KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Reputation-driven Decision-making in Networks of Stochastic Agents

David Maoujoud

Thesis submitted for the degree of Master of Science in Engineering: Computer Science, option Artificial Intelligence

Thesis supervisors:
Prof. dr. L. De Raedt
Dr. G. Rens

Assessors:
Prof. dr. M. Denecker
Dr. F. Yang

Mentor:
Dr. G. Rens

Academic year 2019–2020

Contents

Abstract

This thesis studies multi-agent systems that involve networks of self-interested agents. In 2018, *Rens et al.* [1] developed a Markov Decision Process-derived framework, called RepNet-POMDP, tailored to domains in which agent reputation is a key driver of the interactions between agents. The theoretical foundation of the framework was provided; the framework itself was, however, subsequently left unimplemented.
Due to the highly intractable nature of its exact planning algorithm, we first reduce the framework to its fully observable equivalent, called RepNet-MDP, in an effort to study the framework's properties more efficiently. We further alleviate the intractability of the framework by devising an algorithm for finding approximate solutions. We show that the concept of *directed actions*, as introduced by *Rens et al.*, is subject to several theoretical inconsistencies, and propose an alternative interpretation of this concept that retains its core characteristics. We furthermore demonstrate that the initial notion of *action distribution* can lead to undesirable agent behavior, and provide a revised formulation of the concept.
The viability of the framework is tested in a series of experiments designed to highlight its strengths and shortcomings. The tests display the RepNet agents' ability to leverage the framework's fundamental properties in an effort to adapt their behavior to the past behavior and reliability of the remaining agents of the network. RepNet agents are furthermore shown to be willing to sacrifice their selfish intentions in an attempt to maintain a general level of well-being of the entire network. Finally, our work identifies a limitation of the framework in its current formulation that prevents its agents from learning in circumstances in which they are not a primary actor.

List of Figures and Tables

List of Figures

List of Tables

List of Abbreviations and Symbols

Abbreviations

MAS	Multi-agent System
MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process
SARL	Single-agent Reinforcement Learning
SAL	Single-agent Learning
MARL	Multi-agent Reinforcement Learning
MAL	Multi-agent Learning
RL	Reinforcement Learning
VI	Value iteration
eVTOL	Electric Vertical Take-off and Landing

Symbols

e'	Value of e at the following time-step
$\mathcal{E}$	Finite set of elements
$\lvert\mathcal{E}\rvert$	Cardinality of $\mathcal{E}$
$\Delta(\mathcal{E})$	Set of probability distributions over the elements of $\mathcal{E}$
$\sum_e$	Shorthand notation for $\sum_{e\in\mathcal{E}}$
$\{X_g\}$	Shorthand notation for $\{X_g \mid g \in \mathcal{G}\}$
$\mathcal{A} \oplus \mathcal{B}$	Cross-sum of vectors $\mathcal{A}$ and $\mathcal{B}$, defined as $\{a + b \mid a \in \mathcal{A} \wedge b \in \mathcal{B}\}$
α	Learning rate
γ	Discount factor
ϵ	Exploration-exploitation trade-off parameter
η	Laplace smoothing parameter
D	Look-ahead depth

Chapter 1

Introduction

1.1 Context

Markov Decision Processes (MDPs) form a mathematical framework for single-agent decision-making in stochastic environments [2]. They are characterized by the possible states of an environment of interest, the actions this environment is subject to, its dynamics, and a reward scheme. Solving an MDP yields a policy that instructs the agent on how to behave in each situation. One could, by way of illustration, use this formalism to provide an agent of interest, say, a robot to be deployed in a maze (i.e., the environment), with an optimal way of finding its way out (i.e., a policy). The robot changes the state of the environment by applying actions to it, that is, by moving about in the maze. In addition, one may describe the robot's movements as unreliable, thereby adding an element of stochasticity to the problem; the robot could, for instance, occasionally move further than anticipated. The interactions between the agent and the environment shape the dynamics of the problem.

Several extensions of the original framework have been proposed to accommodate for the presence of multiple agents. Some of them, most notably Multi-agent MDPs (MMDPs) [3] and Decentralized Partially Observable MDPs (Dec-POMDPs) [4, 5], operate under the assumption that the agents are selfless and have a common goal. Others, such as Interactive-POMDPs (I-POMDPs) [6], make no such assumptions, and are designed for each agent to adapt its behavior to its self-centered peers'.

A primary concern when dealing with self-centered agents is that it makes multi-agent learning inherently more complex than single-agent learning [7, 8]. This is, to a large extent, due to the fact that each agent needs to take into account the behavior of the entire network of agents when learning its own behavior. Additionally, agent behavior tends to be ever-changing. This *non-stationarity* of agent behavior leads to the loss of policy *convergence* properties that can often be found in single-agent formalisms [8].

The paper "*Maximizing Expected Impact in an Agent Reputation Network*" published in 2018 by Rens et al. [1] proposes a mathematical framework called *RepNet-POMDP* designed to handle domains in which an agent's reputation among other agents dictates its behavior. The framework builds on top of the MDP framework,

by explicitly modeling agent behavior and reputation, as well as introducing partial observability of the environment and other agents' behavior.

Understanding decision-making in the face of selfishness plays an important role in several domains and is notably the subject of *game theory* [9]. A simple trading scenario can serve as an illustrative example. Let two agents constitute the two parties involved in a trade transaction. The first agent, called the buyer, wishes to buy a good offered by the second agent, called the seller. The seller is at no point forced to accept the trade offer: in fact, if the buyer is known to have a poor reputation, the seller might prefer to refuse the trade offer to steer clear of any interactions with the buyer. The reputation of the buyer shapes the dynamics of the interactions between both agents.

A second and more complex example in which an agent's reputation plays a key role revolves around the topic of Electric Vertical Take-off and Landing aircraft [10]. In this illustrative example, these pilotless aircraft make up a network of agents that require air traffic management. The example furthermore involves a set of vertiports that the aircraft can land on to recharge their batteries. The aircraft are to carry passengers from one vertiport to another. Say a first air-taxi is currently charging its battery while a second one is in the air waiting for the vertiport to free up. As the second aircraft's battery reaches lower levels, it must either fly to another nearby vertiport or trust the first aircraft to leave. This decision is to be made based on the first aircraft's past behavior and reputation.

The RepNet framework is tailored to problems alike and constitutes the subject of the present thesis. In their paper, *Rens et al.* provide the theoretical foundation of the framework, which introduces several concepts that distinguish it from other MDP-derived frameworks. As such, the notions of *action distribution*, *image*, *reputation*, and *directed actions* are established as key components of the framework, and are used by RepNet agents to navigate through the environment and interact with the other agents that are apart of the environment. While these concepts were developed on a theoretical level, the framework itself was not evaluated empirically.

Problem statement: *The primary goal of the present thesis is to study the properties of the RepNet framework, and evaluate its viability as an MDP-derived, multi-agent framework that deals with agent selfishness. Finding exact solutions for RepNet-POMDPs is, however, known to be computationally intractable. The intractability of the framework is to be alleviated before the framework and its properties can be tested.*

The study of the framework's properties can be further refined into four research questions, which will be answered in this thesis.

Research question 1: *Are the concepts of action distribution, image, and reputation effective in practice?*

Research question 2: *Is the concept of directed actions effective in practice?*

Research question 3: *Are RepNet agents capable of drawing general conclusions on the interactions between the agents that are apart of the environment?*

Research question 4: *Are RepNet agents capable of being cooperative in spite of their selfish intentions?*

Broadly speaking, the work carried out in regards to the RepNet framework involves a first part of *theoretical*, and a second part of *experimental*, nature.

Theoretical part: The approach taken in this thesis to alleviate the problem of intractability can be summarized as a two-step procedure: the first step consists in reducing the framework to a fully observable setting called RepNet-MDP. The second step consists in favoring approximate solutions to the reduced framework of satisfactory quality over exact solutions. An algorithm for finding approximate solutions will be presented to this end.

Experimental part: After reducing the framework's intractability, the viability of said framework will be evaluated in a series of simplified scenarios drawn from the *trading* and *air-taxi* domains. Each scenario will be tailored to showcase the strengths and shortcomings of the framework.

1.2 Outline

This thesis is structured in the following way: Chapter 2 summarizes the relevant background required. Chapter 3 provides an overview of the related work and, in particular, the framework proposed by *Rens et al.* [1]. Chapter 4 then discusses the reduction of the starting framework to a fully observable setting. The experimental setup of the reduced framework is given in Chapter 5. Chapter 6 provides the experimental results and discusses the strengths and shortcomings of the RepNet framework. Finally, the work of the previous chapters is brought together and summarized in Chapter 7.

Chapter 2

Background

The present chapter covers background information referred to throughout the thesis. A brief introduction to the *Reinforcement Learning Problem* is first given in Section 2.1. Two Single-agent Markov Decision Process frameworks are then introduced in Section 2.2. The time complexity of exact solutions to these problems is analyzed in Section 2.3. An overview of several exact and approximate solving techniques is given in Section 2.4. Finally, Section 2.5 briefly discusses *first-order* representations of Markov Decision Processes.

2.1 The Reinforcement Learning Problem

Reinforcement Learning (RL) is a Machine Learning paradigm, in which a learning entity, typically called agent, learns in an environment by performing actions and observing the environment's response [11]. More specifically, when an agent performs action a_t at time t, the environment, initially in state s_t, returns its new state s_{t+1} as well as a reward r_{t+1} to the agent. The agent aims to maximize the reward it can accumulate over time. The general feedback loop is schematized in Figure 2.1.

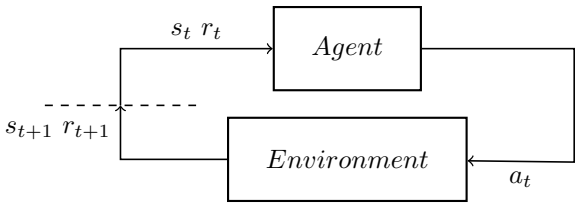


Figure 2.1: Reinforcement Learning feedback loop

The RL paradigm can be formalized in Single-agent and Multi-agent settings. Several properties emerging from the former no longer hold in the latter. The fundamental single-agent formalisms are now covered in further detail.

2.2 Single-agent formalisms

This section formalizes the fundamental concepts of learning in single-agent settings. In the present thesis, the following properties of the environment are assumed to be true:

- The environment behaves stochastically, meaning that given a starting state and an action performed, the environment is not guaranteed to transition to the same new state each time. A robot expected to jump forward one meter might, for instance, miss the target and jump too far.
- The environment is stationary, meaning that the transition rules which the environment abides by do not change over time.
- The environment obeys the *Markov property*. This property states that the probability of the environment transitioning to any given state is solely conditioned by the current state of the environment and the current action of the agent [2].
- The state space is discrete and comprised of a finite number of elements.

Section 2.2.1 covers the simplest form of Markov Decision Processes. Section 2.2.2 then introduces a common extension to the initial framework.

2.2.1 Fully Observable Markov Decision Processes

The Single-Agent Learning (SAL) problem can be formally modeled as a Markov Decision Process.

Definition 2.1 (Fully Observable MDP)**.** *A fully observable Markov Decision Process, or simply MDP, $\mathcal{M}$ is formally defined as a tuple [2]*

$$\mathcal{M} := \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$$

where:

- $\mathcal{S}$ *is the set of possible states of the environment.*
- $\mathcal{A}$ *is the set of possible actions.*
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ *is called the transition model.* $\mathcal{T}(s, a, s')$ *returns the probability of the environment transitioning from state s to state s' after action a is performed.*
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ *is called the reward function.* $\mathcal{R}(s, a)$ *returns the immediate reward received after performing action a in state s.*

The objective of an MDP agent is to maximize its long-term cumulative reward, called *utility*. While the utility can be defined in several ways, the most commonly used definition is called *discounted future reward*. The utility U of a *finite* state-action sequence, sometimes called *episode*, $E = \langle s_0, a_0, s_1, a_1, ..., s_T, a_T \rangle$ is defined as [12]:

$$U(E) = \sum_{t=0}^{T} \gamma^t \mathcal{R}(s_t, a_t) \tag{2.1}$$

where $\gamma \in [0, 1]$ is called the *discount factor*. An agent moves about in the environment by following a *policy*.

Definition 2.2 (Finite-horizon policy)**.** *A finite-horizon policy $\pi : \mathcal{S} \times \mathbb{N} \to \mathcal{A}$ is a function that maps each environment state and remaining time-steps to the action the agent should take.*

The *expected utility*, or *value*, of being in any state s_t at time-step t, while following policy π, with d time-steps remaining, is defined as [12]:

$$V^{\pi}(s_t, d) = \mathbb{E}[U(E_t) \,|\, s_t, \pi] = \mathbb{E}[\sum_{k=t}^{t+d} \gamma^{k-t} \mathcal{R}(s_k, a_k) \,|\, s_t, \pi] \tag{2.2}$$

where E_t is the sub-sequence of E starting at time-step t. An optimal policy $\pi^\star$ is a policy such that

$$\forall s \in \mathcal{S}, \forall d \in \mathbb{N}, \forall \pi : V^\star(s, d) \geq V^{\pi}(s, d) \tag{2.3}$$

where $V^\star : \mathcal{S} \times \mathbb{N} \to \mathbb{R}$ is the value function associated with optimal policy $\pi^\star$. This policy satisfies the *optimality equations*, also known as the *Bellman equations*:

$$\begin{cases} V^\star(s, d) := \max\limits_{a \in \mathcal{A}} \Big\{ \mathcal{R}(s, a) + \gamma \sum\limits_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') V^\star(s', d-1) \Big\} & \forall s \in \mathcal{S}, d > 1 \\ V^\star(s, 1) := \max\limits_{a \in \mathcal{A}} \Big\{ \mathcal{R}(s, a) \Big\} & \forall s \in \mathcal{S} \end{cases} \tag{2.4}$$

MDPs are adequate for modeling a range of stochastic domains in which the state of the environment is fully observable at all times. Yet, fully observable environments can not always be taken for granted. The problem of partial observability is covered hereafter.

2.2.2 Partially Observable Markov Decision Processes

Alongside the stochastic nature of many environments, a prevalent form of uncertainty lies in the partial observability of several domains. Take a robot equipped with proximity sensors advancing in a room filled with obstacles. This robot does not sense the actual state of the room. Rather, it can only sense what the sensors pick up, and may very well bump into obstacles that were missed during the sensing process. Partially Observable Markov Decision Processes are an extension of classic MDPs that deal with the problem of partial observability [2].

Definition 2.3 (Partially Observable MDP)**.** *A partially observable MDP, or POMDP, $\mathcal{P}$ is formally defined as a tuple*

$$\mathcal{P} := \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O} \rangle$$

where:

- *$\mathcal{S}$, $\mathcal{A}$, $\mathcal{T}$ and $\mathcal{R}$ are defined as in Definition* 2.1*.*
- *Ω is the set of observations.*
- *$\mathcal{O} : \mathcal{A} \times \mathcal{S} \times \Omega \to [0, 1]$ is called the observation function or sensor model. $\mathcal{O}(a, s', o)$ returns the probability of making observation o after performing action a and the environment transitioning to state s'.*

Instead of working with the actual states of the environment, the POMDP framework introduces the notion of *belief states*.

Definition 2.4 (Belief state)**.** *A belief state $b \in \Delta(\mathcal{S})$ is a probability distribution over the possible states of the environment. As such, $b(s)$ returns the probability of being in state s. Furthermore,*

$$\sum_{s \in \mathcal{S}} b(s) = 1$$

Suppose the agent makes observation o after taking action a in current belief state b. The updated belief state b' is computed using the *state estimation function* SE defined as follows:

$$b' := SE(b, a, o) := \Big\{ (s', p) \;\Big|\; s' \in \mathcal{S} \wedge p = \frac{\mathcal{O}(a, s', o) \sum_s \mathcal{T}(s, a, s') b(s)}{P(o|b, a)} \Big\} \tag{2.5}$$

where $P(o|b, a) = \sum_{s' \in \mathcal{S}} \mathcal{O}(a, s', o) \sum_{s \in \mathcal{S}} \mathcal{T}(s, a, s') b(s)$ is a normalizing constant. The optimal value function $V^\star : \Delta(\mathcal{S}) \times \mathbb{N} \to \mathbb{R}$ satisfies the following *optimality equations* $(\forall b \in \Delta(\mathcal{S}))$:

$$\begin{cases} V^\star(b, d) := \max\limits_{a \in \mathcal{A}} \Big\{ \sum\limits_{s \in \mathcal{S}} \mathcal{R}(s, a) b(s) + \gamma \sum\limits_{o \in \Omega} P(o|b, a) V^\star(SE(b, a, o), d - 1) \Big\} \\ V^\star(b, 1) := \max\limits_{a \in \mathcal{A}} \Big\{ \sum\limits_{s \in \mathcal{S}} \mathcal{R}(s, a) b(s) \Big\} \end{cases} \tag{2.6}$$

2.3 Time complexity of solving Markov Decision Processes via Value Iteration

The present section covers the time complexity of a solving strategy for fully and partially observable MDPs called the value iteration algorithm. Partial observability is shown to render the computation of exact solutions intractable, even in single-agent settings.

Value iteration (VI) is an iterative process that considers successively longer planning horizons to produce exact, i.e. optimal, solutions to fully and partially observable MDPs [13]. This is accomplished by computing the i-th horizon optimal value function at each iteration i, using the $i-1$-th horizon optimal value function computed at the previous iteration. The termination criterion is different for infinite- and finite-horizon problems: in infinite horizon problems, termination is reached when the value function converges, i.e., when it no longer changes sufficiently between two iterations. The final value function is then kept. In finite-horizon problems, termination is reached when the value function has been computed for each step within the horizon depth. The value functions produced at each step are all kept. The fully observable MDP VI algorithm was introduced by *Bellman* in 1957 [14], while *Sondik* developed its counterpart for partially observable environments in the early 1970s [15].

2.3.1 VI for fully observable, finite-horizon MDPs

Let $\mathcal{M}$ be a finite horizon MDP of horizon d, $|\mathcal{S}|$ the number of states, and $|\mathcal{A}|$ the number of actions of $\mathcal{M}$. Each iteration i of the VI planning algorithm computes the optimal value $V^\star(s,i)$ for each state s, i.e. $|\mathcal{S}|$ computations in total. Each such computation involves the maximization over $|\mathcal{A}|$ actions. For each action a, the evaluation involves the summation over $|\mathcal{S}|$ states. The time complexity of the VI algorithm is thus [16]

$$\mathcal{O}(d|\mathcal{S}|^2|\mathcal{A}|). \tag{2.7}$$

The algorithm is given in Algorithm 1.

Algorithm 1 Value iteration for finite-horizon MDPs

```
1: procedure VALUE-ITERATION
2:     for i = 1 → d do
3:         for all s ∈ S do
4:             if i == 1 then
5:                 V*(s, 1) ← max_{a∈A} {R(s, a)}
6:             else
7:                 V*(s, i) ← max_{a∈A} {R(s, a) + γ Σ_{s'∈S} T(s, a, s') V*(s', i − 1)}
```

As such, MDP VI is polynomial in the number of states $|\mathcal{S}|$. The algorithm is said to be *tractable* and belongs to complexity-class **P**.

2.3.2 VI for partially observable, finite-horizon MDPs

VI is less straightforward with POMDPs than with fully observable MDPs. This comes as a direct result of the belief space being continuous, making it impossible to compute $V^\star(b,n)$ for each belief state b the way it is done for fully observable MDPs [2].

The optimal value function $V^\star$ turns out to always be *piecewise linear* and *convex* in the belief space [17]. Formally, this equates to $V^\star$ having the following form:

$$V^\star(b, n) = \max_{\alpha \in \mathcal{V}_n} \sum_{s \in \mathcal{S}} \alpha(s) b(s) = \max_{\alpha \in \mathcal{V}_n} \alpha \cdot b \tag{2.8}$$

where $\mathcal{V}_n = \{\alpha_0, \alpha_1, ..., \alpha_m\}$ is a set of α-vectors, each α-vector maintaining the parameters of an $|\mathcal{S}|$-dimensional hyperplane describing the optimal value function over a region of the belief space [18]. In other words, $V^\star$ is the upper surface of the collection of hyperplanes described by the α-vectors [19]. Each α-vector is associated with a specific conditional plan, that is, the region over which one hyperplane in $\mathcal{V}_n$ dominates all other hyperplanes in $\mathcal{V}_n$ is associated with one optimal action. Each iteration i of the algorithm consists of two steps [13]:

- Generation of potentially useful i-th horizon α-vectors. This step first consists of the generation of $\mathcal{O}(|A||\Omega||\mathcal{V}_{i-1}|)$ projections in total. The algorithm then proceeds to compute $\mathcal{O}(|A||\mathcal{V}_{i-1}|^{|\Omega|})$ cross-sums. Each α-vector is computed in $\mathcal{O}(|S|^2)$ time [18, 20, 21]. The generation step thus has a worst-case complexity

$$\mathcal{O}(|S|^2 |A||\mathcal{V}_{i-1}|^{|\Omega|}) \tag{2.9}$$

- Pruning of fully-dominated α-vectors. This step usually involves the solving of a linear program and does not add to the asymptotic complexity of the algorithm [22].

