Supervised learning for distribution of centralised multiagent patrolling strategies

Mehdi William Othmani-Guibourg

Supervised learning for distribution of centralised multiagent patrolling strategies

Mehdi William Othmani-Guibourg

Supervisors: Amal El Fallah-Seghrouchni
Jean-Loup Farges

Jury: François Charpillet (Referee)
Damien Pellier (Referee)
Philippe Bidaud (President)
Magali Barbier (Inspector)
Vincent Corruble (Inspector)
Vanda Luengo (Inspector)
Amal El Fallah-Seghrouchni
Jean-Loup Farges

This dissertation is submitted for the degree of
Doctorat de Sorbonne Université

Defended the 11th of December 2019

Acknowledgements

I should like to thank my supervisors, Amal El Fallah-Seghrouchni for her support and for having believed in me since my first internship with her in 2015, as well as Jean-Loup Farges for the quality of his guidance during these four years, and who has been more than a supervisor but undeniably a master who taught me a methodology and a mindset that I shall carry with me for the rest of my life.

I am grateful to my parents without whom I should not be here, particularly to my mother who has provided me through moral and emotional support throughout this dissertation journey, and who has taken care of me during difficult periods. I am also immeasurably grateful to Marion Carel for her extraordinary generosity, goodness, and encouragement, and who has been, from the outset, a light in a stormy night.

A very special gratitude goes out to my *grand* friend Gustav Öman Lundin, who has also been a counsellor and the English teacher that I have never had before, particularly for his listening, his attention, his thoroughness and the richness of our *Greek moments* and of our logorrhoeic discussions more generally.

Above all, however, I should like to thank Jean-Maximilien Cadic, who has put up with my incessant questioning and who helped me stay on course throughout this dissertation journey, despite the miles that separate us.

Remerciements

Je voudrais tout d'abord remercier mes directeurs de thèse, Amal El Fallah-Seghrouchni pour son soutien et pour avoir cru en moi depuis le premier stage que j'ai réalisé auprès d'elle en 2015, ainsi que Jean-Loup Farges pour la qualité de son orientation et de ses conseils pendant ces quatre années, et qui a été plus qu'un directeur de thèse mais indéniablement un maître qui m'a appris une méthodologie et un état d'esprit qui, j'en suis sûr, me servira pour le reste de ma vie.

Je remercie avec gratitude mes parents sans lesquels je ne serais pas là, tout particulièrement ma mère qui m'a apporté un soutien moral et émotionnel pendant toute la durée de la thèse, et qui a pris soin de moi pendant les moments difficiles. Je remercie également du fond du cœur Marion Carel pour son extraordinaire générosité, sa bonté, ses encouragements, et qui a été, depuis le début, une lumière dans une nuit de tempête.

Je tiens tout particulièrement à remercier mon *grand* ami Gustav Öman Lundin, qui a été un conseiller mais aussi le professeur d'anglais que je n'ai jamais eu avant, pour son écoute, son attention, sa méticulosité et la richesse de nos *moments grecs* et de nos discussions logorrhéiques de manière générale.

Par-dessus tout, cependant, je souhaiterais remercier Jean-Maximilien Cadic, qui a supporté mes incessants questionnements et qui m'a toujours aidé à garder le cap tout au long de cette thèse, malgré les kilomètres qui nous séparent.

Abstract

For nearly two decades, patrolling has received significant attention from the multiagent community. Multiagent patrolling (MAP) consists in modelling a *patrol task to optimise* as a multiagent system. The problem of optimising a patrol task is to distribute the most efficiently agents over the area to patrol in space and time, which constitutes a decision-making problem. A range of algorithms based on reactive, cognitive, reinforcement learning, centralised and decentralised strategies, amongst others, have been developed to make such a task ever more efficient. However, the existing patrolling-specific approaches based on supervised learning were still at preliminary stages, although a few works addressed this issue.

Central to supervised learning, which is a set of methods and tools that allow inferring new knowledge, is the idea of learning a *function mapping any input to an output* from a sample of data composed of input-output pairs; learning, in this case, enables the system to generalise to new data never observed before.

