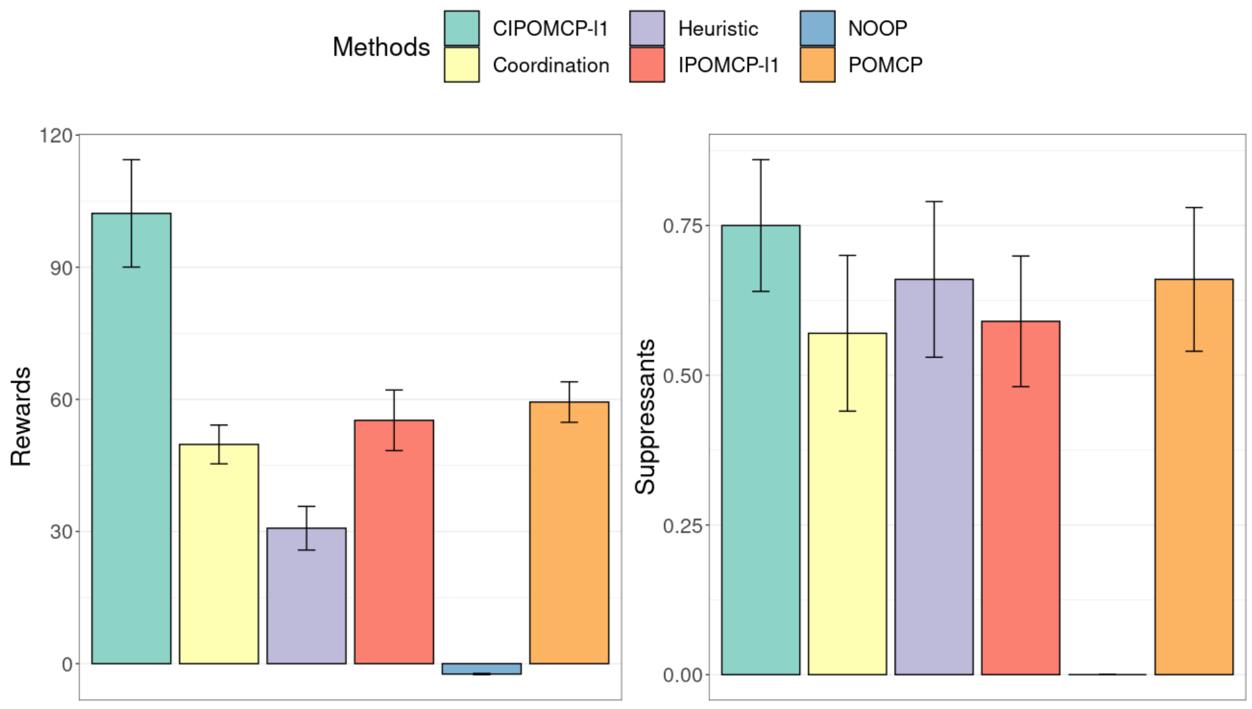


Figure 4.5: Models performance in setup 2: Rewards and Suppressants used

Figure 4.5 shows the performance of models in setup 1. The Coordination, Heuristic, and POMCP have same almost the same rewards. From the average putout counts in the section 4.4, as all the models on putting out the individual fire, they are likely to use similar rewards and use of suppressants. Besides, the CIPOMCP-$l1$ using almost as many suppressants as the other models receives a very high reward as it puts out the shared fire lot more.

Figure 4.6 depicts the communication results. Similar to setup 1, as the communication cost increases, the agents will receive higher negative rewards whenever they chose to communicate. As a result, the percentage of communication decreases as with the rise in communication cost. The *No Message* percentage increases with the rise of communication cost, which proves the trend mentioned above. Unlike setup 1, the percentage of *Full Suppressant* is high when compared to the other messages. As a result, the level-0 CPOMCP predicts that the neighbors are going to fight the shared fire based on the f-function. So the

CPOMCP agent focuses its resources on shared fire more often than not. This makes the CIPOMCP-$l1$ agent also fight the shared fire. Though the communication decreases with the rise in cost, the agents start to communicate more honestly, as shown in the Figure 4.6