The corresponding algorithm is given in Algorithm 2. Solving POMDP via VI is exponential in $|\Omega|$ and, as such, is said to be *intractable*. While the notion of *intractability* will be formalized in Chapter 3, it is worth noting at this point that the *intractability* of this single-agent-based algorithm is evocative of the challenges encountered in most multi-agent learning settings.

Algorithm 2 Value iteration for finite-horizon POMDPs

procedure Value-Iteration
  for all $a \in \mathcal{A}$ **do**
    $\mathcal{V}_1^- \leftarrow \alpha^{a,*}(s) = R(s, a)$
  for $i = 2 \rightarrow d$ **do**
    for all $a \in \mathcal{A}$ **do**
      $\mathcal{V}_i^{a,*} \leftarrow \alpha^{a,*}(s) = \mathcal{R}(s, a)$
      for all $o \in \Omega$ **do**
        for all $\alpha_j \in \mathcal{V}_{i-1}^-$ **do**
          $\mathcal{V}_i^{a,o} \leftarrow \alpha_j^{a,o}(s) = \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') \mathcal{O}(a, s', o) \alpha_j(s')$
    for all $a \in \mathcal{A}$ **do**
      $\mathcal{V}_i^a = \mathcal{V}_i^{a,*} \oplus \mathcal{V}_i^{a,o_1} \oplus ... \oplus \mathcal{V}_i^{a,o_{|\Omega|}}$
    $\mathcal{V}_i = \bigcup_{a \in \mathcal{A}} \mathcal{V}_i^a$
    $\mathcal{V}_i^- = \text{PRUNE}(\mathcal{V}_i)$

2.4 Model-based solving techniques

Various algorithms have followed since the introduction of VI. The present section first provides an overview of the different algorithm classes, after which several solving techniques are described in further detail.

2.4.1 Online and offline learning algorithms

The single-agent (and later multi-agent) frameworks, and more specifically learning algorithms, considered in this thesis can be divided into two categories:

- *Offline learning algorithms*: A learning algorithm is said to be offline if the learning phase takes place entirely prior to the execution of the policy. By the end of the learning phase, a course of action, optimal or not, is known for every possible happening, i.e., state and horizon, and the policy no longer changes during the execution [23]. Offline algorithms are sometimes referred to as traditional search in the literature.

- *Online learning algorithms*: A learning algorithm is said to be online if learning and execution are interleaved, and planning during the learning phase is done using information leveraged during the execution. This often reduces the sum of planning and execution cost, but has the drawback of leading to incomplete policies, as the best course of action is only computed for the information available. Online algorithms are sometimes referred to as agent-centered search [24].

2.4.2 Exact offline methods

In 1960, shortly after Bellman's introduction of the VI algorithm, *Howard* developed an alternative to MDP VI called *policy iteration* [25]. This algorithm proved to have faster convergence properties, but, due to its mathematical formulation, involving the solving of a system of linear equations is usually used with smaller state spaces [26]. In 1963, *d'Epenoux* showed that the Bellman equations could be rewritten as an equivalent linear program [27]. While these methods are used to produce exact optimal solutions, they suffer from the *curse of dimensionality*. In fact, as the number of state variables grows, it becomes increasingly computationally expensive to compute exact policies, as they need to be computed over the entire state space. The problem is even more alarming with POMDPs, due to their associated intractability. As such, in a problem with $|\mathcal{S}|$ states, a POMDP solver has to operate in a $(|\mathcal{S}| - 1)$-dimensional and continuous belief space [18].

2.4.3 Approximate methods

Approximate MDP and POMDP solving methods were introduced to counter the unscalability inherently associated with the exact methods. To accomplish the

alleviation of complexity, several distinct strategies can be identified[1]:

- *Offline* approximations: Several *point-based* approximations have been proposed in the literature. Notably, in 1991, *Lovejoy* employed a grid-based approximation of the POMDP belief-space, to then produce lower and upper value function bounds, using interpolation [28]. Later, *Poon* proposed a grid-based algorithm that computes the gradient alongside the value at each grid point, in an attempt to make the algorithm more generalizable to unexplored belief points, due to the *piecewise linear* and *convex* nature of the exact optimal solution [29]. In 2003, *Pineau et al.* introduced *point-based value iteration* (PBVI) for POMDPs [18], building on top of Poon's work. The novelty in this approach lies in the careful selection of belief points to apply value iteration and gradient updates to. The belief points are selected using *stochastic trajectories*. In short, given a current belief-set $\mathcal{B}$, containing the belief points currently considered for value iteration, a new belief point b is added to $\mathcal{B}$ if it is directly reachable from $\mathcal{B}$ and would improve the uniformity of the density of $\mathcal{B}$ in the set of reachable belief points the most, i.e., b is the furthest away from any point in $\mathcal{B}$. A number of methods have built on top of PBVI since, including HSVI (Smith & Simmons, 2004) [30] and Perseus (Spaan & Vlassis, 2005) [31].

- *Online* approximations: Initial research on approximate online methods was the result of two issues inherently associated with offline approaches: firstly, the time it takes an offline algorithm to solve a (PO)MDP increases rapidly with the number of possible situations, even when carefully selecting the (belief) states to perform calculations on. Secondly, if the transition model used to compute the policy were to change during the execution of the policy, said policy would have to be recomputed from scratch [23].

  The general framework for model-based online planning for MDPs can be described as the interleaving of two phases, the *planning phase*, and the *execution phase*.

  - The planning phase consists of the solving of a state-space search problem. This problem is characterized by the MDP set of states $\mathcal{S}$, its set of actions responsible for the state transitions $\mathcal{A}$, an initial state $s_0 \in \mathcal{S}$, and the value function V giving the value of state transitions [32]. Given the current state of the environment s_0, an AND/OR tree of depth D, specified beforehand, is first constructed. The OR-nodes are state-nodes, while the AND-nodes are action-nodes. The best value is then propagated back to the root of the tree, starting at the leaves and using the Bellman equations (Equation 2.4). More specifically, assuming the leaves of the search tree are not terminal nodes, a starting *heuristic* estimate h of the true value defined as

$$h : \mathcal{S} \to \mathbb{R} \tag{2.10}$$

[1] Note that the algorithms presented hereafter do not form an exhaustive list by any means.

such that $h(s) \leq V(s)\ \forall s \in \mathcal{S}$ (*admissibility* of a heuristic, the estimate of the value must always underestimate the true value [2]) is required. The most straightforward heuristic is the base case of the Bellman equations, as it does not take into account the recursive call of the value function to itself. The action nodes then compute the *state-action* value using the heuristic values computed at the state nodes right below (children). Non-leaf state nodes then use the maximum of the state-action values of their children as their value. Finally, at the root of the tree, the child action associated with the highest state-action value is returned. Figure 2.2 illustrates a look-ahead search of depth $D = 1$.

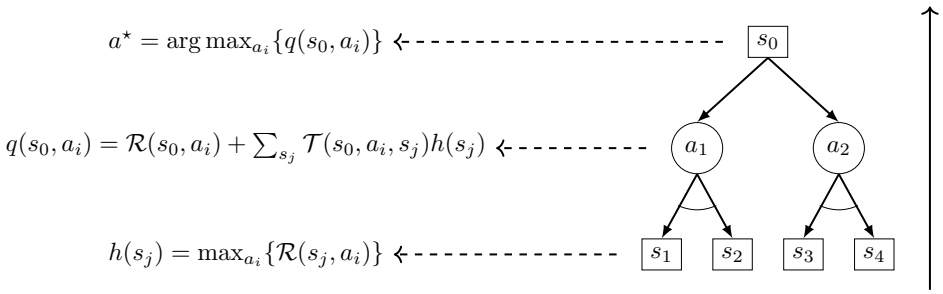


Figure 2.2: Look-ahead search space, depth $D = 1$ (MDP)

– The execution phase consists in executing the action, say $a^\star$, associated with the highest value estimate. The environment returns a new state s' that may now be used as the root of the search tree during the following planning phase.

- *Hybrid Offline-Online* approximations: In 2006, *Parquet et al.* [33] propose three algorithms for POMDPs that combine the advantages of several *online* (RTBSS [34], RTDP-BEL [35]) and *offline* algorithms (Q_{MDP} [36], PBVI [18]), and show that they can often outperform *online* and *offline* approaches alone. In 2011, *Maniloff et al.* [37] introduce *Hybrid Value Iteration* for POMDPs, an algorithm that combines PBVI and look-ahead (tree) search to meet real-time constraints. The heuristic tree search is thereby used to occasionally make improvements to the value function initially computed offline using PBVI.

2.5 Relational representation of the state space

Thus far, states have been handled like elements with no apparent relations between them. Concretely, the state spaces were not described in terms of a set of features that apply to and make up its elements. Consequently, as the environments described become more complex and are made up of more features, the state spaces grow exponentially with the number of features [38], thus quickly leading to unmanageable transition models. It is worth briefly discussing ongoing research in the field of *relational* decision-making.

Many real-world domains are inherently relational [12]. In particular, from a *logic programming* point of view, the state space of an environment can be described by a collection of *relations* that jointly make up the features of the state space [39, 40], and the states are *Herbrand interpretations* of said collection [41]. As such, an environment that describes a blocks world could be made up of the relations `on/2` and `clean/1`, while a possible world, that is, a state could be `on(A,C)`, `clean(B)`, where `A`, `B`, and `C` represent blocks (the example is taken from *Nitti et al.*, 2017 [12]). This state is schematized in Figure 2.3. Actions are then described as facts (e.g., `move(A,B)`), and the state transition model is modeled through probabilistic rules. In 2001, *Boutilier et al.* [39] introduced the first *relational* (*first-order*) MDP (RMDP) and provided an algorithm for solving RMDPs using the *Bellman equations* called *Symbolic Dynamic Programming*. This *structured* approach to the representation of states makes it possible to solve complex (real-world) problems of relational nature, that is, problems in which states are made up of multiple features.

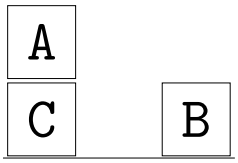


Figure 2.3: A state of the blocks world

Note that, due to the simple nature of the scenarios discussed in later chapters, and given that the use of this representation is not indispensable to the study of the framework of interest of this thesis, we do not adopt a *relational* approach for representing the states of the environment.

Chapter 3

Related work: Multi-agent formalisms

Multi-agent learning (MAL) problems are problems where an agent is to learn to behave optimally in the company of other agents, which may or may not be learning themselves [42]. Multi-agent settings are inherently more complex than their single-agent counterpart. In particular, several useful properties that hold in single-agent settings are no longer valid in multi-agent settings. Nevertheless, MAL techniques are used in practice and generally produce good results [43]. Section 3.1 addresses some of the challenges that arise when moving from a single-agent setting to a multi-agent setting. Sections 3.2, 3.3, and 3.4 then summarize several multi-agent frameworks. Finally, Section 3.5 covers the multi-agent, POMDP-based framework called *Reputation Network* POMDP.

3.1 Challenges in multi-agent formalisms

3.1.1 Non-stationarity

An intuitive and naive way of dealing with the presence of multiple agents from the perspective of one agent g may be to consider every other agent to be a part of the environment. In fact, by assuming that each agent h is following a stationary policy π_h, solving a MAL problem from the perspective of agent g reduces to solving a single-agent problem where the behavior of all other agents is factored in the environment's transition model [44, 45]. This assumption is not always reasonable and, more often than not, the agents will adapt their policy as time goes by [7]. Alongside the impossibility of reducing the problem, *correlated goals* and the *loss of convergence guarantee* discussed hereafter have a significant impact on multi-agent learning.

3.1.2 Correlated goals

In single-agent settings, the rewards collected by the agent are a function of the state of the environment. In multi-agent settings, a first consequence of the non-

stationarity of policies is that one agent's returns will likely depend on the other agents' behavior as well [7]. Explicit modeling of the reward scheme may, therefore, be significantly more complex.

3.1.3 The convergence problem

Another consequence of the non-stationarity of policies is the invalidation of the convergence guarantee of most Single-agent learning (SAL) algorithms [7, 8]. In fact, the convergence of one agent's policy is now a function of all other agents' behavior in the network.

3.1.4 The curse of dimensionality

In complexity theory, a problem is considered to suffer from the curse of dimensionality with respect to a variable n if

$$T(n) = \Omega(2^n), \tag{3.1}$$

where Ω is the lower-bound on complexity, and T is the number of steps required to solve said problem [46]. Such a problem is said to be *intractable*. Multi-agent learning problems are not the only ones subject to the curse of dimensionality: in fact, Section 2.3 already illustrated the intractability of solving classic single-agent POMDPs via *Value Iteration*. The problem worsens considerably when moving to multi-agent frameworks [47].

3.2 Multi-agent MDP: A centralized framework

The Multi-agent MDP (MMDP) was formalized by Boutilier in 1996 [3], and is one of the first and simplest multi-agent learning frameworks.

Definition 3.1 (Multi-agent MDP)**.** *A multi-agent Markov Decision Process, or simply MMDP, $\mathcal{M}$ is formally defined as a tuple*

$$\mathcal{M} := \langle \mathcal{S}, \mathcal{G}, \mathcal{A} = \times_{g \in \mathcal{G}} \mathcal{A}_g, \mathcal{T}, \mathcal{R} \rangle$$

where:

- $\mathcal{S}$ *is the set of possible states of the environment.*
- $\mathcal{G}$ *is the set of agents that can interact with the environment.*
- $\mathcal{A}_g$ *is the set of actions available to agent g.*
- $\mathcal{A}$ *is the set of joint actions.*
- $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times ... \times \mathcal{A}_n \times \mathcal{S} \to [0, 1]$ *is called the transition model.*
- $\mathcal{R} : \mathcal{S} \times \mathcal{A}_1 \times ... \times \mathcal{A}_n \to \mathbb{R}$ *is called the team reward function.*

MMDP agents are assumed to be *fully cooperative* [3], that is, they have no individual motivation. Their sole goal lies in maximizing the global (i.e. shared) expected reward. Each agent is furthermore assumed to have *individual full state observability*, meaning all agents observe the same state. One can, therefore, think of an MMDP as a *joint-action* MDP in which a single, *central* unit applies *joint actions* to the environment [48]. The *joint-action* policy

$$\pi : \mathcal{S} \to \mathcal{A}, \tag{3.2}$$

associated with the shared value function V^{π} is hereby to be maximized. Having a *central* unit follow the present *joint-action* policy is equivalent to having each agent g follow their respective individual policy $\pi_g : \mathcal{S} \to \mathcal{A}_g$, derived from π [48]. As such, MMDPs are said to make up a *centralized* framework.

3.3 Decentralized-POMDP: Decentralized policy execution

A major assumption of MMDPs is that each agent is fully aware of the complete state of the environment, at all times. This assumption may not always be realistic [4]. In 2002, still in the realm of *cooperative* planning, Bernstein et al. [5] took a *decentralized* approach to MDPs in multi-agent settings, by incorporating the idea of *individual- or local- observations*, as well as having the agents be unaware of the other agents' *observations*. This decentralization is fundamentally different to MMDPs and makes DEC-POMDPs more difficult to solve. It should be noted that, while the *online* phase, i.e. execution of the learned plan, is, in fact, completely decentralized, the *offline* phase, i.e., the actual planning, is centralized [49]. More specifically, a single centralized unit computes the joint-action policy offline and then distributes the individual policies to be executed by the agents online.

Definition 3.2 (DEC-POMDP)**.** *A decentralized Partially Observable Markov Decision Process, or DEC-POMDP, $\mathcal{D}$ is formally defined as a tuple*

$$\mathcal{D} := \langle \mathcal{S}, \mathcal{G}, \mathcal{A} = \times_{g \in \mathcal{G}} \mathcal{A}_g, \Omega = \times_{g \in \mathcal{G}} \Omega_g, \mathcal{T}, \mathcal{R}, \mathcal{O} \rangle$$

where:

- *$\mathcal{S}$, $\mathcal{G}$, $\mathcal{A}_g$, $\mathcal{A}$, $\mathcal{T}$, and $\mathcal{R}$ are defined as in Definition* 3.1*.*
- *Ω_g is the set of observations available to agent g.*
- *Ω is the set of joint observations.*
- *$\mathcal{O} : \mathcal{A} \times \mathcal{S} \times \Omega \to [0, 1]$ is called the joint observation function.*

DEC-POMDPs can be reduced to DEC-MDPs if there exists a mapping J such that

$$J : \Omega_1 \times ... \times \Omega_n \to S, \tag{3.3}$$

in other words, if the joint observation of all agents uniquely identifies the state of the environment. The state is then said to be *jointly fully observable*. It is worth noting that even then, the agents themselves are only aware of their *individual observations* [48].

3.4 Interactive-POMDP: A framework for self-interested agents

In 2005, Gmytrasiewicz et al. formalized a further extension of POMDPs to multi-agent settings, called Interactive-POMDP [6]. In contrast to MMDPs and DEC-POMDPs, I-POMDPs make no assumption about the willingness of the agents to be *cooperative*. In fact, I-POMDPs are mainly applied to domains made up of *autonomous* and *self-interested* agents. While in DEC-POMDPs the optimal joint policy is computed by a *centralized* unit before being distributed amid the agents, the self-interested nature of I-POMDP agents requires the *learning* algorithms to be *decentralized*, that is, each agent is to locally compute its own policy as a function of its beliefs about the physical state of the environment as well as its interactions with the other agents.

Definition 3.3 (I-POMDP)**.** *An Interactive Partially Observable Markov Decision Process, or I-POMDP, of an agent, say g, is written $\mathcal{I}_g$ and is formally defined as a tuple*

$$\mathcal{I}_g := \langle \mathcal{IS}_g, \mathcal{A} = \times_{g \in \mathcal{G}} \mathcal{A}_g, \Omega_g, \mathcal{T}_g, \mathcal{R}_g, \mathcal{O}_g \rangle$$

where:

- $\mathcal{IS}_g : \mathcal{S} \times_{h:h \neq g} \mathcal{M}_h$ *is the set of interactive states of agent g. Interactive states are physical environment states augmented with models of each agent in the network.*
- $\mathcal{A}_g$ *and* $\mathcal{A}$ *are defined as in Definition* 3.1*.*
- Ω_g *is the set of observations available to agent g.*
- $\mathcal{T}_g : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$
- $\mathcal{R}_g : \mathcal{IS}_g \times \mathcal{A} \to \mathbb{R}$ *is called the immediate reward of agent g.*
- $\mathcal{O}_g : \mathcal{A} \times \mathcal{S} \times \Omega_g \to [0, 1]$ *is called the observation function of agent g.*

In accordance with the definition of *interactive states*, I-POMDP agents update their beliefs not only over physical states of the environment but also over models of the other agents in the network. The difficulty of solving I-POMDPs lies in the recursive nature of the models. Consider agent g's belief update in a network inhabited by another agent, say h. A model of agent h may consist of the belief function of said agent h over physical states and models of all other agents. These

models may, in turn, consist of belief functions of their own. This nesting of beliefs could theoretically be infinite, rendering the belief update *non-computable*. This inconvenience is overcome by bounding the nesting depth by a finite number n. The problem is then solved as a set of POMDPs.

In 2009, *Doshi et al.* [50] mitigate the computational requirements for solving I-POMDPs by using *Monte Carlo Sampling Methods* to find approximate solutions. In 2011, the complexity of real-world *interactive state spaces* is addressed; *Panella et al.* [51] introduce a new representation for *interactive states* that draws from *relational* (*first-order*) logic and probability theory, in an attempt to compactly and finitely represent *interactive state spaces*. Recently, in 2019, *Han et al.* [52] introduce IPOMDP-Net, a framework that combines I-POMDPs and *Deep Learning*. Their approach is shown to outperform *state-of-the-art* model-free networks. Concretely, IPOMDP-Net is a *deep neural network* that embeds an I-POMDP and a QMDP planner ([36]) in its network architecture.

3.5 The RepNet-POMDP framework

This section introduces the multi-agent, POMDP-based framework called Reputation Network POMDP, which was proposed by *Rens et al.* in their paper "*Maximizing Expected Impact in an Agent Reputation Network*" published in 2018 [1] and constitutes the foundation of the work presented in the rest of the thesis. Several multi-agent frameworks have been presented in this chapter.

To highlight their differences from the RepNet framework, their domains of use are briefly revised here. MMDPs and DEC-POMDPs serve a similar purpose in that they are used to compute a global policy that maximizes the collective well-being of the agents that make up the framework. Consequently, said agents are assumed to be completely selfless. This heavily contrasts with the intended purpose of the RepNet framework, in which each agent uses its *own* instance of the framework, and in which no assumption is made concerning their selflessness. I-POMDPs do serve a similar purpose as far as not making assumptions about an agent's selflessness. They are, however, arguably significantly more difficult to conceptualize than the framework presented hereafter. The RepNet framework achieves a more intuitive layout by making assumptions about the elements that should matter to the agents. In particular, agent *reputation* and *past behavior* of agents are key drivers of the interactions between the agents of the network.

The working principles of the framework are first introduced, after which its planning algorithm is detailed. Note that this section summarizes the framework in its entirety: a few concepts presented hereafter play a minor role in the remainder of this thesis. More concretely, the concepts of *partial observability* and *state estimation* will be worked out of the framework in Chapter 4. The most important concepts of the framework will be reiterated when necessary in future chapters.