Until now, the best online MAP strategy, namely without precalculation, has turned out to be a centralised strategy with a coordinator. However, as for any centralised decision process in general, such a strategy is hardly scalable. The purpose of this work is then to develop and implement a new methodology aiming at turning any high-performance centralised strategy into a distributed strategy. Indeed, distributed strategies are by design resilient, more adaptive to changes in the environment, and scalable. In doing so, the centralised decision process, generally represented in MAP by a coordinator, is distributed into patrolling agents by means of supervised learning methods, so that each agent of the resultant distributed strategy tends to capture a part of the algorithm executed by the centralised decision process. The outcome is a new distributed decision-making algorithm based on machine learning. In this dissertation therefore, such a procedure of distribution of centralised strategy is established, then concretely implemented using some artificial neural networks architectures.

By doing so, after having exposed the context and motivations of this work, we pose the problematic that led our study. The main multiagent strategies devised until now as part of MAP are then described, particularly a high-performance coordinated strategy,

which is the centralised strategy studied in this work, as well as a simple decentralised strategy used as reference for decentralised strategies. Among others, some existing strategies based on supervised learning are also described. Thereafter, the model as well as certain of key concepts of MAP are defined. We also define the methodology laid down to address and study this problematic. This methodology comes in the form of a procedure that allows decentralising any centralised strategy by means of supervised learning. Then, the software ecosystem we developed for the needs of this work is also described, particularly *PyTrol* a discrete-time simulator dedicated to MAP developed with the aim of performing MAP simulation, to assess strategies and generate data, and *MAPTrainer*, a framework hinging on the *PyTorch* machine learning library, dedicated to research in machine learning in the context of MAP. Two MAP strategies relying on Long Short-Term Memory (LSTM) networks are then defined: *RLPM* and *RAMPAGER*. In those strategies, the LSTM network is used as a predictor that agents use to select the next place to visit in the area to patrol. It is trained over data generated by the centralised strategy. We also show that the stochastic selection of the next place to visit leads to better performance. RAMPAGER, which relies on analytical initialisation of the LSTM network guided by the structure of the area to patrol, turns out to be the best decentralised strategy based on LSTM networks. We then present a new generic type of strategy, called *Idleness Estimator*, relying on value estimation. In the strategies of this type, each agent embeds an estimator to estimate the time elapsed since the latest visit of any agent on each place to monitor. This estimator is trained over data generated by a high-performance centralised strategy as previously. Different strategies can then be derived according to the used estimator. In this dissertation we study three types of estimators: artificial neural networks, and particularly MultiLayer Perceptrons (MLPs), a linear model, and the mean. An interaction scheme is lastly set up to make agents communicate and improve their individual estimate through interaction.

Finally, the Idleness Estimator strategies, either with or without interaction, turn out to be the best decentralised strategies studied in this dissertation.

Table of contents

List of figures

List of tables

Nomenclature

Symbols

a	an agent
β	function mapping all the nodes of V to their one-hot representations
$c_{u,v}$	transit time between u and v
$d_a(t)$	decision of agent a at time t
$d(u,v)$	time-to-go from u to v
E	set of edges
G	graph to patrol
γ	function mapping one-hot vectors with their corresponding nodes in V, such as $\gamma = \beta^{-1}$
h_t^l	output of the layer l, at time t
$\hat{i}^a(t)$	enhanced vector of true idleness estimated by agent a at time t
$\hat{i}^{-a}(t)$	vector of true idleness estimated by agent a at time t
$\hat{i}_v^a(t)$	enhanced true idleness of the node v estimated by agent a at time t
$\hat{i}_v^{-a}(t)$	true idleness of the node v estimated by agent a at time t
$i^{+a}(t)$	vector of shared idleness of agent a
$i_v^{+a}(t)$	shared idleness of v for agent a
$i(t)$	vector of true idleness at time t
$i^a(t)$	vector of individual idleness of agent a at time t