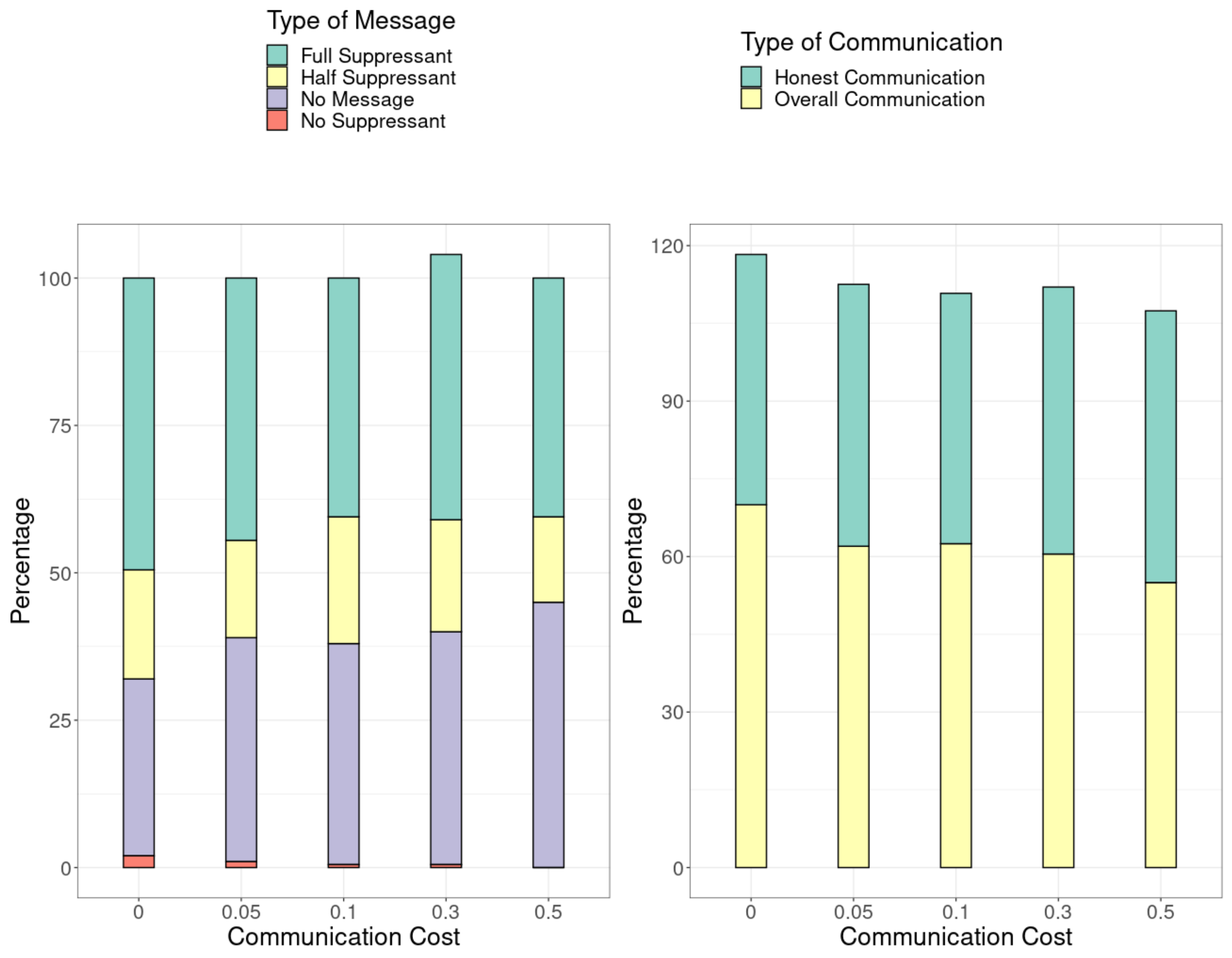


Figure 4.6: The impact of cost on the communication between the agents in Setup 2. a)(Left) Percentages of different messages vs costs of communication, b)(Right) Honest, Overall Communication percentages vs Cost of Communication.