3.5.1 Presentation of the framework

The *RepNet* framework is aimed at solving multi-agent learning problems in which each agent is associated with a reputation in the network, and their reputation, as well as past behavior, dictate the willingness of the rest of the network to engage in communications with them. Similarly to classic POMDP systems, the rules of *stochasticity* and *partial observability* apply to the environments considered here. Intuitively, the framework can be thought of as the combinations of 4 pillars.

The underlying Markov Decision Process structure: RepNet agents move about in a *stochastic* environment characterized by a set of *states*, and the rules of the environment are described in a *transition model*. RepNet agents progress in the environment by applying one of the *actions* at their disposal. The primary goal of RepNet agents is the *maximization of its long-term cumulative reward*.

A tracking mechanism of other agents' past behavior: The actions taken by a RepNet agent are informed at all times by the past behavior of other agents in the network.

A tracking mechanism of other agents' reputation: Each agent has an opinion, be it good or bad, of the remaining agents in the network. RepNet agents explicitly monitor the image they *believe* all agents to have of one another, and use this information in their decision-making process.

A tracking mechanism of the environment's current potential states: Analogously to Partially Observable Markov Decision Processes, RepNet agents are uncertain of the current state of the environment. Additionally, said agents monitor the state of the environment other agents believe to be in.

The formal presentation of the framework follows hereafter.

Definition 3.4 (RepNet-POMDP)**.** *Let Σ be a tuple embedding global information about the system, i.e. the environment and agents that are part of said system, and Γ be a tuple embedding each agent's subjective understanding of the environment it operates in. A reputation network POMDP $\mathcal{M}$ is formally defined as a tuple*

$$\mathcal{M} := \langle \Sigma, \Gamma \rangle.$$

Definition 3.5 (System)**.** *A system Σ is formally defined as a tuple*

$$\Sigma := \langle \mathcal{G}, \mathcal{S}, \mathcal{A}, \Omega, \mathcal{I}, \mathcal{U} \rangle$$

where:

- *$\mathcal{G}$ is the set of agents that can interact with the environment.*
- *$\mathcal{S}$ is the set of possible states of the environment.*

- $\mathcal{A}$ *is the set of possible actions, both directed towards a particular agent and undirected. Formally,*

$$\mathcal{A} := \mathcal{A}^d \cup \mathcal{A}^u$$
$$\mathcal{A}^d \cap \mathcal{A}^u := \varnothing$$

  Note that the concept of undirected actions is identical to the concept of actions in classic MDPs. The concept of directed actions will be discussed in greater detail in Section 3.5.3*.*

- Ω *is the set of observations.*

- $\mathcal{I} : \mathcal{G} \times \mathcal{G} \times \mathcal{S} \times \mathcal{A} \to [-1, 1]$ *is called the impact function.* $\mathcal{I}(g, h, s, a)$ *returns the impact on agent* g *that is due to agent* h *performing action* a *in state* s*. This function can be thought of as analogous to a Markov Decision Process's immediate reward function* $\mathcal{R}$*.*

- $\mathcal{U} : [0, 1] \times [-1, 1] \times [-1, 1] \to [-1, 1]$ *is called the image update function. Given a learning rate* α*, a current value* v*, of the image on agent has of another agent, to be updated, and a new expected total impact* i*, of which the definition will be given shortly,* $\mathcal{U}(\alpha, v, i)$ *returns an updated value of the image* v'*.*

Definition 3.6 (Agents)**.** *Let* Γ *be a tuple embedding each agent's subjective understanding of the environment it operates in, such that*

$$\Gamma := \langle \{UT_g\}, \{DT_g\}, \{O_g\}, \{AD_g\}, \{Img_g\}, \{B_g\} \rangle$$

where:

- $UT_g : \mathcal{G} \times \mathcal{S} \times \mathcal{A}^u \times \mathcal{S} \to [0, 1]$ *is called the undirected transition model of agent* g*.* $UT_g(h, s, a, s')$ *returns the probability of the environment transitioning from state* s *to state* s' *when undirected action* a *is taken by agent* h*.*

- $DT_g : \mathcal{G} \times \mathcal{S} \times \mathcal{A}^d \times \mathcal{S} \times [-1, 1] \to [0, 1]$ *is called the directed transition model of agent* g*.* $DT_g(h, s, a_{h \to i}, r_h, s')$ *returns the probability of the environment transitioning from state* s *to state* s' *when agent* h *performs directed action* $a_{h \to i}$ *towards another agent, say* i*, and has a reputation* r_h *according to agent* g*.*

- O_g *is called the observation function, or sensor model, of agent* g*.* $O_g(h, a, s', o)$ *returns the probability of agent* g *making observation* o *after agent* h*, which may or may not be the same as* g*, performs action* a *and the environment transitions to state* s'*.*

- $AD_g : \mathcal{G} \times \mathcal{S} \to \Delta(\mathcal{A})$ *is called the action distribution of agent* g*.* $AD_g(h, s)$ *returns a probability distribution over actions in* $\mathcal{A}$ *for agent* h *in state* s*, according to agent* g*.*

- $Img_g : \mathcal{G} \times \mathcal{G} \to [-1, 1]$ *is called the image function of agent* g. $Img_g(h, i)$ *returns the image agent* i *has of agent* h *according to agent* g.
- $B_g : \mathcal{G} \to \Delta(\mathcal{S})$ *is called the belief state of agent* g. $B_g(h)$ *(generally written* b_h^g *hereafter to simplify the notation) returns the belief state of agent* h*, according to agent* g.

The image two agents have of each other is conditioned by the *impact* they are expected to have on each other, given their current understanding of the state the environment is in. The notion of expected total impact is formalized in Definition 3.7, while its effect on the image function is defined in Definition 3.8.

Definition 3.7 (Expected total impact)**.** *According to agent* g*, the total impact an agent* h *is expected to have on, as well as perceive from, another agent* i*, assumed to be different from* h*, given current belief state function* B_g *and action distribution function* AD_g*, is defined as*

$$\begin{aligned} ETI_g(h, i, B_g, AD_g) := \sum_{s \in \mathcal{S}} b_h^g(s) b_i^g(s) \sum_{a \in \mathcal{A}} \big[& \delta AD_g(i, s)(a) \mathcal{I}(h, i, s, a) \\ & + (1 - \delta) AD_g(h, s)(a) \mathcal{I}(i, h, s, a) \big] \end{aligned}$$

where $\delta \in [0, 1]$ *trades off the relative importance of the impact due to agent* h *and the impact perceived by* h.

Definition 3.8 (Image update)**.** *Let* Img_g *be the current image function of agent* g*. The updated image function* Img'_g *is computed as follows:*

$$\begin{aligned} Img'_g &:= IE(g, Img_g, \alpha, B_g, AD_g) \\ &:= \Big\{ (h, i, t) \;\Big|\; h, i \in \mathcal{G} \wedge t = \mathcal{U}(\alpha, Img_g(h, i), ETI_g(h, i, B_g, AD_g)) \Big\} \end{aligned}$$

where

- IE *is called the image estimation function.*
- α *is called the learning rate.*
- B_g *is the current belief state of agent* g.
- AD_g *is the current action distribution of agent* g.

The reputation of an agent, say h, as defined by *Rens et al.* [1], brings together the image each agent has of h as well as the image the RepNet agent of interest g has of all the agents that make up the network. The mathematical definition is given in Definition 3.9.

Definition 3.9 (Reputation)**.** *The reputation of an agent* h*, according to agent* g*, is formally defined as*

$$REP_g(h, Img_g) := \frac{1}{|\mathcal{G}|} \sum_{i \in \mathcal{G}} Img_g(h, i) \times Img_g(i, g)$$

where $Img_g(i, i) = 1 \; \forall i \in \mathcal{G}$.

To simplify the notation, it is convenient to define a *global transition model* that combines both directed and undirected transition models. The behavior of the new function should be dictated by the nature of the action passed as an argument to it.

Definition 3.10 (Global transition model)**.** *The global transition model of agent g, $T_g : \mathcal{G} \times \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times [-1, 1] \to [0, 1]$, is formally defined as*

$$T_g(h, s, a_h, s', r_h) := \begin{cases} DT_g(h, s, a_h, s', r_h) & \text{if } a_h \in \mathcal{A}^d \\ UT_g(h, s, a_h, s') & \text{if } a_h \in \mathcal{A}^u \end{cases}$$

where

- *h is the sender of action a.*
- *a_h is the action, directed or undirected, taken by agent h.*
- *s is the current state of the environment.*
- *s' is the potential future state of the environment. Note that this state may be the same as state s.*
- *r_h is the reputation of agent h, according to agent g.*

RepNet agents are uncertain of the state of the environment. Keeping track of a probability distribution over all possible states of the environment assists them in the decision-making process. Additionally, the partial observability that applies to them also applies to every agent in the network. Monitoring the state of the environment that the *other* agents in the network believe to be in has the purpose of helping RepNet agents make better-informed decisions.

An agent's *own* belief state, called *objective belief state* hereafter, changes as it takes actions and makes new observations on the environment. The belief state update, defined in Definition 3.11, is analogous to that performed in classic POMDP problems (Equation 2.5).

Definition 3.11 (Objective state estimation)**.** *Let b_g^g be the current belief state of agent g according to itself, or objective belief state of agent g. The updated objective belief state $b_g^{g\prime}$ is computed as follows:*

$$\begin{aligned} b_g^{g\prime} &:= OSE(b_g^g, a, g, o, r_g) \\ &:= \left\{ (s', p) \;\middle|\; s' \in \mathcal{S} \wedge p = \frac{O_g(g, a, s', o) \sum_s T_g(g, s, a, s', r_g) b_g^g(s)}{P_g(o | b_g^g, a, g, r_g)} \right\} \end{aligned}$$

where:

- *OSE is called the objective state estimation function.*
- *a is the action taken by agent g.*
- *o is the observation made by agent g after taking action a.*

- r_g *is the reputation of agent* g*, according to itself.*
- $P_g(o|b_g^g, a, g, r_g) = \sum_{s'} O_g(g, a, s', o) \sum_s T_g(g, s, a, s', r_g) b_g^g(s)$ *is a normalizing constant.*

Alongside its own belief state, each agent has an idea of what state other agents in the network believe the environment to be in. This information is maintained in so-called *subjective belief states*, which, similarly to their objective counterparts, are updated upon receiving new observations. The update of subjective belief states is defined in Definition 3.12.

Definition 3.12 (Subjective state estimation)**.** *Let* b_h^g *be the current belief state of agent* h *according to agent* g*. The updated subjective belief state* $b_h^{g\prime}$ *is computed as follows:*

$$
\begin{aligned}
b_h^{g\prime} &:= SSE(b_h^g, AD_g, h, o, r_h) \\
&:= \left\{ (s', p) \;\middle|\; s' \in \mathcal{S} \wedge p = \frac{\sum_a O_g(h, a, s', o) \sum_s T_g(h, s, a, s', r_h) b_h^g(s) AD_g(h, s)(a)}{P_g(o|b_h^g, h, o, r_h, AD_g)} \right\}
\end{aligned}
$$

where:

- SSE *is called the subjective state estimation function.*
- o *is the observation made by agent* g*.*
- r_h *is the reputation of agent* h*, according to agent* g*.*
- $P_g(o|b_h^g, h, o, r_h, AD_g) = \sum_{s'} \sum_a O_g(h, a, s', o)$ $\cdot \sum_s T_g(h, s, a, s', r_h) b_h^g(s) AD_g(h, s)(a)$ *is a normalizing constant.*

The concepts of *objective* and *subjective* state estimation can now be combined into a single concept called *total state estimation*.

Definition 3.13 (Total state estimation)**.** *Let* B_g *be the current belief state function of agent* g*, and* o *is the observation made by agent* g *after taking action* a*. The updated belief state function* B_g' *is computed as follows:*

$$
\begin{aligned}
B_g' &:= SE(g, B_g, a, o, Img_g, AD_g) \\
&:= \left\{ (h, SSE(b_h^g, AD_g, h, o, r_h)) \;\middle|\; h \in \mathcal{G} \setminus \{g\} \wedge r_h = REP_g(h, Img_g) \right\} \\
&\quad \cup \left\{ (g, OSE(b_g^g, a, g, o, REP_g(g, Img_g))) \right\}
\end{aligned}
$$

where SE *is called the total state estimation function.*

The next step in the formalization of RepNet-POMDPs consists in defining an updating scheme of the action distribution AD_g of each agent g. *Rens et al.* [1] propose a model based on Bayesian conditioning.
A Bayesian agent g updates its subjective experience of the world it operates in light

of new information. Say an environment hosting two agents g and h is currently in state s. Agent g has an *a priori* idea of the probability of agent h picking an action a in state s,

$$P_g(a|h, s, r_h). \tag{3.4}$$

Following agent h performing action a in this state, the environment transitions from state s to a new unknown state. The observation o agent g makes can be used by said agent to update its subjective perception of the *a posteriori* probability of agent h performing that same action a in state s in the future, using Bayes' theorem[1],

$$\begin{aligned} P_g(a|h, s, r_h, o) &= \frac{P_g(o|h, s, r_h, a)P_g(a|h, s, r_h)}{P_g(o|h, s, r_h)} \\ &= \frac{P_g(o|h, s, r_h, a)P_g(a|h, s)}{\sum_{a'} P_g(o|h, s, r_h, a')P_g(a'|h, s)} \end{aligned} \tag{3.5}$$

where $P_g(o|h, s, r_h, a)$ is calculated as follows:

$$\begin{aligned} P_g(o|h, s, r_h, a) &= \sum_{s' \in \mathcal{S}} P_g(o|h, s, r_h, a, s')P_g(s'|h, s, r_h, a) \\ &= \sum_{s' \in \mathcal{S}} O_g(h, a, s', o)T_g(h, s, a, s', r_h) \end{aligned} \tag{3.6}$$

The probabilities in Equation 3.5 can now be replaced by the terms defined in Definition 3.6 and Definition 3.10:

$$P_g(a|h, s, r_h, o) = \frac{\sum_{s'} T_g(h, s, a, s', r_h)O_g(h, a, s', o)AD_g(h, s)(a)}{\sum_{a'} \sum_{s'} T_g(h, s, a', s', r_h)O_g(h, a', s', o)AD_g(h, s)(a')}. \tag{3.7}$$

Definition 3.14 (Action distribution update)**.** *Let AD_g be the current action distribution function of agent g. The updated action distribution AD'_g is computed as follows:*

$$\begin{aligned} AD'_g &:= ADE(g, o, AD_g, Img_g) \\ &:= \left\{ (h, s, a, p) \;\middle|\; h \in \mathcal{G} \wedge s \in \mathcal{S} \wedge a \in \mathcal{A} \wedge r_h = REP_g(h, Img_g) \right. \\ &\qquad \left. \wedge\, p = \frac{\sum_{s'} T_g(h, s, a, s', r_h)O_g(h, a, s', o)AD_g(h, s)(a)}{\sum_{a'} \sum_{s'} T_g(h, s, a', s', r_h)O_g(h, a', s', o)AD_g(h, s)(a')} \right\} \end{aligned}$$

where

- *ADE is called the action distribution estimation function.*
- *o is the observation made by agent g.*

[1] Bayes' theorem is defined mathematically as follows: $P(A|B, C) = \frac{P(B|A,C)P(A|C)}{P(B|C)}$, where A, B and C are events and $P(B|C) \neq 0$

3.5.2 Transition model

While the notion of transition models is used throughout *Rens et al.* [1], the physical meaning of each agent's transition model and how these transition models relate to one another is not explained. This section summarizes the approach taken in this thesis by illustrating it with a simple example that involves 2 agents.

Let A and B be two agents deployed in an environment that can be in any of the states in $\mathcal{S} = \{s_0, s_1\}$. Each agent can pick any of the actions in $\mathcal{A} = \{a_0, a_1\}$. For the sake of keeping the scenario simple, $\mathcal{A} = \mathcal{A}^u$, that is, all actions are undirected. Agent A's transition model is given by $UT_A(A, s, a, s')$, while Agent B's is given by $UT_B(B, s, a, s')$. These models describe the rules the agents abide by. The current definition for these models does, however, not explain how these individual models relate to the actual working principles of the environment. To address this point, a *transition model of the environment* $\mathcal{T}$ can be introduced:

$$\mathcal{T} : \mathcal{S} \times \mathcal{A}_A \times \mathcal{A}_B \times \mathcal{S} \to [0, 1], \tag{3.8}$$

such that $\mathcal{A}_A = \mathcal{A}_B = \mathcal{A}$. This definition is akin to the definition of transition model for Multi-agent MDPs (Section 3.2) and Decentralized-POMDPs (Section 3.3). The individual transition models are now obtained as follows:

$$\begin{aligned} UT_A(A, s_i, a_i, s_j) &= \frac{1}{|\mathcal{A}_B|} \sum_{a' \in \mathcal{A}_B} \mathcal{T}(s_i, a_i, a', s_j) \quad \forall s_i, s_j \in \mathcal{S}, \forall a_i \in \mathcal{A}_A \\ UT_B(B, s_i, a_i, s_j) &= \frac{1}{|\mathcal{A}_A|} \sum_{a' \in \mathcal{A}_A} \mathcal{T}(s_i, a', a_i, s_j) \quad \forall s_i, s_j \in \mathcal{S}, \forall a_i \in \mathcal{A}_B. \end{aligned} \tag{3.9}$$

Note that with this approach,

$$\begin{aligned} UT_A(A, s_i, a_i, s_j) &= UT_B(A, s_i, a_i, s_j) \\ UT_B(B, s_i, a_i, s_j) &= UT_A(B, s_i, a_i, s_j), \end{aligned} \tag{3.10}$$

that is, the individual transition models are independent from the agent they belong to. In an effort to simplify the notation, the subscript identifying this agent is dropped henceforth.

The definition provided above can be generalized to n agents. Let $\mathcal{A}'$ be the set of all possible combinations of actions for each agent with the exception of Agent i, i.e.,

$$\mathcal{A}' = \mathcal{A}_1 \times ... \times \mathcal{A}_{i-1} \times \mathcal{A}_{i+1} \times ... \times \mathcal{A}_n \tag{3.11}$$

Agent i's individual transition model is obtained as follows:

$$UT(i, s_i, a_i, s_j) = \frac{1}{|\mathcal{A}'|} \sum_{a' \in \mathcal{A}'} \mathcal{T}(s_i, a_i, a', s_j) \quad \forall s_i, s_j \in \mathcal{S}, \forall a_i \in \mathcal{A}_i \tag{3.12}$$

3.5.3 The concept of directed actions

The present section covers the concept of *directed actions*, as described by *Rens et al.* [1]. It furthermore serves to point out a shortcoming in the current definition of directed action. In their paper, *Rens et al.* describe a trade between two agents A and B. A is the *buyer*, and B is the *seller*. The environment is described by the set of states

$$\mathcal{S} = \{s_0, a, r\}, \tag{3.13}$$

where s_0 is the initial state, a, called the *accept* state, is the state in which the trade transaction was successfully completed, and r, called the *refuse* state, is the state in which the trade transaction was unsuccessful. The actions at the disposal of both agents is described by the set

$$\mathcal{A} = \{\texttt{trade\_with\_B}, \texttt{accept}, \texttt{refuse}, \texttt{wait}\}. \tag{3.14}$$

For the sake of simplicity, the environment is assumed to behave *deterministically*, i.e., the outcome of any one combination of actions of A and B is always the same. Formally,

$$\begin{aligned} \mathcal{T}(s_0, \texttt{trade\_with\_B}, \texttt{accept}, a) &= 1 \\ \mathcal{T}(s_0, \texttt{trade\_with\_B}, \texttt{refuse}, r) &= 1, \end{aligned} \tag{3.15}$$

where $\mathcal{T}$ is the *transition model of the environment*, as per Section 3.5.2. The transition model *of the environment* is given in Figure 3.1.

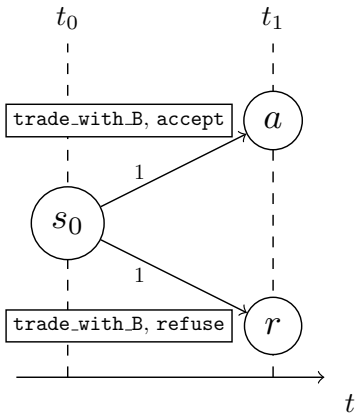


Figure 3.1: Transition model $\mathcal{T}$ of the trading example. s_0 is the initial state, a is the *accept* state, and r is the *refuse* state. The transition model of the environment $\mathcal{T}$ is assumed to be deterministic.