$i_v(t)$ instantaneous node idleness of node v at time t

$i_v^a(t)$ individual idleness for agent a of the node v at time t

$i_{max,G}$, i_{max}, WI, $WI(G,T,J)$ worst idleness, worst interval throughout the patrolling mission

i_G, Iav_G, Iav graph idleness, average idleness

$i_G(t)$ instantaneous graph idleness of G at time t

l a layer of a neural network

$Linear$ linear model

$Mean$ mean model

$MI(G,T,J)$ mean interval

MLP_{ReLU}, $ReLUMLP$ $ReLU$ MLP model

N_a number of patrolling agents

Ng function setting to zero the coordinates not corresponding to the neighbours of a given node

Π a patrolling strategy

$C(K,\Pi)$ value of criterion C for the patrolling strategy Π executed in the configuration K

(L,H) recurrent neural network profile, with L being the number of layers and H the number of hidden units

$QMI(G,T,J)$, QMI quadratic mean interval

R_I^{trans} transitive interaction relation

R_I interaction relation

$s(t)$ global state at time t

$s_a(t)$ local state of agent a at time t

T duration of a patrolling mission

$\boldsymbol{\theta}$ set of parameters of a statistical model

V set of nodes

v a node

$v(n)$ the nth node visited by a given agent

$v^a(n)$ the node visited by agent a at the nth decision step

$v^a(t)$ the node visited by agent a at time t

X^N set of one-hot vectors corresponding to the set of nodes V with $card(V) = N$