4.5 Setup 3

This setup is similar to the previous one as it needs coordination between agents to put down any fire. This setup location contains a ground firefighter and a helicopter at each location. Though the helicopter is twice as potent as a ground firefighter, it should fight along with the ground firefighter to put down the medium individual fire. A shared fire only goes down when the two helicopters and a ground firefighter are fighting it.

Table 4.3: The average putout counts of a fire in setup 3

Method	Setup 3		
	Individual fire at A1 location	Shared Fire	Individual fire at A2 location
Noop	0	0	0
Heuristic	0.12	0.04	**0.15**
Coordination	0.42	0.37	**0.46**
POMCP	0.54	0.16	**0.61**
IPOMCP-$l1$	0.63	0.03	**0.64**
CIPOMCP-$l1$	0.23	**0.58**	0.23

Table 4.3 consists of the average putout counts of the fires in setup 3 per single run. Similar to setup 1, the NOOP method and heuristic method is unable to put out the fires. Coordination reasoning is a modified heuristic method and is able to put out some more fires than the Heuristic reasoning. Unlike in setup 202, the coordination algorithm benefits from the concentration of agents at two locations and the reduced action state space of these agents as the chance of fire going down increases with agents acting randomly. The coordination algorithm successfully exploits this feature as it is the second-best technique after CIPOMCP. Besides, this setup increases the complexity of coordinating further, pushing the POMCP and inturn IPOMCP agents to focus on putting down the individual fire. Similar to setup 202, the CIPOMCP - $l1$ uses communication to coordinate between its agents and shift their focus to the shared fire.

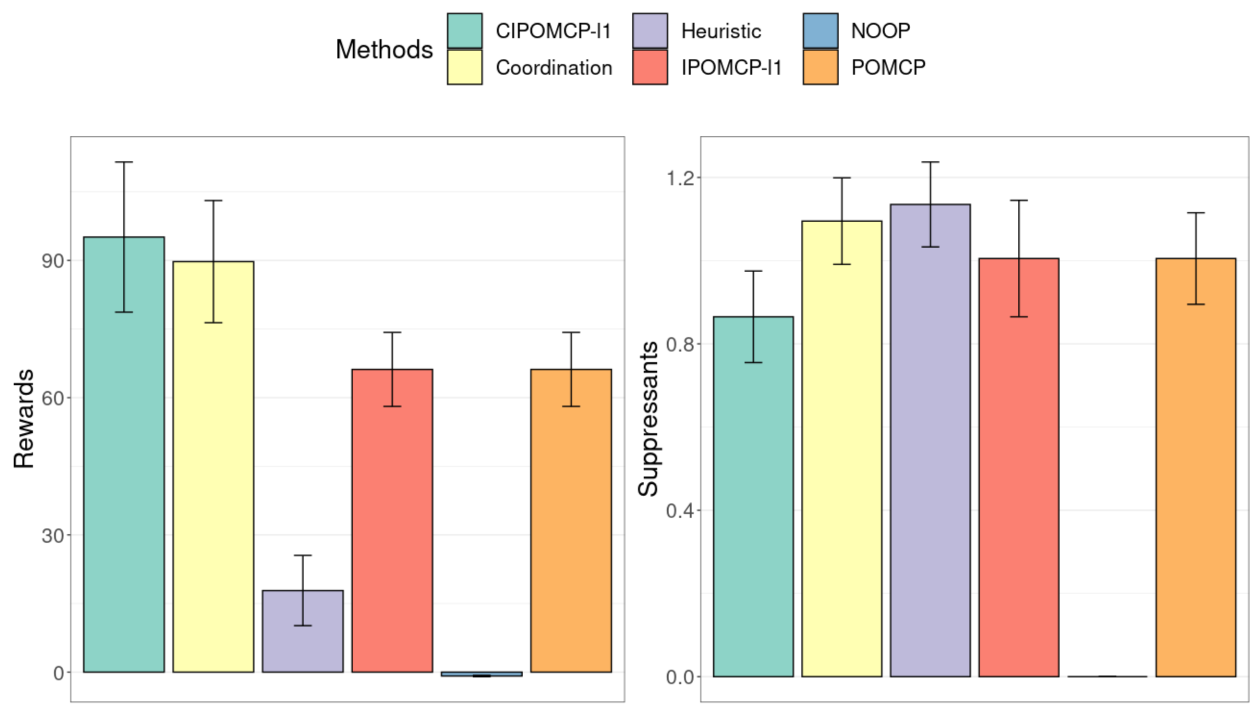
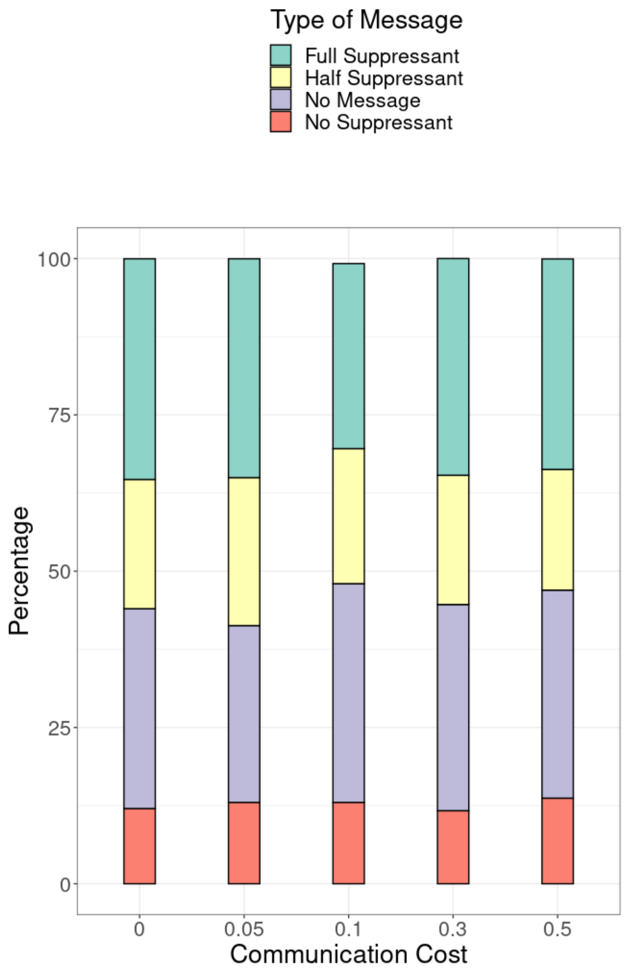