At time t_0, agent A performs action `trade_with_B` in state s_0. According to *Rens et al.*'s definition of directed actions, the *directed* probability of landing in state

a or r at time t_1 is conditioned by the reputation of A, the agent initiating the trade. Consider the following situation: at time t_0 agent A has a reputation of 0.5, which is considered to be relatively good. The probability of B being willing to trade with A is, say, 0.9, that is,

$$\begin{aligned} DT(A, s_0, \texttt{trade\_with\_B}, a, 0.5) &= 0.9 \\ DT(A, s_0, \texttt{trade\_with\_B}, r, 0.5) &= 0.1. \end{aligned} \tag{3.16}$$

Said differently, at time t_0, agent B will select action `accept` with a probability of 0.9, and action `refuse` with a probability of 0.1. The problem with the interpretation of the concept of directed actions by *Rens et al.* is twofold:

- The first problem resides in the fact that agent B is not aware of agent A's desire to trade at time t_0, as agent A has not yet performed `trade_with_B`. Said differently, agent B has no reason to prefer `accept` over `refuse` at this point in time.
- Suppose now that agent B is made aware of the action agent A is about to perform at t_0. The second problem relates to the fundamental properties of transition models. A transition model describes the rules the environment abides by, that is, it explains how the environment reacts *on the basis of the actions taken* by its actors. To move the environment from s_0 to a, agents A and B need to commit to actions `trade_with_B` and `accept` respectively. *Directed* transition models fail to be able to describe the rules of the environment. To illustrate the problem clearly, the cases in which B has committed to action `accept` and in which it has not yet done so will be considered separately:
  - **Agent B has committed to action `accept`:** If both agents have committed to their respective actions, then the transitions can simply be explained using *undirected* transitions, that is, transitions that do not depend on the reputation of any agent. As per Section 3.5.2, the individual *undirected* transitions that explain the rules of the environment as they pertain to agent A are formally given by

$$\begin{aligned} UT(A, s_0, \texttt{trade\_with\_B}, a) &= \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} \mathcal{T}(s_0, \texttt{trade\_with\_B}, a', a) \\ &= \frac{1}{4}(0 + 1 + 0 + 0) = \frac{1}{4} \\ &= UT(A, s_0, \texttt{trade\_with\_B}, r) \end{aligned}$$

  - **Agent B has not yet committed to action `accept`:** If agent B has not committed to an action yet, then the missing action makes it impossible to use the *transition model of the environment* (Section 3.5.2) as a basis for deriving the *directed* transition models. Said differently, the directed transitions given in Equation 3.16 can not possibly describe the rules of the environment as they apply to agent A. At most, they describe agent A's *subjective* impression of the impact that its own reputation might have on the willingness of agent B to accept its trade offer.

A solution to both problems covered in this section will follow in Section 4.2.

3.5.4 Planning for Optimal Impact

In RepNet-POMDPs, an agent's ability to thrive in the network is reflected in the reputation it has in the eyes of the remaining agents. A good reputation for agent g increases the likelihood of other agents being willing to *cooperate* with g. Moreover, an agent should perform actions according to the *perceived immediate impact* they have on itself.

Definition 3.15 (Perceived immediate impact)**.** *The perceived immediate impact on agent g, written PI_g, is defined as*

$$\begin{aligned} PI_g(B_g, AD_g, a) :=& \frac{1}{|\mathcal{G}|} \sum_{s \in \mathcal{S}} b_g^g(s) \big[\mathcal{I}(g, g, s, a) \\ &+ \sum_{h \in \mathcal{G} \setminus \{g\}} \sum_{a' \in \mathcal{A}} \mathcal{I}(g, h, s, a') b_h^g(s) AD_g(h, s)(a') \big] \end{aligned}$$

where

- *a is the action performed by agent g.*
- *AD_g is the current action distribution of agent g.*
- *B_g is the current belief state of agent g.*
- *The first term describes the immediate self-impact as a consequence of agent g performing action a.*
- *The second describes the expected immediate impact that the network, i.e., the remaining agents, has on agent g.*

Analogously to classical POMDPs, a RepNet-POMDP agent g strives to maximize its expected discounted perceived impact

$$\mathbb{E}\big[\sum_{t=0}^{k} \gamma^t PI_{g,t} \big] \tag{3.17}$$

where γ is the *discount factor* and $PI_{g,t}$ is agent g's perceived immediate impact at time-step t. This is accomplished by computing the optimal value function V_g, as defined in Definition 3.16.

Definition 3.16 (Value function)**.** *Let V_g be the optimal value function of agent g in a finite-horizon setting. It satisfies the optimality equations, which are defined as*

$$\begin{cases} V_g(B_g, AD_g, Img_g, k) := & \max\limits_{a \in \mathcal{A}} \Big\{ PI_g(B_g, AD_g, a) \\ & \quad + \gamma \sum\limits_{o \in \Omega} P_g(o | B_g, AD_g, Img_g, a) V_g(B'_g, AD'_g, Img'_g, k-1) \Big\} \\ V_g(B_g, AD_g, Img_g, 1) := & \max\limits_{a \in \mathcal{A}} \Big\{ PI_g(B_g, AD_g, a) \Big\} \end{cases}$$

where

- $B'_g = SE(g, B_g, a, o, Img_g, AD_g)$.
- $AD'_g = ADE(g, o, AD_g, Img_g)$.
- $Img'_g = IE(g, Img_g, \alpha, B_g, AD_g)$.
- $\gamma \in [0, 1]$ *is called the discount factor.*

$P_g(o|B_g, AD_g, Img_g, a)$ *can be reformulated using the concepts of observation function, transition model, and belief state (note that the development that follows was left out of Rens et al.'s original paper [1]):*

$$\begin{aligned} P_g(o|B_g, AD_g, Img_g, a) &= \sum_{s' \in \mathcal{S}} P_g(o|B_g, AD_g, Img_g, a, s') P_g(s'|B_g, AD_g, Img_g, a) \\ &= \sum_{s' \in \mathcal{S}} P_g(o|a, s') \sum_{s \in \mathcal{S}} P_g(s'|a, s) P_g(s|B_g, AD_g, Img_g, a) \\ &= \sum_{s' \in \mathcal{S}} O_g(g, a, s', o) \sum_{s \in \mathcal{S}} T_g(g, s, a, s', r_g) b_g^g(s) \end{aligned}$$

Definition 3.17 (Optimal policy)**.** *Let B_g be the current belief state function, AD_g the current action distribution, Img_g the current image function of agent g, and k the current horizon. The optimal action for agent g, written $\pi(B_g, AD_g, Img_g, k)$, is defined as*

$$\begin{aligned} \pi(B_g, AD_g, Img_g, k) := \arg\max_{a \in \mathcal{A}} \Big\{ & PI_g(B_g, AD_g, a) \\ & + \gamma \sum_{o \in \Omega} P_g(o|B_g, AD_g, Img_g, a) V_g(B'_g, AD'_g, Img'_g, k-1) \Big\}. \end{aligned}$$

Chapter 4

The RepNet-MDP framework

The primary goal of this thesis is to study the properties of the RepNet framework. More concretely, the concepts of *action distribution*, *image*, *reputation*, and *directed actions*, as well as their impact on RepNet agents' behavior are key elements that distinguish it from classic (Partially Observable) Markov Decision Processes. While the notion of *partial observability* for RepNet agents differs slightly from the definition employed by POMDP agents, in that RepNet agents have to keep track of the state in which other agents believe to be in addition to their own belief, it is arguably the feature that separates RepNet-POMDPs the least from other MDP-derived frameworks. In an attempt to bring the RepNet framework closer to *tractability* and simplify the study of its properties, it can be reduced from its POMDP setting to a fully observable setting. Several adjustments, covered in Section 4.1, are necessary to make the starting framework *fully observable*. Further alleviation of the *intractability* of the framework can be achieved by favoring approximate solutions of satisfactory quality over exact solutions. To this end, a look-ahead-based approach similar to that described in the context of classic MDPs in Section 2.4 can be taken.

This chapter follows the following structure: Section 4.1 describes the steps necessary to reduce the RepNet framework to a fully observable setting and follows with the presentation of the reduced framework. Section 4.2 provides a solution to the concept of *directed actions* imagined by *Rens et al.* [1]. Section 4.3 reviews the concept of *action distribution estimation* in fully observable settings. Section 4.4 then discusses planning for optimal impact in the context of RepNet-MDPs. Section 4.5 introduces the online look-ahead algorithm used to further alleviate the intractability of the reduced RepNet framework. Finally, Section 4.6 briefly reviews the mathematical definition of *reputation*.

4.1 Reduction of the RepNet-POMDP framework

A RepNet-MDP is, analogously to its POMDP counterpart (Definition 3.4), defined as a tuple

$$\mathcal{M} := \langle \Sigma, \Gamma \rangle.$$

where Σ is the *system* tuple and Γ is the *agents* tuple. Several adjustments are necessary to make the starting framework fully observable.

- The system as defined in Definition 3.5 includes the set of possible observations Ω. Given that neither impact function $\mathcal{I}$ nor update function $\mathcal{U}$ make use of Ω, it may be removed without further ramifications in the new definition of the system (Definition 4.1).
- The agents that make up Γ in Definition 3.6 notably each have their own observation function O_g, which furthermore appears in several equations:

  1. Each agent g's subjective perception of the world is maintained in the functions AD_g, Img_g, and B_g. The update of these concepts is conditioned by the observations it makes.
  2. The optimality equations (Definition 3.16) sum over all possible observations, after which the recursive calls to the value function are weighted using the observation function.

  In a fully observable environment, each agent is completely aware of the current state of the environment, rendering the belief state function B_g of each agent g futile. More specifically,

  1. Each agent g's image function Img_g is updated partly on the basis of g's current belief function, as it appears in the expected total impact function (Definition 3.7).
  2. The perceived immediate impact on agent g (Definition 3.15) is likewise conditioned by its current belief state function B_g, and is simplified in the reduced framework.
  3. The recursive calls in the optimality equations are weighted by the agent's current belief state function.

Definition 4.1 (RepNet-MDP system)**.** *A system in a RepNet-MDP Σ is formally defined as a tuple*

$$\Sigma := \langle \mathcal{G}, \mathcal{S}, \mathcal{A}, \mathcal{I}, \mathcal{U} \rangle$$

where $\mathcal{G}$, $\mathcal{S}$, $\mathcal{A}$, $\mathcal{I}$, and $\mathcal{U}$ are defined as in Definition 3.5*.*

Definition 4.2 (Agents)**.** *Let Γ be a tuple embedding each agent's subjective understanding of the environment it operates in, such that*

$$\Gamma := \langle \{UT_g\}, \{DT_g\}, \{AD_g\}, \{Img_g\} \rangle$$

where UT_g, DT_g, AD_g, and Img_g are defined as in Definition 3.6*.*

The updated definition of *expected total impact* is given in Definition 4.3, while its effect on the *image function* is given in Definition 4.4.

Definition 4.3 (Expected total impact)**.** *According to agent g, the total impact an agent h is expected to have on, as well as perceive from, another agent i, assumed to be different from h, when the environment is in state s, is defined as*

$$ETI_g(h,i,s,AD_g) := \sum_{a\in\mathcal{A}} \left[\, \delta AD_g(i,s)(a)\mathcal{I}(h,i,s,a) + (1-\delta)AD_g(h,s)(a)\mathcal{I}(i,h,s,a) \,\right]$$

where $\delta \in [0,1]$ trades off the relative importance of impact due to agent h and impact perceived by h.

Definition 4.4 (Image update)**.** *Let Img_g be the current image function of agent g. The updated image function Img'_g is computed as follows:*

$$\begin{aligned} Img'_g &:= IE(g, Img_g, \alpha, s, AD_g) \\ &:= \left\{ (h,i,t) \;\middle|\; h,i \in \mathcal{G} \wedge t = \mathcal{U}(\alpha, Img_g(h,i), ETI_g(h,i,s,AD_g)) \right\} \end{aligned}$$

where

- *IE is called the image estimation function.*
- *α is called the learning rate.*
- *s is the current state of the environment.*

The mathematical definition of reputation for RepNet-POMDPs (Definition 3.9), given by

$$REP_g(h, Img_g) := \frac{1}{|\mathcal{G}|} \sum_{i\in\mathcal{G}} Img_g(h,i) \times Img_g(i,g), \tag{4.1}$$

can produce undesirable results in regards to self-reputation. To demonstrate this, consider a scenario in which two agents, say A and B, are deployed in an environment without ever having been in contact with one another. Their image of each other can, therefore, be assumed neutral, i.e.,

$$Img_A(B,A) = 0, \qquad Img_A(B,A) = 0. \tag{4.2}$$

At this point, the desired reputation-related numbers should show that agent A believes every agent's reputation, including its own, to be neutral. While this is indeed the case for agent B's reputation, the same does not apply to agent A's self-reputation. In fact,

$$\begin{aligned} REP_A(A, Img_A) &= \frac{1}{2}\left[Img_A(A,B) \times Img_A(B,A) + Img_A(A,A) \times Img_A(A,A)\right] \\ &= \frac{1}{2}\left[0+1\right] = \frac{1}{2}. \end{aligned}$$

As such, agent A believes itself to initially have a higher reputation than agent B. The definition of reputation that follows was adjusted to address this inconvenience.

Definition 4.5 (Adjusted Reputation)**.** *The reputation of an agent h, according to agent g, is formally defined as*

$$REP_g(h, Img_g) := \begin{cases} \frac{1}{|\mathcal{G}|} \sum_{i \in \mathcal{G}} Img_g(h,i) \times Img_g(i,g) & \text{if } h \neq g \\ \frac{1}{|\mathcal{G}|-1} \sum_{i \in \mathcal{G} \setminus \{g\}} Img_g(h,i) \times Img_g(i,g) & \text{otherwise} \end{cases}$$

where $Img_g(i,i) = 1 \; \forall i \in \mathcal{G}$.

4.2 Reimagined concept of directed actions

In Section 3.5.3, the concept of *directed* transitions, as it was developed by *Rens et al.* [1], was covered. Using a trading example between two agents, of which the transition model of the environment $\mathcal{T}$ is repeated in Figure 4.1, two problems were identified.

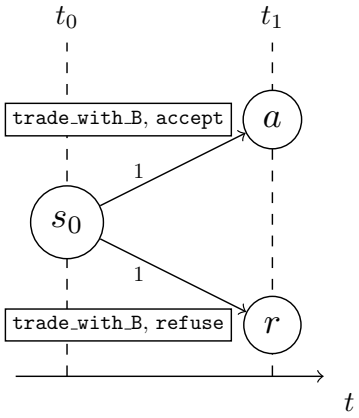


Figure 4.1: Transition model $\mathcal{T}$ of the trading example. s_0 is the initial state, a is the *accept* state, and r is the *refuse* state. The transition model of the environment $\mathcal{T}$ is assumed to be deterministic.

The first problem resides in the fact that agent A has not yet performed action `trade_with_B` at time t_0 and, as such, agent B is not aware of A's intentions. Concretely, there is no incentive for agent B to favor action `accept` over action `refuse` at t_0. This problem is easily addressed by adding an *intermediate* state s_1, in which agent B is made aware of agent A's desire to trade, to the transition model of the environment, and by rearranging the combinations of actions between any two states. The transition model with the intermediate state added is given in Figure 4.2.

The second problem pertains to the fact that *directed* transition models can not describe the actual rules of the environment. In the case of the trade between agents A and B, they can, at most, describe agent A's *subjective* impression of the impact that its own reputation might have on the willingness of agent B to accept its trade offers. As such, one must make a clear distinction between the purpose of an

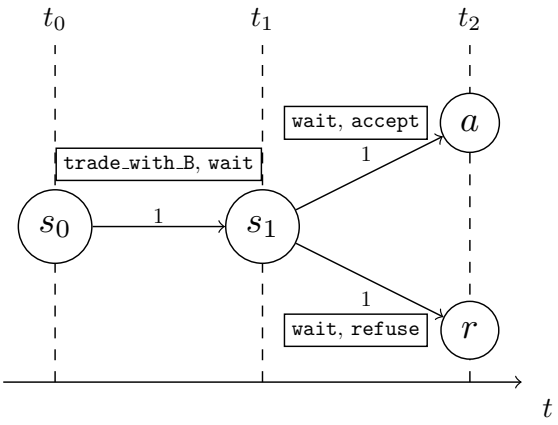


Figure 4.2: Updated transition model $\mathcal{T}$ of the trading example, s_1 is the state in which Agent B is made aware of the trade offer.

undirected transition model, which describes the actual rules of the environment as they apply to a single agent, and that of a *directed* transition model, which describes that agent's *subjective* perception of the rules of the environment. A version of the trading scenario between agents A and B that was adapted to make use of the concept of directed actions is presented hereafter.

The environment is made up of the same set of states $\mathcal{S} = \{s_0, s_1, a, r\}$ depicted in Figure 4.2. The set of *undirected* actions at the disposal of both agents is given by

$$\mathcal{A}^u = \{\texttt{trade\_with\_B}, \texttt{accept}, \texttt{refuse}, \texttt{wait}\}. \tag{4.3}$$

In the eyes of agent A, agent B's response to a trade offer depends on A's reputation. Concretely, $s_1 \to a$ and $s_1 \to r$ make up the transitions for which agent A believes its reputation to be relevant. The action taken by agent A during these transitions is `wait`. To make use of the notion of directed actions, the set of *directed* actions $\mathcal{A}^d$ will contain the equivalent of `wait` in its *directed* form, that is,

$$\mathcal{A}^d = \{\texttt{wait\_d}\}. \tag{4.4}$$

The way agent A makes use of actions in $\mathcal{A}^u$ and $\mathcal{A}^d$ can now be detailed. When reasoning about the value of each action, that is, when *planning* to maximize its expected impact, agent A will make use of *undirected* actions in $\mathcal{A}^u$ whenever the action currently investigated has no equivalent in $\mathcal{A}^d$. For instance,

$$T_A(A, s_0, \texttt{trade\_with\_B}, s_1, r_A) = UT(A, s_0, \texttt{trade\_with\_B}, s_1). \tag{4.5}$$

When an action in $\mathcal{A}^u$ has an equivalent in $\mathcal{A}^d$, agent A will make use of the concept of directed actions in planning to maximize its expected impact. For instance,

$$T_A(A, s_1, \texttt{wait}, a, r_A) = DT_A(A, s_1, \texttt{wait\_d}, a, r_A). \tag{4.6}$$

As such, the reputation of agent A is accounted for when agent A *plans* to maximize its expected impact. It is worth stressing that the rules of the environment do not change when agent A reasons in terms of directed actions. Moreover, when agent A is in s_1 and is to wait for agent B's response to the trade offer, the action that is performed on the environment is the *undirected* variant `wait`.

4.3 Reviewing the action distribution estimation function

The next step in the formalization of RepNet-MDPs consists in redefining the updating scheme of the action distribution AD_g of each agent g. Initially, the reasoning applied here is similar to that applied to RepNet-POMDPs. Say an environment hosting two agents g and h is currently in state s. Agent g has an *a priori* notion of the probability of agent h picking an action a in state s,

$$P_g(a|h, s, r_h). \tag{4.7}$$

Following agent h performing action a in this state, the environment transitions from state s to state s'. The *a posteriori* probability of agent h performing that same action a in state s in the future is now computed using Bayes' rule:

$$\begin{aligned} P_g(a|h, s, r_h, s') &= \frac{P_g(s'|h, s, r_h, a)P_g(a|h, s, r_h)}{P_g(s'|h, s, r_h)} \\ &= \frac{P_g(s'|h, s, r_h, a)P_g(a|h, s)}{\sum_{a'} P_g(s'|h, s, r_h, a')P_g(a'|h, s)}. \end{aligned} \tag{4.8}$$

The probabilities may now be replaced by the terms defined in Definition 4.2 and Definition 3.10:

$$AD'_g(h, s)(a) = \frac{T_g(h, s, a, s', r_h)AD_g(h, s)(a)}{\sum_{a'} T_g(h, s, a', s', r_h)AD_g(h, s)(a')}. \tag{4.9}$$

While MDP-derived formalisms are designed to deal with stochastic environments, one should expect them to work with deterministic transition models. The *action distribution update* in *fully observable* settings, as given in Equation 4.9, can produce undesirable results that make it impossible for a RepNet agent to *unlearn* what it has learned. Consider the trading scenario between agents A and B described in Section 4.2 and whose transition model is schematized in Figure 4.2. The transitions of the environment are assumed to be deterministic, that is,

$$UT(B, s_1, \texttt{accept}, a) = 1, \quad UT(B, s_1, \texttt{refuse}, r) = 1. \tag{4.10}$$

If agent B refuses the trade offer made by agent A, the environment transitions to state r and the action distribution is updated as follows:

$$\begin{aligned} AD'_A(B, s_1)(\texttt{accept}) &= \frac{UT(B, s_1, \texttt{accept}, r) AD_A(B, s_1)(\texttt{accept})}{\sum_{a'} UT(B, s_1, a', r) AD_A(B, s_1)(a')} \\ &= \frac{0 \cdot AD_A(B, s_1)(\texttt{accept})}{\sum_{a'} UT(B, s_1, a', r) AD_A(B, s_1)(a')} \\ &= 0 \end{aligned}$$

$$\begin{aligned} AD'_A(B, s_1)(\texttt{refuse}) &= \frac{UT(B, s_1, \texttt{refuse}, r) AD_A(B, s_1)(\texttt{refuse})}{\sum_{a'} UT(B, s_1, a', r) AD_A(B, s_1)(a')} \\ &= \frac{1 \cdot AD_A(B, s_1)(\texttt{refuse})}{\sum_{a'} UT(B, s_1, a', r) AD_A(B, s_1)(a')} \\ &= 1 \end{aligned}$$

As dictated by the new action distribution, agent A is, in the wake of a *single* unsuccessful trade, convinced that agent B will never accept any trade offer in the future. Moreover, the primary issue does not reside in the fact that agent A was quick to make up its mind, but in the fact that it is now impossible for agent A to change its strategy in the future. In fact, the probability of B accepting a trade is 0, and regardless of what this value is multiplied by in the future, it will always remain 0.