Acronyms / Abbreviations

ACO Ant Colony Optimisation

ANN Artificial Neural Networks

BBLA Black-Box Learner Agent

BPTT BackPropagation Through Time

C-AEMS Coordinated Anytime Error Minimisation Search

CBLS Concurrent Bayesian Learner Strategy

CC Cognitive Coordinated

CDT Constrained Delaunay Triangulation

CE Cross-Entropy

CR Censcientious Reactive

CST Constrained Spanning Tour

DEC-POMDP Decentralized Partially Observable Markov Decision Process

EHP Evolutionary Heuristic to approximate the Min-Max Cost Closed Walk Problem

FSM Finite State Machine

GBLA Grey-Box Learner Agent

GBS Greedy Bayesian Strategy

GD Gradient Descent

GSMDP Generalised Semi-Markov Decision Process

HCC Heuristic Cognitive Coordinated

HPCC Heuristic Pathfinder Cognitive Coordinated

HPLE Heuristic Pathfinder Linear Estimator

HPME Heuristic Pathfinder Mean Estimator

HPRE Heuristic Pathfinder ReLU Estimator

IC Idleness Coordinator

IE Idleness Estimator

IHPLE Interacting Heuristic Pathfinder Linear Estimator

IHPME Interacting Heuristic Pathfinder Mean Estimator

IHPRE Interacting Heuristic Pathfinder ReLU Estimator

IIE Interacting Idleness Estimator

IRIE Interacting Random Idleness Estimator

IRHPLE Interacting Random Heuristic Pathfinder Linear Estimator

IRHPME Interacting Random Heuristic Pathfinder Mean Estimator

IRHPRE Interacting Random Heuristic Pathfinder ReLU Estimator

LIP Optimal Left-Induced Partition

LSTM Long Short-Term Memory

MAP MultiAgent Patrolling

MAS MultiAgent Systems

MDP Markov Decision Process

MI Mean Interval

MSE Mean Squared Error

MSP Multilevel Subgraph Patrolling

MST Minimum Spanning Tree

PART Self-organised partitioning algorithm for MAP

PCC Pathfinder Cognitive Coordinated

QMI Quadratic Mean Interval

RAMPAGER RAndom Muliagent PAtrollinG LSTM-Path- MakER

RF Reactive with Flags

RHPLE Random Heuristic Pathfinder Linear Estimator

RHPME Random Heuristic Pathfinder Mean Estimator

RHPRE Random Heuristic Pathfinder ReLU Estimator

RIE Random Idleness Estimator

RLPM Random-LSTM-Path-Maker

RMDP Reactive Markov Decision Process

RNN Recurrent Neural Networks

RR Random Reactive

SC Single Cycle

SEBS State Exchange Bayesian Strategy

SMDP Semi-Markov Decision Process

TBPTT Truncated Backpropagation Through Time

TSP Travelling Salesman Problem

WI Worst Idleness, Worst Interval throughout the patrolling mission

Chapter 1

Introduction

1.1 Context and motivations

Area monitoring is necessary for important practical applications: at one hand, in the field of civil security with the detection of fire outbreaks or intruders for example, and on the other hand, in the military field where the detection of hostile activities is an imperative task. Patrolling, also known as patrol task, is a generic activity consisting of humans, drones or robots monitoring an area. Such a task is well-suited to the purpose of collecting information, seeking objects, or watching over places to detect any intrusion for example. In the case of wide areas, one or several vehicles must move through the latter to monitor its different regions as often as possible. The choice of the temporal distribution of vehicles over the different regions of the area, as well as their route, determine the operational effectiveness of the monitoring. The control of those vehicles may be strongly constrained by communication issues. Indeed, for example in the context of a reconnaissance mission performed by a swarm of drones, or even for silent bots penetrating a network, communications may be impossible or, at the very least, discouraged.

Vehicles can be addressed as interacting and evolving entities, being even sometimes autonomous. A convenient and widely recognised way to represent and study such interactions between entities, as well as with their environment, is to regard them as agents evolving in a system; such systems are called *MultiAgent Systems* (MAS). They constitute a scientific paradigm based on computing systems which allow studying numerous complex fields such as artificial intelligence and life, software engineering, but also the natural and social sciences, as for example biology, economics, etc. In such a paradigm, the focus is put on interactions between parts, and not only on elements themselves. Interacting entities are defined as agents, and what is under study is the

result of their interactions with each other and with the environment. In that, multiagent systems are a relevant way, if not the most relevant, to represent and study complex systems.

Patrolling is a coordination problem with mobile agents that can be modelled as a multiagent system where the whole is greater than the sum of its parts. Such a multiagent system is referred to as the *multiagent patroling problem* or simply *multiagent patrolling* (MAP). For MAP and more generally for all multiagent coordination problems, a system is regarded as efficient if its components — namely the agents — are well coordinated via interaction, leading to a good distribution of agents over any kind of units. MAP problem consists in agents that must visit places of interest as often as possible. To that end, a good strategy of agent is informally that which leads to a good distribution of visits over the places throughout the patrolling mission.

MAP has been regarded as a good benchmark for MAS [34], because it makes the representation and the understanding of new methods easier, letting the researcher focus on the solution rather than on the representation of the problem [13]. Besides, by its use of generic performance measures independent from the studied strategy — based on the time lag between two visits to a place —, and the simplicity of its model, MAP seems to be a reasonable candidate as benchmark of multiagent coordination problem, to the extent that it is sufficiently generic, specific and representative. The quality of a coordination strategy is then evaluated in simulation by using different measures, each one measuring a specific property of the distribution of visits generated by strategies.

For over fifteen years, different types of strategies have been proposed: cognitive, reactive, coordinated, reinforcement-learning-based, auction-based, Ant-Colonisation-Optimisation-based, evolutionary-based, etc. Strategies wherefor the decision-making process is laid down before the mission are referred to as *precalculated strategies*, whereas those depending upon a central entity, generally a coordinator, are referred to as *centralised strategies*. So far, it has been showed experimentally that *precalculated* and *centralised* strategies are the best strategies according to the usual measures of performance used in the context of MAP [3]. However, precalculated strategies cannot adapt to any peculiar disturbance in the system, whereas coordinated strategies may necessitate an omniscient coordinator, fully and continuously available; such a strong assumption is difficult to hold. Indeed, in real scenarios communications can be interrupted, asynchronous and even jammed by enemies. In the worst case the coordinator may be eliminated. Furthermore, learning is known to allow for adaptation to unseen data. Thus, learned strategies are

expected to be more adaptive and result in more autonomous agents than precalculated or centralised strategies.

Machine learning is the scientific field which studies the means and methods by which computer systems can perform a class of tasks without having been explicitly programmed for that; machine must be here understood as a computer program. It relies on statistical models and algorithms using sample data, termed as *training data*, to perform inference and patterns detection, allowing then generalisation to new data never presented to the system before.

Although some works studied the use of machine learning within the framework of MAP, few addressed the advantages afforded by feed-forward artificial neural networks (ANN) to create new distributed strategies from centralised ones. The recent breakthrough in deep learning, a subfield of machine learning, have shone light on ANN with, among other, sequential data such as text, speech, audio and video with recurrent neural networks (RNN)[27]. The reader unfamiliar with ANNs and RNNs can find basic information in **Appendix A**.