Figure 4.7: Models performance in setup 3: Rewards and Suppressants used

Figure 4.7 shows the performance of models in setup 3. The IPOMCP-$l1$ has almost the same rewards as the POMCP because the former depends on the decisions made by POMCP agents. As a result, almost the same actions or fires are picked most of the time resulting in the same reward. In this setup, the coordination reasoning benefits due to the decrease in the action space of agents. As a result, it has almost as much reward as the CIPOMCP-$l1$ even though it uses a lot more suppressants than the CIPOMCP-$l1$. With its focus on the shared fire, the CIPOMCP-$l1$ reasoning obtains the highest reward with minimum least use of suppressants.

Figure 4.8 depicts the communication results. The agents mostly communicated *Full Suppressant* and *No Message*. As a result, the CPOMCP and inturn CIPOMCP-$l1$ tries to fight the shared fire due to the f-function.



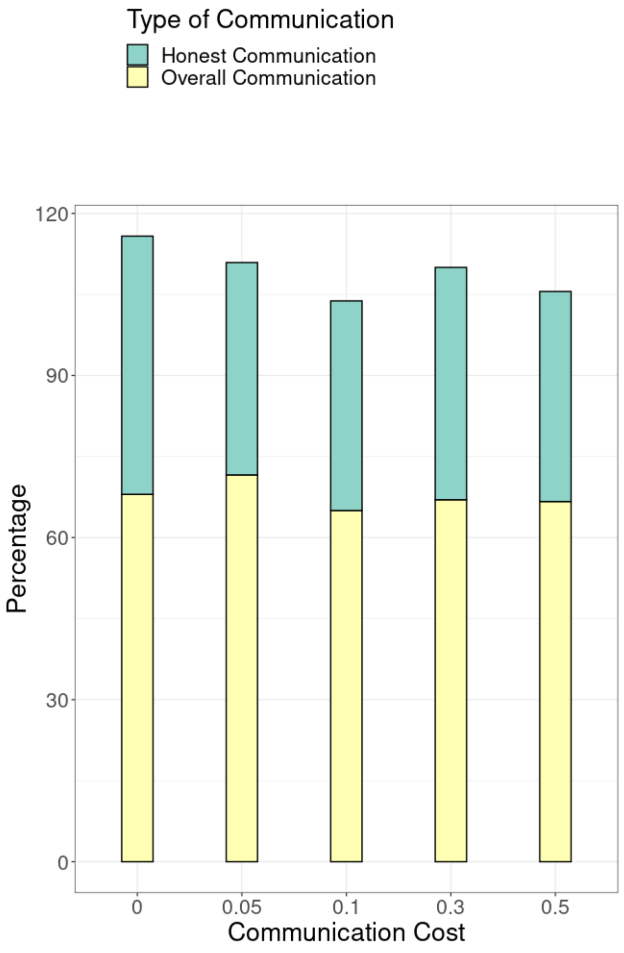


Figure 4.8: The impact of cost on the communication between the agents in Setup 3. a)(Left) Percentages of different messages vs cost of communication, b)(Right) Honest, Overall Communication percentages vs Cost of Communication.

4.6 Runtime Results

Table 4.4 shows the average runtime(in seconds) results for the IPOMCP-$l1$ and CIPOMCP-$l1$ models for different setups and trajectories when run on the server-client architecture and run sequentially. The server-client architecture runs faster than the sequential planning. As the agents or planning parameters increase, the sequential planning takes a lot of time to complete a run, whereas the parallel planning using server-client architecture can complete the run in comparatively less time.

Table 4.4: Average runtime results of IPOMCP and CIPOMCP for different trajectories

Setup	Model	Trajectories-500		Trajectories-1000	
		Server-Client (s)	Sequential (s)	Server-Client (s)	Sequential (s)
1	IPOMCP-$l1$	11.17	15.95	19.99	34.35
	CIPOMCP-$l1$	30.45	41.43	58.42	77.63
2	IPOMCP-$l1$	13.38	27.29	25	52.8
	CIPOMCP-$l1$	36.24	75.93	63.86	146.59
3	IPOMCP-$l1$	12.69	29.6	26.57	58.93
	CIPOMCP-$l1$	43.78	108.92	74.23	210.2

Chapter 5

Conclusion

Decision theoretic planning is a major challenge in open and typed multi-agent systems. Though the use of communication has its benefits, individual planning in such systems is not tractable. Our work extends the existing CIPOMDP framework by using the Monte Carlo tree search as the solving technique. To counter the challenges caused by the nested modeling in the framework, we introduce the f, g functions. Besides, we introduce a group of agents called cliques in which every agent models everyone else. Our approach demonstrates the utility of various techniques like frame action anonymity, pre-policy computation of level 0 policies, and server-client architectures in increasing the algorithm's scalability.

Our experiments in the wildfire suppression domain show the reduction in computational time through the use of the above-mentioned strategies. The results from the experiments depict that the agents in CIPOMCP-$l1$ use communication to put out the shared fire thereby, obtaining a high reward. Further, they show the potency of the framework in the settings that require high coordination.

Some limitations and future work for the current approach are:

- The planning suffers from the massive fan out at the possible belief nodes, i.e., observations under an action node in the search tree. Each possible message set from neighbors produces a different observation leading to a different belief node. As the number of possible message sets and, in turn, belief nodes is exponential in the number of agents, the algorithm's scalability is limited. In addition to that, the MCTS planner cannot visit most of the belief nodes during the search in a decent time. When the planner receives the observation from the domain, the corresponding observation with the same message set might not be visited in the tree during planning, leading to an empty belief. Hence the belief update may be improper in-between time steps. Algorithms like PLEASE (Zhang et al., 2015) and α-DESPOT (Garg et al., 2019) have been proposed in the literature to counter the large observation spaces which should be tested in the current framework.

- Though the CIPOMCP-$l1$ agent uses communication to influence neighbors in the system, promoting coordination, it does not expect to receive any messages from its modeled level 0 neighbors, and they do not send messages. As a result, intentional communication in the planning occurs when the agents start to plan at a level greater than one. However, the algorithm's complexity increases exponentially with the planning level, which makes the framework highly intractable to plan at higher levels. Though the proposed framework utilizes the server-client network to parallelize the planning process, it is not enough to scale the planning to level-2 or more. One direction for future work is to explore the effectiveness of recent MCTS parallelization techniques in decreasing the planning time to improve the algorithm's scalability.