This inconvenience is addressed hereafter by applying a *smoothing* technique called *Laplace smoothing* [53]. Let $\theta = \langle \theta_1, \theta_2, ..., \theta_d \rangle$, such that $\theta_i \in [0, 1]\ \forall i$ and $\sum_i \theta_i = 1$, be the probabilities of a multinomial distribution with d categories. The *Laplace-smoothed* estimators of the probabilities are given by

$$\hat{\theta}_i = \frac{\theta_i + \eta}{\sum_i (\theta_i + \eta)} = \frac{\theta_i + \eta}{\sum_i \theta_i + d\eta}, \tag{4.11}$$

where $\eta \in [0, 1]$ is called the *smoothing parameter*. If $\theta_i \gg \eta$, then $\hat{\theta}_i \simeq \theta_i$. If $\theta_i \ll \eta$, then $\hat{\theta}_i \simeq \frac{1}{d}$. The smoothing technique thus prevents probabilities of 0 from ever occurring. The same smoothing technique can be applied to the *action distribution update* function, resulting in the following equation:

$$AD'_g(h, s)(a) = \frac{T_g(h, s, a, s', r_h) AD_g(h, s)(a) + \eta}{\sum_{a'} (T_g(h, s, a', s', r_h) AD_g(h, s)(a') + \eta)}. \tag{4.12}$$

The new definition of the *action distribution estimation* function (Definition 4.6) takes this smoothing technique into account.

Definition 4.6 (Adjusted Action distribution update)**.** *Let AD_g be the current action distribution function of agent g. The updated action distribution function*

AD'_g is computed as follows:

$$\begin{aligned} AD'_g &:= ADE(g, s', AD_g, Img_g) \\ &:= \left\{ (h, s, a, p) \;\middle|\; h \in \mathcal{G} \wedge s \in \mathcal{S} \wedge a \in \mathcal{A} \wedge r_h = REP_g(h, Img_g) \right. \\ &\qquad \left. \wedge\, p = \frac{T_g(h, s, a, s', r_h) AD_g(h, s)(a) + \eta}{\sum_{a'} (T_g(h, s, a', s', r_h) AD_g(h, s)(a') + \eta)} \right\} \end{aligned}$$

where

- *ADE is called the action distribution estimation function.*
- *s' is the state the environment transitions to.*
- *η is the smoothing parameter.*

4.4 Planning for optimal impact

Optimal behavior in the context of RepNet-MDPs may now be described. To simplify the notation in this as well as in later sections, a construct called *epistemic state* is defined hereafter.

Definition 4.7 (Epistemic state)**.** *The epistemic state θ_g of agent g is formally defined as a tuple*

$$\theta_g := \langle s, AD_g, Img_g \rangle$$

where

- *s is the current state of the environment.*
- *AD_g is the current action distribution of agent g.*
- *Img_g is the current image function of agent g.*
- *$\theta_g \in \Theta_g$, and Θ_g is called the epistemic state set. This set contains every possible combination of physical states of the environment, action distributions, and image functions of agent g.*

As with RepNet-POMDPs, an agent should perform actions according to the *perceived immediate impact* they have on the agent itself.

Definition 4.8 (Perceived immediate impact)**.** *The perceived immediate impact on agent g, written PI_g, is defined as*

$$PI_g(s, AD_g, a) := \frac{1}{|\mathcal{G}|} \left[\mathcal{I}(g, g, s, a) + \sum_{h \in \mathcal{G} \setminus \{g\}} \sum_{a' \in \mathcal{A}} \mathcal{I}(g, h, s, a') AD_g(h, s)(a') \right]$$

where

- *a is the action performed by agent g.*
- *AD_g is the current action distribution of agent g.*
- *The first term describes the immediate self-impact as a consequence of agent g performing action a.*
- *The second describes the expected immediate impact that the network, i.e., the remaining agents, has on agent g.*

Analogously to classical MDPs, a RepNet-MDP agent g strives to maximize its expected discounted perceived impact

$$\mathbb{E}[\sum_{t=0}^{k} \gamma^t PI_{g,t}] \tag{4.13}$$

where γ is the *discount factor* and $PI_{g,t}$ is agent g's perceived immediate impact at time-step t. This is accomplished by computing the optimal value function V_g, as defined in Definition 4.9.

Definition 4.9 (Value function)**.** *Let $V_g : \Theta_g \times \mathbb{N} \to \mathbb{R}$ be the optimal value function of agent g in a finite-horizon setting. It satisfies the optimality equations, which are defined as*

$$\begin{cases} V_g(\theta_g, k) := \max\limits_{a \in \mathcal{A}} \Big\{ PI_g(s, AD_g, a) + \gamma \sum\limits_{s' \in \mathcal{S}} T_g(g, s, a, s', r_g) V_g(\theta'_g, k-1) \Big\} & \forall \theta_g \in \Theta_g \\ V_g(\theta_g, 1) := \max\limits_{a \in \mathcal{A}} \Big\{ PI_g(s, AD_g, a) \Big\} & \forall \theta_g \in \Theta_g \end{cases}$$

where:

- $\theta_g = \langle s, AD_g, Img_g \rangle$
- $\theta'_g = \langle s', AD'_g, Img'_g \rangle = \langle s', ADE(g, s', AD_g, Img_g), IE(g, Img_g, \alpha, s, AD_g) \rangle$.
- $\gamma \in [0, 1]$ *is called the discount factor.*
- $r_g = REP_g(g, Img_g)$.

Definition 4.10 (Optimal policy)**.** *Let θ_g be the current epistemic state of agent g, and k the current horizon. The optimal action for agent g, written $\pi(\theta_g, k)$, is defined as*

$$\pi(\theta_g, k) := \arg\max_{a \in \mathcal{A}} \Big\{ PI_g(s, AD_g, a) + \gamma \sum_{s' \in S} T_g(g, s, a, s') V_g(\langle s', \phi'_g \rangle, k-1) \Big\}.$$

4.5 Online planning for RepNet-MDPs

Section 2.4 summarized several approximate MDP and POMDP solving techniques. The general principle of model-based online planning was described as the interleaving of two phases, the *planning phase*, in which the (PO)MDP performs a look-ahead search of a given depth D, starting at the current environment state, the goal being to determine the most suitable action, and the *execution phase*, in which this action is applied to the environment [23]. This section describes the adaptation of this approximate technique to the RepNet-MDP framework.

Let g be an agent deployed in an environment that can be in two physical states s_0 or s_1. The set of possible actions $\mathcal{A}$ comprises actions a_0 and a_1, and the environment is currently in state s_0. Agent g has current action distribution AD_g^0 and image function Img_g^0. The physical state of the environment, action distribution, and image function can be combined to form an *epistemic state* $\theta_g^0 = \langle s_0, AD_g^0, Img_g^0 \rangle$, as proposed in Definition 4.7.

Similarly to the MDP algorithm, the RepNet agent can construct a look-ahead tree of depth D starting at the current *epistemic state*. Figure 4.3 depicts a search space of depth $D = 1$. Every action in $\mathcal{A}$ first leads to the formation of a new branch, at the end of which is an AND-node (or *action* node) with the corresponding action. From each AND-node, every physical state is a potential future state and requires the formation of a new branch at the end of which is an OR-node (or *epistemic state* node) that contains the corresponding *epistemic state*. Consider the *epistemic state* in the lower-left corner of Figure 4.3, i.e. $\langle s_0, AD_g^1, Img_g^1 \rangle$. AD_g^1 and Img_g^1 are obtained by applying Definitions 4.6 and 4.4 respectively to the previous action distribution AD_g^0, the previous image function Img_g^0, and the new physical state s_0.

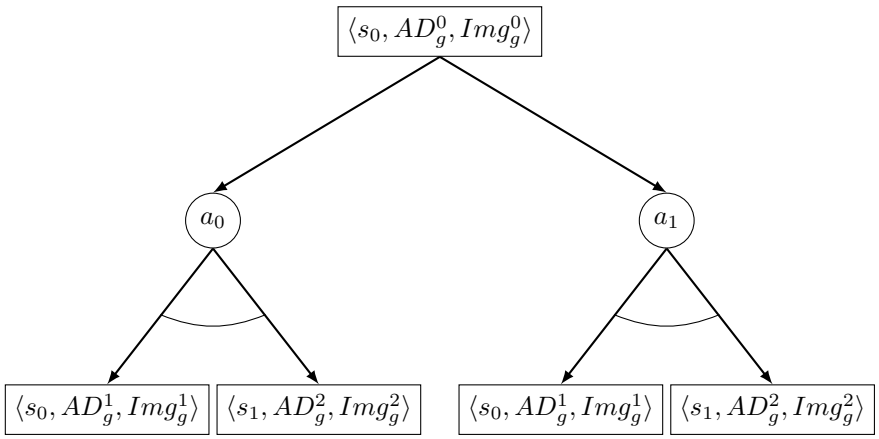


Figure 4.3: Look-ahead search space, depth $D = 1$ (RepNet-MDP)

After constructing the search space, an estimation of the value function needs to be back-propagated from the leaves to the root of the tree. An *admissible* heuristic

estimate of the true value function

$$h : \Theta_g \to \mathbb{R} \tag{4.14}$$

can be computed at the leaves by taking the base case of the Bellman equations for RepNet-MDPs:

$$h(\theta_g) = \max_{a \in \mathcal{A}} \Big\{ PI_g(s, AD_g, a) \Big\}. \tag{4.15}$$

The *epistemic state-action* values are then computed at the action nodes as follows:

$$q(\theta_g, a) = PI_g(s, AD_g, a) + \gamma \sum_{s' \in \mathcal{S}} T_g(g, s, a, s', r_g) h(\langle s', \phi'_g \rangle). \tag{4.16}$$

Non-leaf state nodes use the maximum of the *epistemic state-action* values of their children as their estimate of the value function, i.e.

$$h(\theta_g) = \max_{a \in \mathcal{A}} \Big\{ q(\theta_g, a) \Big\}. \tag{4.17}$$

4.6 General discussion: the reputation on a conceptual level

The present section briefly discusses the current definition of reputation on a conceptual level and serves as a comment on how it might have to be revised in future work.

The current definition of reputation (Definition 4.5), while arguably intuitive, can lead to undesirable results in regards to self-reputation, fault of its simplicity. This is best demonstrated with an example. Let A and B be two agents, and Img_A agent A's image function, such that:

$$Img_A(B, A) = -1, \quad Img_A(B, A) = -1, \tag{4.18}$$

meaning that in agent A's eyes, B has a poor image of A, and A has a poor image of B. By definition of the image function, $Img_A(i, i) = 1 \ \ \forall i \in \mathcal{G}$. As such the self-reputation of agent A is given by:

$$\begin{aligned} REP_A(A, Img_A) &= \frac{1}{2} \left[Img_A(A, B) \times Img_A(B, A) + Img_A(A, A) \times Img_A(A, A) \right] \\ &= \frac{1}{2} \left[1 + 1 \right] = 1 \end{aligned}$$

Agent A believes its own reputation to be good, despite the fact that it thinks B has a poor image of A. Given that A and B are the only agents in the network, this skewed perception of its own reputation could result in agent A performing sub-optimal actions in the network.

Chapter 5

Testing methodology

The initial RepNet framework, as described by *Rens et al.* [1], was developed as a novel extension to the standard POMDP framework, but was subsequently left unimplemented. Chapter 4 covered the simplification process of the RepNet framework to a fully observable setting, in an effort to reduce its intractability. The chapter furthermore discussed an alternative to exact planning such as *value iteration*, called *approximate online* planning, in the context of the simplified RepNet framework. More concretely, a look-ahead search algorithm can be used to approximate the real value function given the current *epistemic state* (Section 4.5). The goal of the present chapter is to present the experimental setup devised to test the framework's properties. Section 5.1 introduces the *scenarios* used to showcase the working principles of the RepNet framework. Section 5.2 then details the actual experimental setup. Finally, Section 5.3 discusses the settings for the parameters that are common across all experiments, while Section 5.4 covers the experiment-dependent parameters.

5.1 Test bed

This section outlines the *scenarios* that make up the entirety of the experimental setup. Each scenario was designed to showcase specific elements of the RepNet framework. Two types of scenarios are considered in this thesis: *Trading* and *Air-taxi* scenarios. All scenarios are used for different purposes and highlight different aspects of the framework.

5.1.1 Trading scenarios

A *trade transaction* is the activity by which a first party, called the *buyer*, wishes to buy a good from a second party, called the *seller*. A trade transaction materializes when the buyer is willing to buy, and the seller is willing to sell. The trading scenarios used in this thesis involve two to three agents and are designed to showcase the interaction between RepNet and non-RepNet agents. The first trading scenario, described in Example 5.1, involves two agents and is schematized in Figure 5.1. As

the scenario involves only a small number of states, its corresponding transition model is furthermore depicted in Figure 5.2.

Example 5.1 (Trade transaction between two agents)**.** *Let A and B be two agents. Agent A plays the role of the buyer, agent B the role of the seller. Agent A can engage in a trade with agent B, and B can accept or refuse the trade offer. Furthermore, agent A can, prior to making a trade offer, do a good deed in an effort to improve its image in the eyes of agent B.*

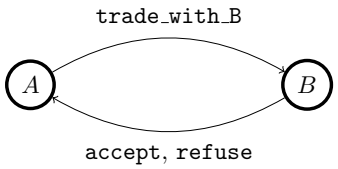


Figure 5.1: Trading scenario with two autonomous agents. Agent A can make trade offers, which agent B can accept or refuse.

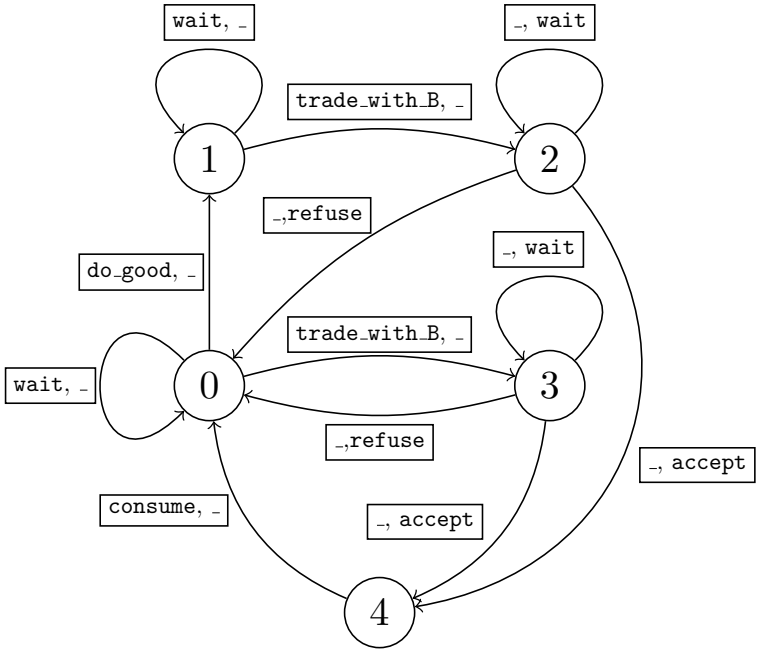


Figure 5.2: Transition model of the Trading scenario between agents A and B. Node 0 is the starting state, node 1 is the state following agent A's good deed, nodes 2 and 3 are states in which agent A has made an offer and is waiting for B's response, and nodes 4 is the *accept* state.

The second trading scenario, described in Example 5.2, involves three agents. The scenario is schematized in Figure 5.3.

Example 5.2 (Trading between three agents)**.** *Let A, B, and C be three agents. Each agent simultaneously plays the role of buyer and seller, and can thus engage in a trade with any other agent. Each agent can accept or refuse any trade offer made by any remaining agent.*

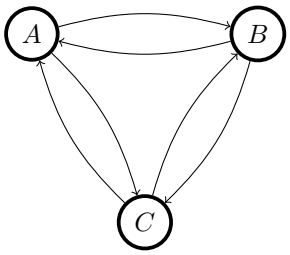


Figure 5.3: Trading scenario with 3 autonomous agents, each agent can `trade` with any remaining agent, `accept`, and `refuse` trade offers

5.1.2 Air-taxi (eVTOL) scenarios

Electric Vertical Take-off and Landing (eVTOL) aircraft are electric powered aircraft that can can take-off and land vertically, meaning they do not require a runway, and can hover at a given altitude [10, 54]. Recently, several big companies, including Boeing and NASA, Uber and Lilium GmbH have invested large amounts of money into the development of eVTOL aircraft designed to provide taxi services in and out of cities [55, 56]. These pilotless aircraft make up a network of agents that require air traffic management. Simplified air-taxi scenarios are used in the experimental setup, and are described hereafter. The first and simplest air-taxi scenario is given in Example 5.3 and involves two agents. The scenario is schematized in Figure 5.4.

Example 5.3 (Air-taxi battery level management)**.** *The airspace considered in this example is made up of 1 vertiport, i.e., airport for eVTOL aircraft, as well as a common flying space. The network is made up of 2 eVTOL aircraft, and each aircraft has 3 battery levels (high, medium, low). Upon landing at a vertiport, the battery of an aircraft is fully recharged. A central unit called an oracle, receives the position and battery level of each aircraft at every time-step, and immediately distributes the information to the entire network, thereby bringing the problem to a fully observable setting. At any point in time, only one aircraft may use the vertiport. Each agent's goal is to not run out of battery in the air. The selfish incentive would, therefore, appear to be to stay at the vertiport. This would, however, prevent the other agent from using that same platform, possibly when they most need to. The combined constraint, therefore, lies in the fact that no aircraft may crash at any point in time.*

The second air-taxi scenario, detailed in Example 5.4, involves 4 agents and figures as the most complex example. A scheme of the scenario is given in Figure 5.5.

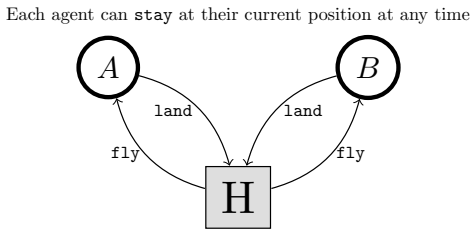


Figure 5.4: Air-taxi scenario with 2 autonomous agents

Example 5.4 (Air-taxi travel)**.** *The airspace considered in this example is designed as a grid that includes 4 vertiports. The network is made up of 4 eVTOL aircraft/agents, each aircraft assumed to have infinite battery. The goal of each agent is to travel to their assigned vertiport along a fixed itinerary, without crashing into any other aircraft. At the start of a run, these agents are randomly placed on the grid somewhere along their axis of travel. Each agent is fully aware of every agent's position on the grid, making the problem fully observable. The selfish goal resides in the fact that each agent wishes to reach their vertiport as quickly as possible. In doing so, the risk of two agents crashing into each other is increased substantially. Agents are, therefore, required to be cooperative, in spite of their selfish intentions [57].*

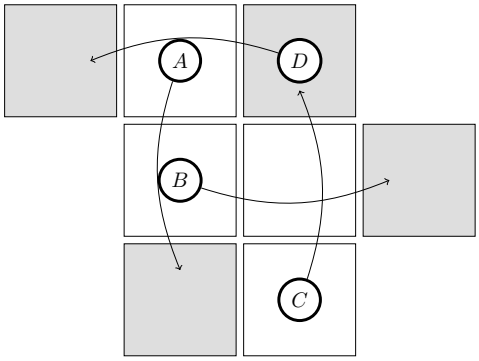


Figure 5.5: Air-taxi scenario with 4 autonomous agents

5.2 Experimental setup

The scenarios described in Section 5.1 are used in a series of experiments designed to test different aspects of the RepNet framework. Concretely, the experimental setup is divided into three parts, which are discussed in the present section. Section 5.2.1 deals with tests conducted to test the main properties of the RepNet framework. In Section 5.2.2, the ability of RepNet agents to quickly adapt to changes in other

agents' behavior is discussed. Finally, Section 5.2.3 extends the experimental setup to include more than two agents.

5.2.1 The RepNet framework features

The first part of the experimental evaluation concerns itself with testing the properties that distinguish RepNet-MDPs from other MDP-derived frameworks, and are thus conducted without the presence of a baseline framework. As a reminder, the properties of the RepNet framework include the notions of *action distribution*, *image*, *reputation*, and *directed actions*. The first series of experiments addresses the first three features, while the second one showcases the benefits and drawbacks of using directed actions. Both series of experiments take part in the environment described in Example 5.1.