The purpose of this work is to take a bet on distributed strategies, which are by design resilient and more adaptive to changes in the environment owing to their *acentricity*, in order to make the system more scalable and dynamic. To that end, the approach is to learn via machine learning, and more precisely via supervised learning methods, a distributed strategy from a centralised one, by distributing the coordinator's centralised decision process over patrolling agents. Such strategies split and crystallise, in some ways, this decision process into agents via supervised learning methods.

1.2 Overview

In order to thoroughly test and evaluate the earning of supervised machine learning methods, we first make extreme assumptions about the ability of agents: they can neither communicate nor perceive each other. These extreme assumptions enable studying and identifying the impact of these methods over the performance of the new MAP strategies devised in this dissertation. Moreover, in some scenarios these assumptions are justified. Generally, high-performance strategies make use of communications and centralised decision-making. The aim here is then to create a non-communicating strategy that approaches the performance of a high-performance centralised strategy, by *distributing* or *decentralising* the latter by means of machine learning.

Therefater, we introduce limited communications between agents as a support for interaction, in order to study and evaluate these new strategies in the context of interaction.

Therefore, **Chapter 2** describes the concepts underlying this work, such as MAP and preexisting strategies, including machine-learning-based strategies.

Chapter 3 presents in the first place a formalisation of the MAP problem as well as key ideas pertaining to it, but also the databases made up for this study, particularly for the needs of supervised learning. The methodology and the tools that allow the production of experimental results for the next parts of this work are also exposed.

Chapter 4 presents a new agent strategy based on a RNN architecture, the Long Short-Term Memory (LSTM) architecture. After training, an LSTM network is used by agents to navigate over the nodes to visit.

Chapter 5 extends **Chapter 4** and is devoted to an analytical method providing knowledge about the environment to the LSTM network used by agents to patrol.

Chapter 6 presents the application of statistical models and particularly several architectures of ANNs to the task of estimating general features of the environment from individual information collected by agents while patrolling. In that sense, machine learning models are used here to perform information reconstruction filling the gap of knowledge with regard to certain features of the environment, this gap of knowledge is owing to the absence of communication. Each statistical model gives rise to a new MAP strategy as part of the multiagent patrolling. Furthermore, in this chapter a methods that take into account large estimation noise of statistical models is described.

Chapter 7 describes an interaction scheme as part of the strategies introduced in **Chapter 6**, enabling limited interactions between agents.

Finally, **Chapter 8** concludes this dissertation and outlines perspectives for further works on machine learning for MAP.

Chapter 2

State of the art

Much is unknown about *distributed computational intelligence*, generally referred to as *distributed artificial intelligence*, and much remains to comprehend about mechanisms and phenomena at stake in the *emergence* of collective behaviours, self-organisation and synergies of autonomous agents. Considerable research is currently being devoted to the understanding of interaction in order to solve complex problems in a distributed manner, especially in MAS.

Central to computational intelligence is the construction of processes or system models [22] [23] which are not amenable to mathematical or traditional modelling because:

- the processes are too complex to represent mathematically;
- the process models are difficult and expensive to evaluate;
- there are uncertainties in process operation;
- the process is nonlinear, distributed, incomplete and stochastic in nature [54].

Such processes can be represented as complex systems. Complex systems are able to self-organise — over time — by modifying their own structure to release some tensions, or respond to a brutal change in their environment. The very example illustrating this tendency is the brain's ability to modify its neural connections during the learning process. They are also subject to phase transitions: a small variation of one control parameter can radically and irreversibly change the system's behaviour; such a kind of change is also referred to as a *bifurcation*.

In such systems, interactions are relevant and must be taken into account to the extent that they codetermine the future. Moreover, because of continuous interactions between their components, complex systems are in a perpetual state of becoming. Therefore, they

can only be described and analysed by modelling and simulating interactions between their components: complex systems lie on local interactions which occur there. To that end, interactions between components are studied by means of simulations in computer programs where components are regarded as agents: such a method is termed *agent-based simulation*, consisting, in fact, in MAS simulation.