Bibliography

Åström, K. J. (1965). Optimal control of markov processes with incomplete state information i. *10*, 174–205. https://doi.org/10.1016/0022-247X(65)90154-X

Bernstein, D. S., Zilberstein, S., & Immerman, N. (2013). The complexity of decentralized control of markov decision processes.

Cassandra, A. R., Kaelbling, L. P., & Littman, M. L. (1994). *Acting optimally in partially observable stochastic domains* (tech. rep.). USA, Brown University.

Chandrasekaran, M., Eck, A., Doshi, P., & Soh, L. (2016). Individual planning in open and typed agent systems. *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence*, 82–91. http://dl.acm.org/citation.cfm?id=3020948.3020958

Eck, A., Shah, M., Doshi, P., & Soh, L. (2020). Scalable decision-theoretic planning in open and typed multiagent systems. *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*.

Garg, N. P., Hsu, D., & Lee, W. S. (2019). Despot-alpha: Online pomdp planning with large state and observation spaces. In A. Bicchi, H. Kress-Gazit, & S. Hutchinson (Eds.), *Robotics: Science and systems xv, university of freiburg, freiburg im breisgau, germany, june 22-26, 2019*. https://doi.org/10.15607/RSS.2019.XV.006

Gmytrasiewicz, P. (2020). How to do things with words: A bayesian approach. *Journal of Artificial Intelligence Research*, *68*, 753–776. https://doi.org/10.1613/jair.1.11951

Gmytrasiewicz, P. J., & Doshi, P. (2005). A framework for sequential planning in multiagent settings. *Journal of Artificial Intelligence Research*, *24*, 49–79.

Hsu, D., Lee, W. S., & Rong, N. (2007). What makes some pomdp problems easy to approximate? *Proceedings of the 20th International Conference on Neural Information Processing Systems*, 689–696.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*(1), 99–134. https://doi.org/https://doi.org/10.1016/S0004-3702(98)00023-X

Katt, S., Oliehoek, F. A., & Amato, C. (2017). Learning in pomdps with monte carlo tree search. *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, 1819–1827.

Lee, J., Kim, G.-h., Poupart, P., & Kim, K.-E. (2018). Monte-carlo tree search for constrained pomdps. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in neural information processing systems*. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2018/file/54c3d58c5efcf59ddeb7486b7061ea5a-Paper.pdf

Oliehoek, F. A., & Amato, C. (2016). *A concise introduction to decentralized pomdps* (1st). Springer Publishing Company, Incorporated.

Russell, S., & Norvig, P. (2010). *Artificial intelligence: A modern approach* (3rd ed.). Prentice Hall.

Silver, D., & Veness, J. (2010). Monte-carlo planning in large pomdps. *Proceedings of the 23rd International Conference on Neural Information Processing Systems - Volume 2*, 2164–2172.

Somani, A., Ye, N., Hsu, D., & Lee, W. S. (2013). Despot: Online pomdp planning with regularization. *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, 1772–1780. http://dl.acm.org/citation.cfm?id=2999792.2999810

Sonu, E., Chen, Y., & Doshi, P. (2017). Decision-theoretic planning under anonymity in agent populations. *J. Artif. Int. Res.*, *59*(1), 725–770. http://dl.acm.org/citation.cfm?id=3176788.3176805

Sunberg, Z., & Kochenderfer, M. J. (2017). POMCPOW: an online algorithm for pomdps with continuous state, action, and observation spaces. *CoRR*, *abs/1709.06196*. http://arxiv.org/abs/1709.06196

Thomas, V., Hutin, G., & Buffet, O. (2021). Monte carlo information-oriented planning. *CoRR*, *abs/2103.11345*. https://arxiv.org/abs/2103.11345

Zhang, Z., Hsu, D., Lee, W. S., Lim, Z., & Bai, A. (2015). Please: Palm leaf search for pomdps with large observation spaces. *SOCS*.