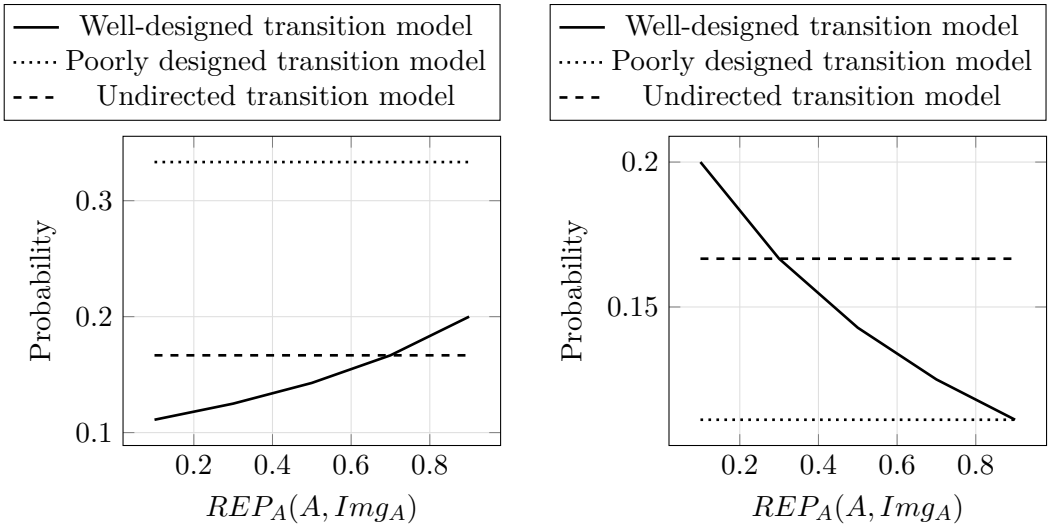
Testing action distribution, image, and reputation

In this series of experiments, Example 5.1 is used as follows: Agent A, the *buyer*, is managed by the RepNet algorithm. Agent B, the *seller*, is run by a simple algorithm that accepts or rejects trade offers made by agent A according to a set schedule. In particular, agent B is asked to reject trade offers for the 20 first time-steps, accept them for the 60 subsequent time-steps, and finally reject them for the last 20 time-steps. This experiment is averaged over 5 runs consisting of 100 time-steps. The variables tracked are the action distribution, image, and reputation of both agents *in the eyes of the RepNet agent (agent A)*. These variables are expected to change with time and, as such, produce tangible changes in the RepNet agent's behavior. The frequency at which agent A makes trade offers is therefore tracked as well.

Testing directed actions

The same series of experiments is then coupled with the concept of directed actions, as it was reimagined in Section 4.2. In particular, the action of agent A awaiting agent B's response to a trade offer is modeled as a directed action. An important part of using directed actions needs to be addressed here: if the transition probabilities of directed actions depend on the agent's reputation, then a suitable representation of the directed actions' transition model needs to be provided. Said differently, one could easily create a directed transition model that leads agent A to believe that it would be able to engage in a successful trade with agent B regardless of its current reputation. As such, the design of the directed transition model plays an important role in the agent's ability to perform well. Two directed transition models are therefore put to the test, a well-designed model that realistically reflects how the reputation of agent A may influence the willingness of agent B to accept A's trade offers and a poorly designed model with which agent A believes its reputation to have no impact on the outcome of a trade. Both models are schematized in Figure 5.6a. Figure 5.6a furthermore shows the directed transition model equivalent to the situation in which agent A does not make use of directed actions. In fact, if the transition probability of B accepting or refusing the trade offers is equal to $\frac{1}{|\mathcal{A}|}$ (where $|\mathcal{A}|$ is the number of actions at the disposal of agent B), regardless of agent

A's reputation, then reasoning using directed actions is equivalent to reasoning using undirected actions.



(a) Accepting trade offers

(b) Refusing trade offers

Figure 5.6: Perceived probability of agent B accepting and refusing the trade offers, as a function of the self-reputation of agent A.

5.2.2 Adaptability of the RepNet framework

The second part of the experimental evaluation tests the ability of RepNet agents to adapt to changes in the behavior of other agents in the network when adaptation needs to happen quickly. In the previous set of experiments, while agent B had a negative impact on agent A when it refused each trade offer, A could, in theory, turn around the situation, however bad it may be. Said differently, no state in the transition model represented a *trap* that is impossible to escape. In Example 5.3, the scenario used in this section, if one agent starts monopolizing the vertiport for more time-steps than are necessary, the other agent is expected to quickly change its strategy in a way that minimizes the chances of crashes occurring. Failing to do so could result in the second agent not being able to land when absolutely necessary. From an implementational point of view, a crash is represented as a *trap* state, and is, therefore, to be avoided at all costs.

To test the adaptability of RepNet agents, agent A is managed by the RepNet algorithm, while agent B is controlled by a simple algorithm that gradually increases the average time it spends on the vertiport. Agent A is expected to be able to pick up on these changes and adapt its own behavior accordingly. Specifically, agents A and B both start in the airspace, with their batteries fully charged. Agent B is designed to initially only make use of the vertiport when absolutely necessary, i.e., when its battery is low. Every 10 time-steps, Agent B's selfishness is increased

by a fixed factor $\xi \in [0, 1]$. Concretely, ξ represents the probability of agent B deciding to stay at the vertiport for the following time-step. As the battery is fully recharged immediately upon landing at the vertiport, deciding to stay on the ground is perceived as a selfish and unnecessary move by agent B. 2 series of 15 consecutive runs of 50 time-steps are conducted as follows:

- The first series of runs aims at tracking the number of crashes that occur as a function of the number of time-steps. As the selfishness of agent B increases, the likelihood of a crash does so as well.
- In the second series of experiments, agent B's behavior is altered slightly: it will never choose to remain on the ground when agent A's battery is low. This change makes it possible to uninterruptedly observe the behavioral changes of the RepNet agents across the 50 time-steps.

In an effort to provide an idea of how robust to changes in the behavior of agent B the RepNet agent is, the same experiment is conducted under the same circumstances with agent A now controlled by a standard *MDP look-ahead* algorithm (Section 2.4). Standard MDP-based algorithms are, by definition, incapable of adapting to behavioral changes of other agents, as these changes would have to be reflected in updates to the transition model. This MDP-based algorithm, therefore, serves as a baseline for comparison in this experiment.

As the *immediate reward function* $\mathcal{R}$ of MDP agents is different from the *perceived immediate impact* PI of RepNet agents, the reward function was defined to be equal to the initial perceived impact of the RepNet agent, i.e.:

$$\forall s \in \mathcal{S}, \forall a \in \mathcal{A} : \mathcal{R}(s, a) = PI_g(s, AD_g^0, a) \tag{5.1}$$

where $AD_g^0(h, s)(a) = \frac{1}{|\mathcal{A}|}\ \forall h \in \mathcal{G}, \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$.

The RepNet agent is expected to be able to sacrifice its selfish intentions, which would be to use the vertiport when in the air and leave the vertiport after landing, in an effort to avoid crashes.

5.2.3 Large-scale experiments

In the final part of the experimental evaluation, the ability of the RepNet algorithm to deal with an increased number of selfish agents is tested. To this end, Examples 5.2 and 5.4 from the test bed are used hereafter.

Large-scale trading experiment

Example 5.2 is used to verify the ability of a RepNet agent, say agent A, to manage its trades with the two remaining agents B and C (control agents), based not only on their behavior towards the agent of interest but also their behavior with each other. In fact, the action distribution, image, and reputation are general constructs, that is, they are not limited to tracking interactions that impact the RepNet agent *directly*.

The experiment can be divided into three parts: in the first part, agent B is asked to refuse each trade offer made by agent A, while agent C is expected to accept each trade offer coming from that same agent. This portion of the experiments assesses the ability of the RepNet agent (agent A) to accurately determine which agent it is more likely to successfully engage in trades with. In the second part, the roles are switched, and agent B accepts the trade offers, while agent C refuses them. This portion assesses the ability of the agent of interest to *unlearn* what it has learned and adapt its behavior accordingly. In the third and final part, the RepNet agent's actions are *blocked* from having any effect on the environment, preventing said agent from accumulating knowledge based on communications it is directly affected by. Furthermore, agents B and C are asked to engage in trades with each other. Agent B is asked to reject all trade offers, while agent C is asked to accept all trade offers. This portion of the experiment aims at testing the ability of the RepNet agent to draw conclusions on how it should act based on interactions it is not directly affected by. In fact, while the RepNet agent's actions have no effect on the environment, they are still recorded in this experiment and provide insight into which agent it is more likely to trade with. As the results described in Chapter 6 will show, Agent A is, in fact, unable to adapt its behavior in these instances.

Large-scale air-taxi experiment

A final experiment is conducted using Example 5.4 as the testing ground. The goal of this experiment is to demonstrate the ability of multiple RepNet agents to learn to cooperate with one another, despite their selfish intentions [57]. This experiment ties into the notion of *convergence* in *Multi-agent learning* (MAL), which was briefly discussed in Section 3.1. In particular, three of the four agents are controlled by their respective instances of the RepNet algorithm, while the last agent is controlled by a *control* algorithm. Concretely, the control agent is initially tuned to be entirely selfless, i.e., it will always prioritize the other agents reaching their destination *first* over reaching its own vertiport itself. Every 20 time-steps, the selfishness of the fourth agent is gradually increased by a constant factor $\xi \in [0, 1]$. ξ represents the probability with which the control agent decides to move forward in situations in which the selfless move would be to stay in place. The RepNet agents are expected to notice these behavioral changes and adapt their own behavior accordingly. In doing so, they, however, alert the remaining RepNet agents of their own behavior change, forcing them to take these changes into account as well. The experiment itself runs for 100 time-steps. The agents are initially randomly dropped in the environment somewhere along their axis of travel. Each time every agent reaches its designated vertiport, the successful run is recorded, and the agents are randomly placed back in the environment. Each time a crash occurs, the unsuccessful run is recorded, and the agents are likewise randomly placed back in the environment.

5.3 General parameters

5.3.1 RepNet parameters

The image update function $\mathcal{U}$

Given a current image value $v = Img_A(B, C)$ describing the image C has of B, according to A, a learning rate α, and the expected impact $i = ETI_A(B, C, s, AD_A)$ that C has on B when performing one of its actions in current the state s, $\mathcal{U}(\alpha, v, i)$ (Definition 3.5) returns the updated value $v' = Img'_A(B, C)$. Agent A's behavior is dictated by the image it believes each agent in the network to have of each other. Consequently, the image update function $\mathcal{U}$ has an important impact on A's performance. Two instantiations of the update function $\mathcal{U}$ are proposed by *Rens et al.* [1], namely the *difference* update and the *saturation* update.

- Difference update:

$$\mathcal{U}(\alpha, v, i) := \begin{cases} v + \alpha(1 - v)i & \text{if } i \geq 0 \\ v + \alpha(1 + v)i & \text{if } i < 0 \end{cases} \tag{5.2}$$

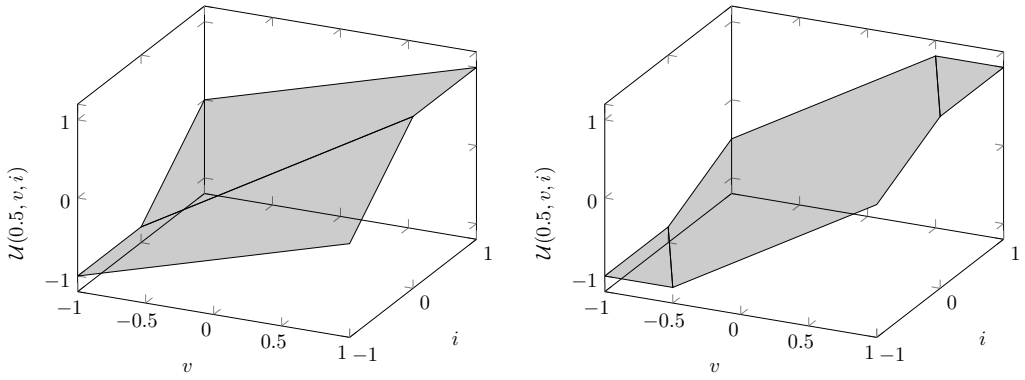
- Saturation update:

$$\mathcal{U}(\alpha, v, i) := \begin{cases} 1 & \text{if } v + \alpha i > 1 \\ -1 & \text{if } v + \alpha i < -1 \\ v + \alpha i & \text{otherwise} \end{cases} \tag{5.3}$$

The update surfaces are shown in Figure 5.7 for $\alpha = 0.5$. The experiments discussed in this chapter were all conducted using the difference update, as it produced the most consistent results across all experiments. Note that this does not mean that the saturation update is of inferior quality. In fact, a small portion of the experiments was indeed found to work better with the saturation update, which is indicative of the domain-dependence of the *image update function*. Nevertheless, in an attempt to keep the results consistent between tests, in particular those that make use of the same test bed, the same function was kept throughout the experimental evaluation.

Expected total impact trade-off parameter δ

The trade-off parameter $\delta \in [0, 1]$ in the expected total impact between agents h and i according to g, written $ETI(h, i, s, AD_g)$, trades off the relative importance of the impact due to agent h and the impact perceived by h (Definition 4.3). This parameter was set to 0.8 across all experiments, that is, whenever g's opinion on the image that i has of h ($Img_g(h, i)$) is updated, more weight will be given to the expected impact that i has on h in the current state, as opposed to the expected impact that h has on i. Said differently, the image that a first agent i has of another agent h is conditioned mainly by i's likeliness to impact (positively or negatively) h in the current state s.



(a) Difference update

(b) Saturation update

Figure 5.7: Image update functions proposed by *Rens et al.* [1]. The learning rate α is set to 0.5

Discount factor γ

The discount factor $\gamma \in [0, 1]$ in the RepNet framework's value function (Definition 4.9) determines the importance future time-steps should have on the agent's decision-making process. The closer it is to 1, the more important the future is, the closer it is to 0, the more important the present is. The value was set to 0.7 across all experiments.

Initial settings for the action distribution and the image

The RepNet agents deployed in a new environment are assumed to have no preconceived notions of other agents' behavior, reputation, and image of one another. This translates to the following initial settings for the action distribution and image for agent g:

$$\begin{gathered} AD_g^0(h, s)(a) = \frac{1}{|\mathcal{A}|} \quad \forall h \in \mathcal{G}, \forall s \in \mathcal{S}, \forall a \in \mathcal{A}, \\ Img_g(h, i) = 0 \quad \forall h, i \in \mathcal{G}, h \neq i. \end{gathered} \tag{5.4}$$

5.3.2 MDP parameters

Discount factor γ

The discount factor $\gamma \in [0, 1]$ in the MDP framework's value function plays the same role as in the RepNet framework and was likewise set to 0.7 across all experiments involving an MDP agent.

Look-ahead depth D

The scenarios in which an MDP agent is used feature a look-ahead depth $D = 3$, as the same depth is used for the RepNet agents in said scenarios.

5.4 Experiment-specific parameters

Several parameters were tailored to each experiment separately. This section aims at summarizing the settings for these parameters.

As such, D is defined as the look-ahead depth of the online search algorithm used by RepNet agents (Section 4.5). The higher it is, the more informed the RepNet agent's decisions can be. This comes at the drawback of increased computation times. Section 5.3.1 reviewed the concept of image update function. This function takes a parameter $\alpha \in [0, 1]$ called the learning rate that determines how quickly the image function is updated. η is the *Laplace* smoothing parameter and was introduced in Section 4.1 in an effort to smooth out the action distribution, especially in cases where the transition models are deterministic.

The only parameter that has not been discussed thus far is the *exploration-exploitation trade-off parameter* ϵ. In order for the RepNet agent to learn, sometimes it must be forced to try out actions even if the associated value is significantly lower than that of another action. In fact, in the trading examples between agents A and B, if A realizes B's unwillingness to accept the trade offers, the value associated with the `trade_with_B` action decreases. If now B changes its strategy and accepts all trade offers, A can not know of this change unless it attempts to trade with B occasionally. This concept is known in the literature as the *exploration-exploitation trade-off* [58]. *Exploration* refers to the phase in which an agent performs actions that it would otherwise consider to be sub-optimal. *Exploitation* refers to the phase in which an agent utilizes its current policy to move about in the environment. Several strategies for balancing *exploration* and *exploitation* exist, some simpler, others more complex. As the primary goal of this thesis does not lie in evaluating the benefits and drawbacks of different strategies, the arguably simplest strategy, called ϵ-greedy, was used [59]. The ϵ-greedy strategy consists in selecting a random action $a \in \mathcal{A}$ at each time-step with a fixed probability $\epsilon \in [0, 1]$, instead of selecting the action $a^\star$ that would, given the current policy, be considered optimal. The ϵ-greedy strategy was solely utilized in the trading experiments, as using it with the air-taxi experiments can easily lead to crashes, and thus tarnish the significance of the results. The settings for each parameter and experiment are summarized in Table 5.1.

Trading between 2 agents	Parameters			
	D	ϵ	α	η
RepNet agent A	3	0.2	0.8	0.1
Trading between 3 agents	**Parameters**			
	D	ϵ	α	η
RepNet agent A	3	0.2	0.8	0.1
Air-taxi with 2 agents	**Parameters**			
	D	ϵ	α	η
RepNet agent A	3	0	1	0.15
Air-taxi with 4 agents	**Parameters**			
	D	ϵ	α	η
RepNet agent A	2	0	1	0.1
RepNet agent B	2	0	1	0.1
RepNet agent C	2	0	1	0.1

Table 5.1: Settings for the experiment-specific parameters, for each of the experiments.

Chapter 6

Results

This chapter covers the findings of the experiments described in Chapter 5. These experiments aim at quantifying the performance of the RepNet algorithm in both simple and more complex scenarios while demonstrating its strong points, as well as shortcomings. The chapter follows the following structure: in Section 6.1, the fundamental properties that differentiate the RepNet framework from other MDP-derived frameworks are presented. Section 6.2 covers the ability of the algorithm to quickly adapt to changes in the behavior of other agents. Section 6.3 presents the results related to the two large-scale experiments. Finally, Section 6.4 summarizes the results.

6.1 The properties of the RepNet framework

In this section, the concepts of *action distribution*, *image*, *reputation*, and *directed actions*, as well as these concepts' effects on RepNet agents' behavior, are quantified. The results of two sets of experiments are presented hereafter. The first set of experiments covers *action distribution*, *image*, *reputation* in the absence of *directed actions*, while the second set covers the same properties in their presence.

6.1.1 Testing action distribution, image, and reputation

The *action distribution*, *image*, and *reputation* of agent A are tracked throughout the experiment, which is made up of 5 runs, of which the average one is analyzed hereafter. As such, Figure 6.1a shows the evolution of agent A's action distribution for target agent B in state $s = 2$ (the state in which B is aware of the trade offer made by agent A, which has done a good deed prior to the offer, see Figure 5.2). Figure 6.1b displays the image B has of A and vice versa, according to A. Figure 6.2a shows the reputation of A and B, according to A. Finally, Figure 6.2b displays the tangible effects of these variables on A's behavior. Concretely it shows the evolution of the frequency at which A makes trade offers. Note that to maintain clarity, a shorthand notation for the reputation is used in this chapter, that is,

$$REP_g(h, Img_g) = REP_g(h) \;\; \forall g, h \in \mathcal{G}. \tag{6.1}$$

In the first 20 time-steps, B refuses each trade offer. Agent A is able to pick up on this via the action distribution. As a consequence, it quickly reduces the frequency at which it attempts to trade with B. A's image of B, and, by extension, B's reputation (Definition 4.5) quickly drop alongside. Interestingly, A's self-reputation decreases in a similar fashion. In fact, B refusing to trade with A can be explained as having two effects on A's perception of the network, both following from the current mathematical definition of reputation (Definition 4.5):

- B is an unreliable agent and should, therefore, have a poor reputation. The image A has of B decreases, and consequently, so does B's reputation.
- B refuses to trade with A because it believes A to be unreliable. A must adapt its self-reputation based on the evidence it is provided with. In this case, the evidence points towards the fact that A might be seen as unreliable in the network, which only includes B.

The similarity between A's and B's reputations can be further explained by the fact that the network is made up of only two agents. As such, the evidence provided to A is insufficient for it to make up its mind as to the cause of B's changes in behavior.

In the 60 following time-steps, B is asked to change its behavior and accept each trade offer. Hesitant at first, A gradually increases the frequency at which it attempts to trade with B. B's reputation, as well as A's self-reputation, are positively impacted by B's change of behavior. The same two-effect explanation can be applied to explain the increase of both agents' reputations.

Finally, B reverts back to its old behavior during the final 20 steps. Again, Agent A is able to pick up on this change and adapt its behavior accordingly.

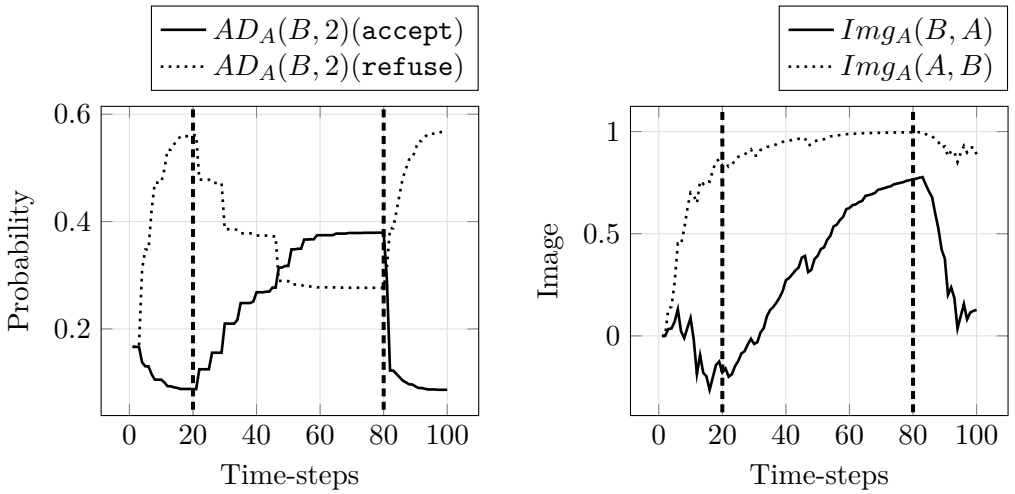
6.1.2 Testing directed actions

The same set of experiments was conducted again, now using the concept of *directed actions*. As discussed in Section 5.2.1, the design of the directed transition model has a significant impact on the performance of the RepNet agent. Two directed transition models of vastly different quality were proposed, and the results are described hereafter.