To analyse such interactions, the theory of complex systems pays a particular attention to the theory of *self-organisation* within global systems arising from interactions between their elements. A convenient way to study interactions is to represent them as networks. The underlying structure of complex systems is thereby most of the time a network: social network in social sciences, semantic network in linguistics, boolean network in logic, or neural networks in computational intelligence.

Moreover, a system, and more specifically a controller, can be regarded as *robust* or *adptative* if it contains a sufficient *variety* — variety that can be thought of as a synonym of *complexity* [4]. Thereupon, to increase the complexity of a system, and thereby its robustness and adaptability, random variations shall be introduced or augmented. It would also be worth noting that another way to increase the complexity of the system would be to augment the interactions between its components.

Generally, in science the conventional flow is to frame the phenomenon or problem under study by devising a mathematical model describing it. Such a model is meant to capture the key features while disregarding irrelevant aspects of this problem; it is a very abstraction process. However, this engineering design flow may be costly and necessitate domain experts. Alternatively, machine learning does not define beforehand an explicit model based on the domain knowledge, but rather, *discovers* dynamically a model from data specific to the problem at hand. In doing so, the machine learning approach relies on general-purpose algorithms — the *optimiser* — evolving a metamodel — the *machine learning model* — with the aim of optimising an objective — the *objective function*. The data used by the optimiser already exist or can be generated extempore.

The purpose of this chapter is to propose, a state of the art of MAP, as well as different approaches used to solve the MAP problem, more precisely a methodology and different algorithms to do so. It also shows to what extent this problem is a temporal decision-making problem. Finally, certain of existing supervised learning approaches applied to MAP are presented.

2.1 Multiagent patrolling

Patrol task — sometimes referred to as *patrolling* or *area coverage* in the field of robotics — is an activity by nature distributed in space and time. Several agents patrolling an area actually perform a *distributed surveillance*, namely the surveillance of an area using a group of agents coordinated in a distributed manner [45]. A convenient way to model such a task is to represent it as a MAS, resulting in MAP. Then, the problem to find the best coordination strategy to patrol that area is called the *MAP problem*, or by metonymy simply MAP.

There are many possible ways to define the MAP problem, and many of these definitions are equivalent up to small transformations. A consensual definition is to define it as a generic problem of multiagent coordination providing a formalism to study, for a group of agents, the task of visiting a set of places as soon as possible. Poulet pointed out it can also be regarded as an allocation problem, by considering that visiting a place is an indivisible and recurrent task whose the execution time has a constant part — the visit itself — and a variable part — the time for an agent to reach the place to visit —, variable part hinging upon its distance to the place [42]. As such, the MAP problem is a coordination problem whose the object of coordination consists of the places of interest to visit. As an allocation problem, it is more generally a *temporal decision-making problem* that can be reformulated as a complex system problem where agents must be processed as often as possible by the places to visit [42].

Also, it is worth noting that the system composed of communicating agents constitutes a mobile sensor network. These agents can share sensory information through this network.

The MAP problem has two variants:

- patrolling with the aim of defending an area against an adversarial intrusion, called the *adversarial MAP problem*,
- patrolling with the aim of monitoring and checking a set of points of interest, called the *temporal MAP problem* [42] or *multiagent timed patrolling*[1] [52].

In the first case, the area to patrol represents the border of the region to supervise, border wherein intruders can penetrate. The objective is then to optimise the probability of intercepting any intruder given certain assumptions about their behaviour. This type of MAP is akin to *multi-agent fence-patrolling*. Some works concentrated upon

[1]The term “timed” seems to be irrelevant as far as it can be considered as a synonym of “scheduled”.

adversarial MAP: multi-agent fence-patrolling strategies [15], algorithms based on a two-player leader-follower game model where the patroller is the leader and the intruder is the follower [5], probabilistic strategies coping with three types of intrusion: a random intruder, an intruder that waits until the guard leaves the site to initiate the attack, and an intruder that uses statistics to forecast how long the next visit to the site will be [50]. Cooperative strategies based on a DEC-POMDP model facing several adversaries with limited observability and rationality were also studied [7].