Well-designed directed transition model

Analogously to the previous section, Figure 6.3a shows the evolution of agent A's action distribution for agent B in state $s = 2$, Figure 6.3b displays the image B has of A and vice versa, Figure 6.4a shows the reputation of A and B, and Figure 6.4b displays the tangible effects of these variables on A's behavior.

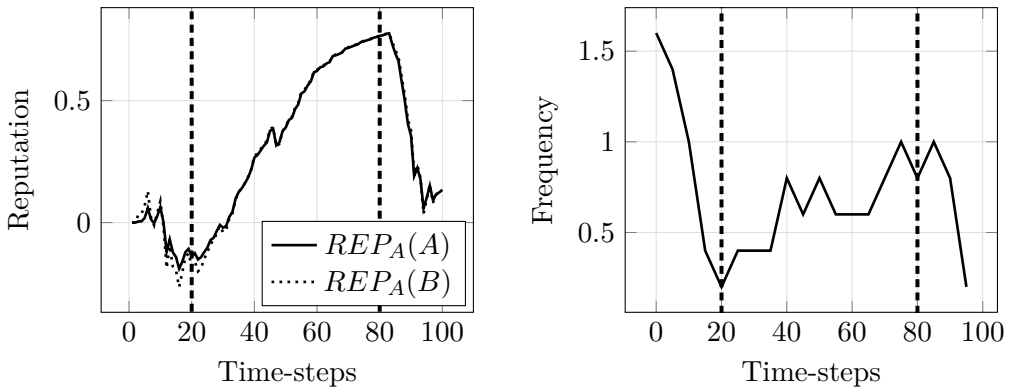
The well-designed transition model noticeably improves the performance of the RepNet agent. While the trajectories showcase the same key elements as in the previous section, the pace at which agent A is able to adapt has greatly improved. As B refuses the offers during the 20 first time-steps, its reputation decreases along with A's self-reputation. The directed transition model was designed



(a) Probability of B accepting and refusing A's trade offers, according to A

(b) Image agents A and B have of each other, according to A

Figure 6.1: Undirected actions: Evolution of the action distribution and image function, according to agent A.



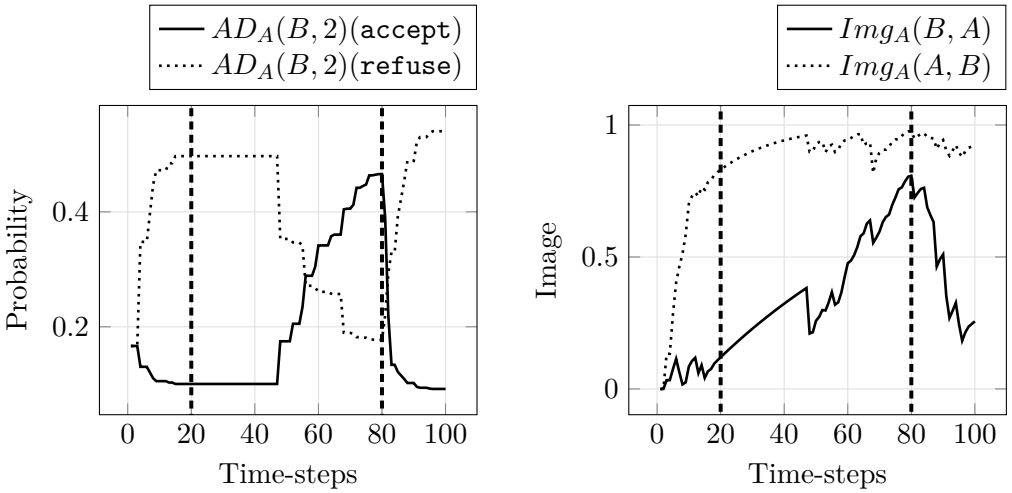
(a) Reputation of agents A and B, according to A

(b) Frequency of the trade offers made by A, measured in 5 time-step intervals

Figure 6.2: Undirected actions: Evolution of the reputation according to agent A, and frequency of trade offers made by agent A.

such that agent A believes that its reputation must be good for B to be willing to trade with A (Figure 5.6a and Figure 5.6b). As such, A's relatively poor-in-comparison self-reputation has an immediate effect on the *value* it associates with action `trade_with_B` during the look-ahead search. Said differently, it quickly becomes more *valuable* to pick a different action over `trade_with_B`. This explains the rash changes in A's behavior at the start. Incidentally, agent A is able to quickly increase the rate at which it attempts to trade with B once it notices B's willingness

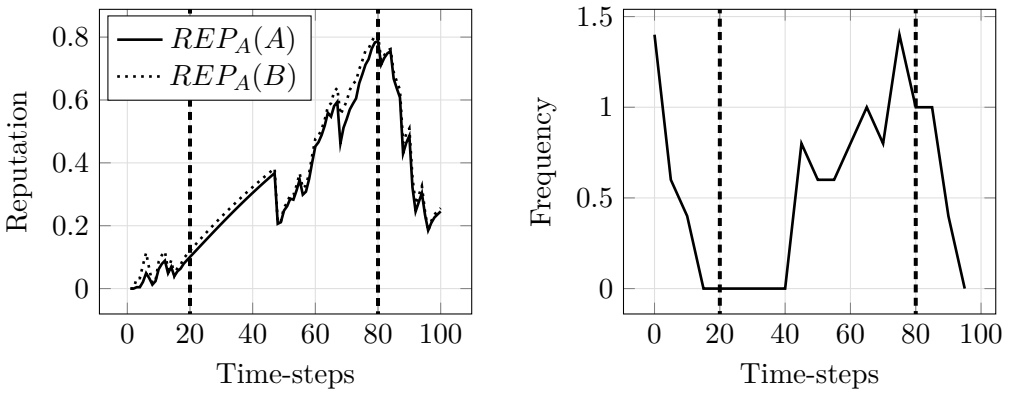
to trade. The average rate ends up topping off higher than without the use of directed transitions.



(a) Probability of B accepting and refusing A's trade offers, according to A

(b) Image agents A and B have of each other, according to A

Figure 6.3: Well-designed directed transition model: Evolution of the action distribution and image function, according to agent A.



(a) Reputation of agents A and B, according to A

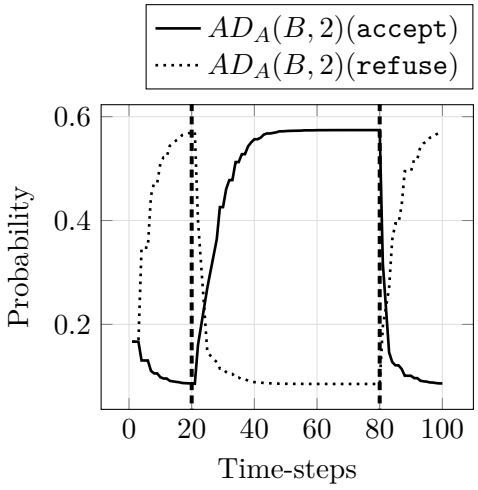
(b) Frequency of the trade offers made by A, measured in 5 time-step intervals

Figure 6.4: Well-designed directed transition model: Evolution of the reputation according to agent A, and frequency of trade offers made by agent A.

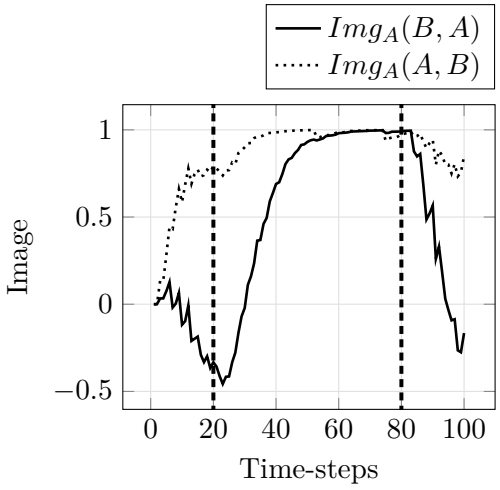
Poorly designed directed transition model

The same set of experiments was conducted a third time with a poorly designed *directed transition model*. As discussed in Section 5.2.1, the presently used directed transition model aims at making agent A believe that agent B will want to trade with A in spite of a poor reputation. Figure 6.5a displays the evolution of agent A's action distribution for agent B, Figure 6.5b shows the image B has of A and vice versa, Figure 6.6a shows the reputation of A and B, and Figure 6.6b displays the tangible effects of these variables on A's behavior.

The present experiment demonstrates that the presence of directed actions does not guarantee that the RepNet agent will perform better. In fact, a poorly designed directed transition model mitigates many of the benefits initially obtained by introducing the concepts of *action distribution*, *image*, and *reputation*. In fact, the *action distribution*, *image*, and *reputation* trajectories show that although the algorithm is capable of noticing the same changes in B's behavior, the fact that it believes its reputation to be of little importance prevents it from adapting its own behavior. Consequently, agent A has become slow at learning from other agents' behavior.



(a) Probability of B accepting and refusing A's trade offers, according to A

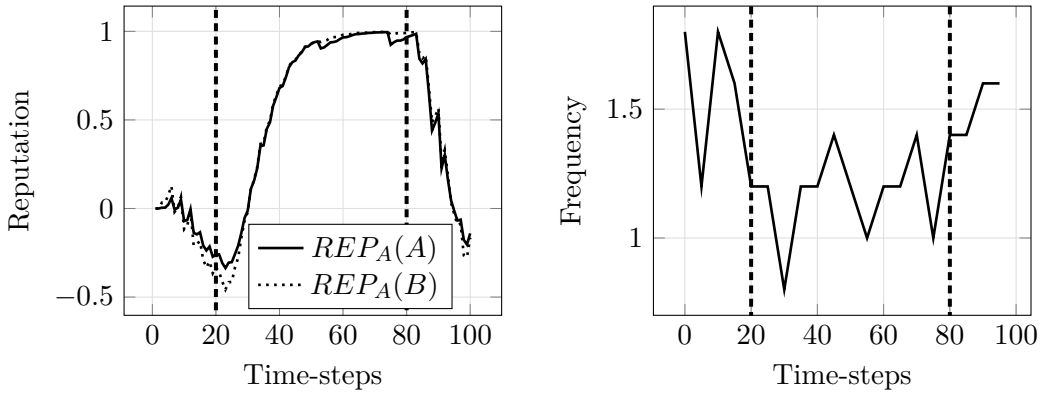


(b) Image agents A and B have of each other, according to A

Figure 6.5: Poorly designed directed transition model: Evolution of the action distribution and image function, according to agent A.

6.2 Adaptability of the RepNet framework

As discussed in Section 5.2.2, the present experiment aims at demonstrating the ability of the RepNet agent to adapt to changes in the behavior of another agent *in the presence of imminent danger*. In this case, the danger comes in the form of



(a) Reputation of agents A and B, according to A

(b) Frequency of the trade offers made by A, measured in 5 time-step intervals

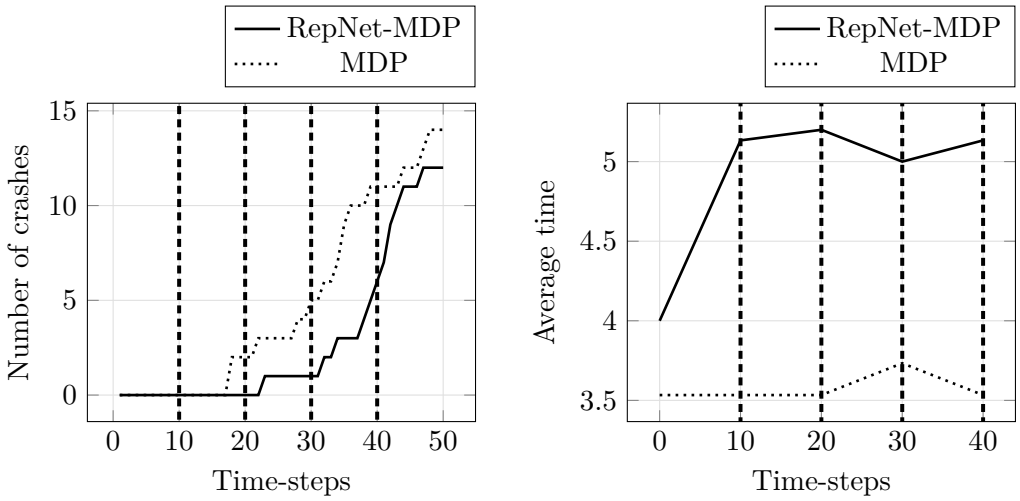
Figure 6.6: Poorly designed directed transition model: Evolution of the reputation according to agent A, and frequency of trade offers made by agent A.

aircraft crashes. Concretely the RepNet agent (agent A) is expected to be willing to sacrifice its own *selfish* intentions to ensure no aircraft crashes occur. As a reminder, the selfishness of the second agent is increased every 10 time-steps. To show that the adaptations made by the RepNet agent are sound, the performance of the RepNet agent is compared to the performance of a classic MDP agent, i.e., an agent that does not adapt its behavior to fit the other agent's.

The first experiment of 15 runs was conducted to assess the ability of the RepNet agent to effectively reduce the number of crashes. Figure 6.7a shows the evolution of the number of runs in which a crash has occurred as a function of the time-steps, for both the RepNet and MDP agents. It should be noted that these crashes can be inevitable. In fact, as the selfishness of the second agent increases, the probability of it deciding to stay at the vertiport when the RepNet agent's battery is low increases alongside. As such, an unfortunate sequence of steps, in which the second agent does not allow the RepNet agent to land when necessary will result in a crash regardless of the RepNet agent's actions.

The second experiment of 15 runs was conducted under the same circumstances, with the exception that the second agent never chooses to stay at the vertiport when the RepNet agent has little battery left. This makes it possible to uninterruptedly observe the adaptations made by the RepNet agent. Figure 6.7b displays the time the RepNet agent spends at the vertiport as a function of the time-steps.

The MDP agent's performance serves as a baseline. In particular, the graphs show that the RepNet agent is, on average, effectively able to *delay* the time at which a crash occurs (Figure 6.7a) by increasing the time it spends at the vertiport (Figure 6.7b). It thereby sacrifices its *most* selfish intentions, which are to leave the vertiport when on the ground and leave the airspace when in the air, in an effort to avoid crashes.



(a) Number of runs in which a crash has occurred, as a function of the time-steps

(b) Average time agent A spends on the ground, averaged over 10 time-steps

Figure 6.7: Number of crashes and average time spent at the vertiport by agent A.

6.3 Large-scale experiments

To provide an initial understanding of the fundamental features that make up the RepNet framework, the number of agents was kept to 2 thus far. Increasing the number of agents aims at investing the robustness of the RepNet algorithm in the face of uncertainty. It also makes it possible to highlight potential deficiencies of the framework. As such, two large-scale experiments were set up to stress-test the multi-agent capabilities of the RepNet framework. Section 6.3.1 investigates the ability of RepNet agents to draw conclusions on which actions to take based on other agents' actions that do not directly affect said RepNet agent. Section 6.3.2 explores the ability of several RepNet agents to cooperate in spite of their *selfish* nature.

6.3.1 Large-scale trading experiment

The results of the 3-agent trading experiment are included in this section. The RepNet agent will be referred to as agent A hereafter. The results have been averaged over 10 runs. Figure 6.8a shows the evolution of the reputations of agents B and C. Figure 6.8b displays the evolution of the probabilities of agents B and C accepting trade offers from agent A. Finally, Figure 6.9 shows the average action taken by agent A in the state in which the agents are allowed to make an offer.

Agent B is told to refuse, and agent C to accept, each trade offer during the first 33 time-steps. Expectedly, and in accordance with the results obtained in Section 6.1, agent A is able to pick up on the other agents' behavioral habits as they directly affect it. As a result, the reputation of B decreases, while the reputation of C increases. All the while, agent A chooses to conduct the majority of its trades with C.

The following 33 time-steps reverse B's and C's roles. Similarly, agent A is able to adapt its behavior accordingly and ends up trading most of the time with B. The reputation of B has increased, while the reputation of C has decreased.

The interesting observations occur during the last 33 time-steps. Agents B and C are tasked with trading with one another. B is asked to refuse all trade offers, while C is asked to accept all trade offers. Here, Agent A's actions have been blocked from having any effect on the environment. This is done in an effort to avoid that the RepNet agent can obtain direct feedback through its actions. It is thereby left with observing the trades that occur between agents B and C. Unfortunately, the results show that even though the reputation of B decreases and the reputation of C increases according to A, the RepNet agent will still prefer to trade with B. In other words, as long as agent A has not received direct confirmation from B that it will refuse its *own* offers, it will keep trying to trade with B over Agent C.

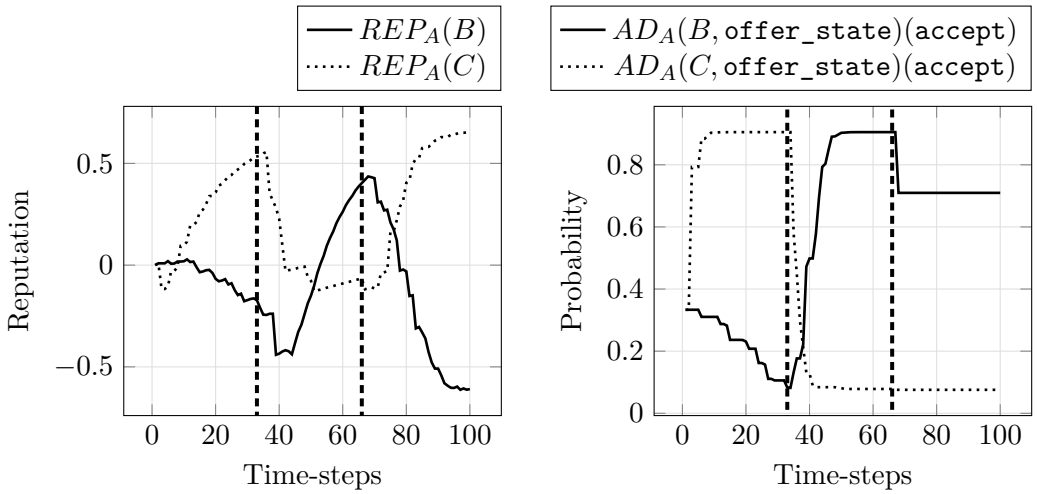
The reason for this behavior is as follows: When agent A performs a directed action targeted at another agent, say B, the result of the trade is, in the eyes of A, conditioned only by its own reputation (Section 4.2). Likewise, the probability of B accepting (or refusing) A's trade offer, according to A, is updated only through the direct experience it has with B (as per Definition 4.6, and as can be seen in Figure 6.8b). The current implementation of the RepNet framework makes it impossible for Agent A to learn from interactions it is not *directly* affected by.

Whether this absence is to be seen as a shortcoming is up for debate. In fact, it may be desirable for a RepNet agent to require direct information before making up its mind, in an effort to avoid ill-informed decision-making. As an illustrative example, the fact that agent B is refusing each trade offer from C might just mean that B believes C to be unreliable; it might very well still be willing to trade with Agent A. On the other hand, this narrow-minded view on reputation noticeably mitigates the importance of said reputation as a whole.

6.3.2 Large-scale air-taxi experiment

In this final experiment, multiple RepNet agents are deployed in the same environment. In particular, 3 RepNet agents and 1 agent controlled by a simple algorithm have to find a way of reaching their goal destination (*selfish* goal), whilst making sure crashes are avoided. As the *non*-RepNet agent's selfishness increases every 20 time-steps, it becomes all the more important for each RepNet agent to find a balance between their most *selfish* goal and the *well-being* of the network as a whole. As discussed in Section 5.2.3, agents are redeployed randomly anytime they either all reach their goal destination of a crash occurs. The end goal for the RepNet agents is to minimize the number of crashes and simultaneously maximize the number of successful runs.

Figure 6.10a depicts the frequency at which crashes occur, as a function of the time-steps. Figure 6.10b shows the frequency at which the four agents manage to collectively reach their respective goal destinations. The graphs provide important insights into the ability of multiple RepNet agents to collectively adapt to the unpredictability of the *non*-RepNet agent's behavior. As such, each increase in the selfishness of this agent is met by a peak in crashes and a drop in successful runs



(a) Reputation of agents B and C, according to A

(b) Probability of agents B and C accepting trade offers from agent A, according to A

Figure 6.8: Evolution of the reputation of agents B and C, and of the action distribution, according to agent A.

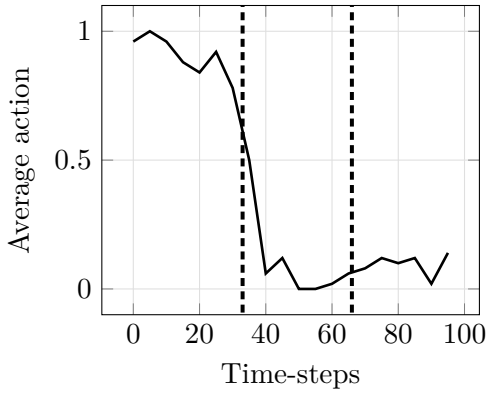
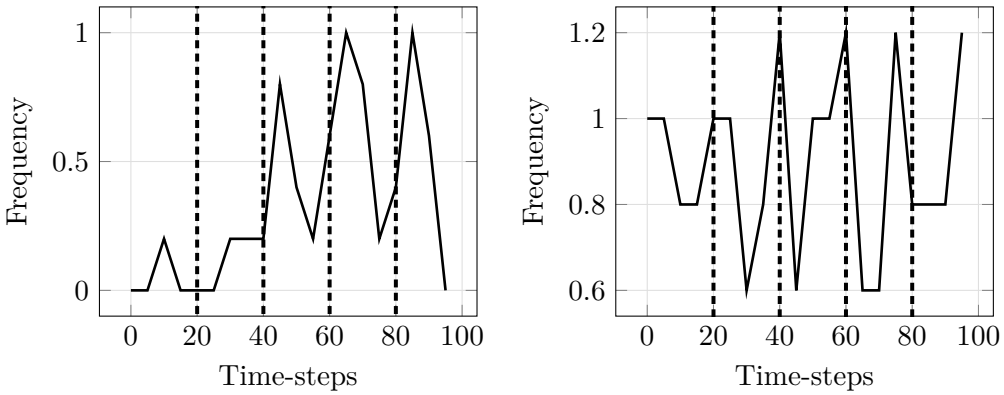


Figure 6.9: Average action taken by agent A, averaged over 5 time-steps. Action $a = 0$ corresponds to trading with agent B, action $a = 1$ corresponds to trading with agent C.

shortly after. This is to be expected as the RepNet agents can not predict the *non*-RepNet agent's behavioral changes. Following each peak, the RepNet agents manage to each find a new stable policy that allows each agent to reach their goal destination safely. The frequency of successful runs increases until the next increase in the selfishness of the *non*-RepNet agent. The ability of the RepNet agents to collectively *converge* to a stable situation is not trivial. Section 3.1 briefly discussed the convergence problem inherently present in *Multi-agent learning* (MAL). While *Single-agent learning* (SAL) algorithms are often able to provide certain convergence

guarantees, the same can not be said of MAL algorithms [7, 8]. Concretely, the convergence of one agent's policy is a function of the behavior of all other agents in the network. This makes it all the more interesting that given a properly tuned set of parameters, RepNet agents are able to come to a mutual agreement that benefits the entire network. It should be noted here that the topic of convergence in MAL is vastly more complex than can be summarized in a single experiment [44, 45, 7, 8]. The present experiment merely provides encouraging results as to the viability of the RepNet framework.



(a) Frequency of crashes

(b) Frequency of successful runs

Figure 6.10: Frequency of crashes and successful runs during the 4-agent air-taxi experiment, averaged over 5 time-steps.

6.4 Summary of the results

The first part of the experimental setup has shown the RepNet algorithm to be able to leverage the notions of *action distribution*, *image*, and *reputation* to aid in its decision-making process, in the presence of a second agent. The concept of *directed actions* was shown to be effective, provided that the corresponding *directed transition model* was carefully designed. Failing to do so resulted in the RepNet agent barely being able to learn. The second part demonstrated the ability of the RepNet to partially sacrifice its *selfish* goal in an effort to prevent crashes from occurring. The large-scale trading example highlighted a potential shortcoming of the current version of the RepNet algorithm. In fact, the algorithm is unable to learn from experiences that do not affect it directly. Finally, the large-scale aircraft example showcases the possibility of several RepNet agents to be deployed in the same environment and having them cooperate with one another. It provides early results as to the convergence properties of well-tuned RepNet agents.

Chapter 7

Conclusion

This thesis has examined the viability of a multi-agent extension to the POMDP framework called RepNet, and first introduced by *Rens et al.* [1]. In this chapter the main findings are summarized, the research questions are briefly reviewed, and future work is suggested.

7.1 Thesis summary

In Chapter 2, the relevant background for the thesis was summarized. In particular, Markov Decision Processes and their extension to partially observable environments were presented. The well-known *Value Iteration* algorithm was then reviewed for both frameworks. Lastly, several online and offline solving techniques for these frameworks were discussed.

In Chapter 3, the previous work closely tied to this thesis was presented. Early multi-agent MDP extensions, namely Multi-agent MDPs (MMDPs) and Decentralized POMDPs (Dec-POMDPs) were first summarized. These frameworks deal with situations in which selfishness is not considered an issue. A first multi-agent, MDP-derived framework in which selfishness is explicitly dealt with, namely the interactive POMDP (I-POMDP) framework, was then discussed. Finally the framework of interest, called RepNet-POMDP and published by *Rens et al.* [1], was summarized. This chapter ends with a few identified issues of the initial RepNet framework.

In Chapter 4, the reduction of the RepNet-POMDP framework to a fully observable setting called RepNet-MDP was first presented. The goal of the reduction resided in the alleviation of the intractability of exact planning. Solutions to the issues mentioned in the previous chapter were then elaborated on. Finally, online planning was introduced in the context of RepNet-MDPs in an effort to further reduce the computational requirements of the framework.

In Chapter 5, the testing methodology and scenarios were explained in detail. The test bed consists of examples taken from two domains, namely the trading and air-taxi domains. The experiments were constructed around those scenarios with the intent of showcasing the strengths and shortcomings of the RepNet framework.

In Chapter 6, the results of the experimental setup were presented and discussed. In particular, the adaptability of RepNet agents was tested under different conditions. While the agents showed to be capable of adapting to situations that directly affected them, they were unable to draw conclusions on behavior from other agents that did not directly concern them.

7.2 Review of the research questions

Research question 1: *Are the concepts of action distribution, image, and reputation effective in practice?*
The notions of *action distribution*, *image*, and *reputation* were shown to allow RepNet agents to effectively monitor other agents' past behavior and reliability, and draw sound conclusions on the best course of action in cases where the RepNet agents were directly affected by the other agents' behavior.

Research question 2: *Is the concept of directed actions effective in practice?*
The concept of *directed actions*, as is was imagined by *Rens et al.*, was shown to be unable to describe the rules of the environment in which the RepNet agents are deployed. An alternative to this concept, in which directed transitions are defined to explain the RepNet agent's *subjective* impression of the impact its reputation has on the interactions with other agents, was proposed. The reimagined concept of directed actions proved to improve the adaptability of the RepNet agents, provided that the directed transition models were designed so as to realistically reflect the extent to which the RepNet agents' reputation affects their ability to communicate with other agents.

Research question 3: *Are RepNet agents capable of drawing general conclusions on the interactions between the agents that are apart of the environment?*
The current version of the RepNet framework makes it impossible for a RepNet agent to adapt its behavior in the event of interactions that it is not directed affected by, even when these interactions are indicative of a given agent's poor intentions.

Research question 4: *Are RepNet agents capable of being cooperative in spite of their selfish intentions?*
Properly tuned RepNet agents were shown to be able to sacrifice their most selfish intentions in an effort to maintain a general level of well-being of the network. In particular, a network of 3 RepNet agents was shown to be able to find common ground in the face of uncertainty and selfishness from a fourth party.

7.3 Limitations and future work

This section discusses the limitations of this thesis, and provides suggestions for future work.

The current definition of *directed* transitions could be extended to incorporate the reputation of agents other than the RepNet agent. In fact, the experimental results showed that the RepNet agent is incapable of adapting its behavior to situations which it is not directly affected by. Including the reputation of the agent at the receiving end of a directed action in the *directed* transition model is likely to lead to better-informed decision-making, provided that the *directed* transition model is designed so as to realistically reflect the impact that the receiving agent's reputation has on the outcome of the interaction.

To reduce the intractability of the framework, the framework was reduced to a fully observable setting. As many real-world problems do not benefit from full observability, it would be interesting to bring the RepNet framework, together with the corrections made, back to a partially observable setting.

Several techniques for speeding up computations can be added to the look-ahead algorithm. While this thesis did not focus on the performance gains that could be achieved while still offering the same quality of solutions, it should be noted that the number of tests conducted was limited by the time taken by each experiment. Simple pruning techniques would help in alleviating the computational requirements.

While the features and adaptability of the RepNet framework were rigorously tested on their own, a comparative analysis of the performances of several state-of-the-art algorithms may have to be conducted in the future. However, as it currently stands, much MDP-derived research has shifted towards model-free solutions, as real-world problems can easily become too difficult for transition models to be designed by any one person. In fact, the state space of an environment grows exponentially with the number of domain features [39]. A sub-branch of MAL has experienced a jump in popularity when *DeepMind* released the paper "*Playing Atari with Deep Reinforcement Learning*", in which they use *deep learning* on a convolutional neural network to approximate the *function* of the environment [60]. A potentially interesting direction in which further research could be conducted is the development of a *hybrid* framework in which the transition model is learned through trial and error using state-of-the-art (*neural network*-based) techniques (model-free part), and in which the concepts of action distribution, image, and reputation assist the agent in its decision-making process (RepNet part).

Alternatively, an interesting way in which one could keep the model-based nature of the current RepNet framework and use it to solve real-world problems consists in introducing elements of relational logic. In 2001, *Boutilier et al.* [39] introduced relational MDPs (RMDPs). From a *logic programming* point of view, a state space is hereby defined by a collection of relations, while a state is an *interpretation* of this collection [41]. Transition models and reward schemes are then represented by *probabilistic rules* [12].

Finally, Section 6.3.2 provided early results of the convergence properties of RepNet agents. The topic of convergence in Multi-agent learning is significantly more complex than can be summarized in a single experiment. An interesting direction for future research could include the study of the convergence properties of the RepNet framework.

Bibliography

[1] Rens G, Nayak A, Meyer T. Maximizing Expected Impact in an Agent Reputation Network - Technical Report. CoRR. 2018;abs/1805.05230. Available from: http://arxiv.org/abs/1805.05230.

[2] Russell S, Norvig P. Artificial Intelligence: A Modern Approach. 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall Press; 2009.

[3] Boutilier C. Planning, Learning and Coordination in Multiagent Decision Processes. In: Proceedings of the Sixth Conference on Theoretical Aspects of Rationality and Knowledge. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 1996. p. 195–210. Available from: http://dl.acm.org/citation.cfm?id=645875.671730.

[4] Becker R, Zilberstein S, Lesser V, Goldman C. Solving Transition Independent Decentralized Markov Decision Processes. J Artif Intell Res (JAIR). 2004 07;22:423–455.

[5] Bernstein DS, Zilberstein S, Immerman N. The Complexity of Decentralized Control of Markov Decision Processes. CoRR. 2013;abs/1301.3836. Available from: http://arxiv.org/abs/1301.3836.

[6] Doshi P, Gmytrasiewicz PJ. A Framework for Sequential Planning in Multi-Agent Settings. CoRR. 2011;abs/1109.2135. Available from: http://arxiv.org/abs/1109.2135.

[7] Busoniu L, Babuska R, De Schutter B. A Comprehensive Survey of Multiagent Reinforcement Learning. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on. 2008 04;38:156 – 172.

[8] Abbeel P. Learning for Robotics and Control - Value Iteration, CS294-40, University of California, Berkeley. 2008. Available from: https://inst.eecs.berkeley.edu/~cs294-40/fa08/scribes/lecture2.pdf.

[9] Aberer K, Despotovic Z. On Reputation in Game Theory Application on Online Settings. 2019 12.

[10] Bacchini A, Cestino E. Electric VTOL Configurations Comparison. Aerospace. 2019 02;6:26.

[11] Kapoor S. Multi-Agent Reinforcement Learning: A Report on Challenges and Approaches. CoRR. 2018;abs/1807.09427. Available from: http://arxiv.org/abs/1807.09427.

[12] Nitti D, Belle V, De Laet T, De Raedt L. Planning in hybrid relational MDPs. Machine Learning. 2017 Dec;106(12):1905–1932. Available from: https://doi.org/10.1007/s10994-017-5669-x.

[13] Williams JD. Partially Observable Markov Decision Processes for Spoken Dialogue Management. University of Cambridge; 2006. Available from: https://pdfs.semanticscholar.org/bb08/29c9a0c71c39d96afaecd070b1e1c047e53c.pdf.

[14] Bellman R. Dynamic Programming. 1st ed. Princeton, NJ, USA: Princeton University Press; 1957. Available from: http://books.google.com/books?id=fyVtp3EMxasC&pg=PR5&dq=dynamic+programming+richard+e+bellman&client=firefox-a#v=onepage&q=dynamic%20programming%20richard%20e%20bellman&f=false.

[15] Smallwood RD, Sondik EJ. The Optimal Control of Partially Observable Markov Processes Over a Finite Horizon. Operations Research. 1973;21(5):1071–1088. Available from: http://www.jstor.org/stable/168926.

[16] D'Andrea R. Solving the Bellman Equation, Dynamic Programming and Optimal Control, ETH Zrich. 2019. Available from: https://ethz.ch/content/dam/ethz/special-interest/mavt/dynamic-systems-n-control/idsc-dam/Lectures/Optimal-Control/Lecture%20Notes/Lecture5.pdf.

[17] Smallwood RD, Sondik EJ. The Optimal Control of Partially Observable Markov Processes over a Finite Horizon. Operations Research. 1973;21(5):1071–1088. Available from: https://EconPapers.repec.org/RePEc:inm:oropre:v:21:y:1973:i:5:p:1071-1088.

[18] Pineau J, Gordon G, Thrun S. Point-based value iteration: An anytime algorithm for POMDPs. In: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI); 2003. p. 1025 – 1032. Available from: http://www.cs.cmu.edu/~ggordon/jpineau-ggordon-thrun.ijcai03.pdf.

[19] Braziunas D. POMDP solution methods, Department of Computer Science, University of Toronto. 2003. Available from: https://www.techfak.uni-bielefeld.de/~skopp/Lehre/STdKI_SS10/POMDP_solution.pdf.

[20] Hauskrecht M. Value-Function Approximations for Partially Observable Markov Decision Processes. CoRR. 2011;abs/1106.0234. Available from: http://arxiv.org/abs/1106.0234.

[21] Feng Z, Zilberstein S. Efficient Maximization in Solving POMDPs.; 2005. p. 975–980. Available from: http://rbr.cs.umass.edu/shlomo/papers/FZaaai05.pdf.

[22] Kaelbling LP, Littman ML, Cassandra AR. Planning and acting in partially observable stochastic domains. Artificial Intelligence. 1998;101(1):99 – 134. Available from: http://www.sciencedirect.com/science/article/pii/S000437029800023X.

[23] Ross S, Pineau J, Paquet S, Chaib-draa B. Online Planning Algorithms for POMDPs. CoRR. 2014;abs/1401.3436. Available from: http://arxiv.org/abs/1401.3436.

[24] Koenig S. Agent-Centered Search. AI Magazine. 2001 Dec;22(4):109. Available from: https://www.aaai.org/ojs/index.php/aimagazine/article/view/1596.

[25] Howard RA. Dynamic Programming and Markov Processes. Cambridge, MA: MIT Press; 1960.

[26] Kristensen AR. Textbook notes of herd management: Dynamic programming and Markov decision processes; 1996.

[27] d'Epenoux F. A Probabilistic Production and Inventory Problem. Management Science. 1963;10(1):98–108. Available from: https://EconPapers.repec.org/RePEc:inm:ormnsc:v:10:y:1963:i:1:p:98-108.

[28] Lovejoy WS. Computationally Feasible Bounds for Partially Observed Markov Decision Processes. Operations Research. 1991;39:162–175.

[29] Poon KM. A fast heuristic algorithm for decision-theoretic planning; 2001. .

[30] Smith T, Simmons RG. Heuristic Search Value Iteration for POMDPs. CoRR. 2012;abs/1207.4166. Available from: http://arxiv.org/abs/1207.4166.

[31] Spaan MTJ, Vlassis NA. Perseus: Randomized Point-based Value Iteration for POMDPs. CoRR. 2011;abs/1109.2145. Available from: http://arxiv.org/abs/1109.2145.

[32] Hansen E, Zilberstein S. LAO: A heuristic search algorithm that finds solutions with loops. Artificial Intelligence. 2001 06;129:35–62.

[33] Paquet S, Chaib-draa B, Ross S. Hybrid POMDP algorithms. 2006 01.

[34] Paquet S, Tobin L, Chaib-draa B. An online POMDP algorithm for complex multiagent environments; 2005. p. 970–977.

[35] Geffner H, Bonet B. Solving Large POMDPs using Real Time Dynamic Programming; 1998. .

[36] Littman ML, Cassandra AR, Kaelbling LP. Learning policies for partially observable environments: Scaling up; 1995.

[37] Maniloff D, Gmytrasiewicz P. Hybrid Value Iteration for POMDPs.; 2011. .

[38] Weiss G. Multiagent Systems. The MIT Press; 2013.

[39] Boutilier C, Reiter R, Price B. Symbolic Dynamic Programming for First-Order MDPs.; 2001. p. 690–700.

[40] Wang C, Joshi S, Khardon R. First Order Decision Diagrams for Relational MDPs. J Artif Int Res. 2008 Mar;31(1):431472.

[41] Joshi S, Kersting K, Khardon R. Self-Taught Decision Theoretic Planning with First Order Decision Diagrams.; 2010. p. 89–96.

[42] Tuyls K, Weiss G. Multiagent Learning: Basics, Challenges, and Prospects. AI Magazine. 2012 Sep;33(3):41. Available from: https://www.aaai.org/ojs/index.php/aimagazine/article/view/2426.

[43] Hernandez-Leal P, Kartal B, Taylor ME. Is multiagent deep reinforcement learning the answer or the question? A brief survey. CoRR. 2018;abs/1810.05587. Available from: http://arxiv.org/abs/1810.05587.

[44] He H, Boyd-Graber JL, Kwok K, III HD. Opponent Modeling in Deep Reinforcement Learning. CoRR. 2016;abs/1609.05559. Available from: http://arxiv.org/abs/1609.05559.

[45] Neto G. From Single-Agent to Multi-Agent Reinforcement Learning: Foundational Concepts and Methods. 2005 01. Available from: http://users.isr.ist.utl.pt/~mtjspaan/readingGroup/learningNeto05.pdf.

[46] Rust J. Using Randomization to Break the Curse of Dimensionality. Econometrica. 1997;65(3):487–516. Available from: http://www.jstor.org/stable/2171751.

[47] Rust J. Using Randomization to Break the Curse of Dimensionality. Econometrica. 1997;65(3):487–516. Available from: http://www.jstor.org/stable/2171751.

[48] Oliehoek FA, Amato C. A Concise Introduction to Decentralized POMDPs. 2015. Available from: https://www.fransoliehoek.net/docs/OliehoekAmato16book.pdf.

[49] Wiering M, Otterlo M. Reinforcement Learning: State-Of-The-Art. vol. 12; 2012.

[50] Doshi P, Gmytrasiewicz P. Monte Carlo Sampling Methods for Approximating Interactive POMDPs. J Artif Intell Res (JAIR). 2009 01;34:297–337.

[51] Panella A, Gmytrasiewicz P. Interactive first-order probabilistic logic. AAAI Workshop - Technical Report. 2011 01:52–59.

[52] Han Y, Gmytrasiewicz P. IPOMDP-Net: A Deep Neural Network for Partially Observable Multi-Agent Planning Using Interactive POMDPs. Proceedings of the AAAI Conference on Artificial Intelligence. 2019 07;33:6062–6069.

[53] Kikuchi M, Yoshida M, Okabe M, Umemura K. Confidence Interval of Probability Estimator of Laplace Smoothing. CoRR. 2017;abs/1709.08314. Available from: http://arxiv.org/abs/1709.08314.

[54] Pawar V. Electric Vertical Take-off and Landing (eVTOL) Aircraft Market 2020. 2020. Available from: https://www.prnewsprime.com/2020/04/electric-vertical-take-off-and-landing-evtol-aircraft-market-2020/.

[55] Hornyak T. The flying taxi market may be ready for takeoff, changing the travel experience forever. 2020. Available from: https://www.cnbc.com/2020/03/06/the-flying-taxi-market-is-ready-to-change-worldwide-travel.html.

[56] Hawkins AJ. Liliums electric air taxi is finally actually flying in new video. 2019. Available from: https://www.theverge.com/2019/10/22/20925606/lilium-electric-air-taxi-flying-new-test-video.

[57] Kleiman-Weiner M, Ho MK, Austerweil JL, Michael L L, Tenenbaum JB. Coordinate to cooperate or compete: abstract goals and joint intentions in social interaction. In: Proceedings of the 38th Annual Conference of the Cognitive Science Society; 2016. .

[58] Cassano L, Sayed AH. ISL: Optimal Policy Learning With Optimal Exploration-Exploitation Trade-Off; 2019.

[59] Tokic M. Adaptive -Greedy Exploration in Reinforcement Learning Based on Value Differences; 2010. p. 203–210.

[60] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, et al. Playing Atari with Deep Reinforcement Learning. CoRR. 2013;abs/1312.5602. Available from: http://arxiv.org/abs/1312.5602.