In the second case, each point of interest is regarded as an endangered place where an intruder potentially stands, and the objective is then to check all places as often as possible to neutralise any potential intrusion: every location has the same importance. This work was carried out within this framework: MAP studied here is neither adversarial, nor fence-patrolling. It is temporal and generic, and the algorithms are investigated and developed for drones in a military context, making inapplicable all type of algorithms based on communication through environment.

2.1.1 Environment

Area patrolling missions can be studied as an area coverage problem. In the robotics' perspective, Choset proposed to divide coverage approaches into offline methods, wherefor the layout of the environment is known, and online methods, for which the map of the environment is unknown [9]. In what follows, we shall only focus upon the first approach, that is offline methods wherefor the layout of the environment is known beforehand.

Choset pointed out that an area coverage can be achieved using a *cellular decomposition* of the free space [9]. A cellular decomposition breaks down the target region into cells such that coverage in each cell becomes "simple". A complete coverage can then be attained by ensuring that the agents visit each cell in the decomposition. There exists three types of decomposition: *approximate*, *semi-approximate*, and *exact cellular* decomposition. Methods based on approximate cellular decomposition, such as grid-based methods, have limitations since they do not consider the structure of the environment [16]. Therefore, they are unable to handle partially obstructed cells or cover areas close to the boundaries in continuous spaces. In contrast, methods based on exact cellular decomposition, e.g. graph-based methods, which employ structures such as the *Reeb graph* to represent the environment do not suffer from those restrictions [44].

Similarly, Russel, Norvig and Stout, in a more generic approach speak rather of *skeletonisation* that consists in replacing the real ground by a graph, which can be interpreted as a skeleton, representing possible paths between the places to visit covering all the area [49] [55]. *Voronoi diagrams* [56], *Constrained Delaunay Triangulation* (CDT)

[16], *Visibility Graphs* and *C-cells* can be used to generate such a graph. For example Tan et al. proposed a distributed model for cooperative multirobot systems based on Voronoi diagram to model the area coverage [56]. Visiting all places ensures then that the entire area be covered. In the case where agents communicate and make up a mobile sensor network, visiting all places while maintaining the network will lead to maximise the sensor network coverage area of the environment. This is generally the main objective of robot cooperation in the context of area coverage [56]. However, although guaranteeing to cover the whole environment in continuous spaces while traversing the graph, this might also include many redundant movements.

Another example of graph generation is given by Fazli et al. Initially, with their approach a set of N_a agents standing for guards are positionned to cover visually the entire area to supervise. Then, a representation of the environment in the form of a spanning tree over the vertices of the graph, which represent in fact the guards, is computed using the CDT to yield a graph over the guards.

A significant advantage of such an abstract representation is that a wide range of problems, from ground patrolling to web navigation, can be modelled as a MAP. The graph, as an abstract representation of the area to patrol, is termed as *topology* in our context, and depicts the layout of the environment. In the rest of this dissertation, if that does not lead to confusion, the topology may also be termed as *network*.

Finally, the environment of an agent consists of the topology and the *society* of other agents patrolling over there.

2.1.1.1 Topology

Definition 2.1.1. Topology. Chevaleyre depicts a topology as a graph $G = (V, E)$, where $V = \{1, \ldots, N\}$ is the set of nodes and $E \subseteq V^2$ the set of edges of G [8]. To each edge $(u, v) \in E$ is associated a weight $c_{u,v}$ representing the distance between nodes and assuming that any agent travels one unit of distance in one unit of time, $c_{u,v}$ stands for the time to travel this edge. Any edge weight c_{uv} represents also the time taken by an agent to travel an edge (u, v) and will therefore be referred to as the *transit time* of (u, v) in the remainder of this dissertation.

In its model, Chevaleyre assumes the graph is *metric*, i.e. the triangular inequality is not violated, that is, $\forall u, v, w \in V$ three nodes connected by edges, $c_{u,v} + c_{vw} \geq c_{u,w}$. Accordingly, the weight of a path $v_0, ..., v_n$ equal to $\sum_{j=0}^{n-1} c_{v_j, v_{j+1}}$ is noted $c(v_0, ..., v_n)$, whereas

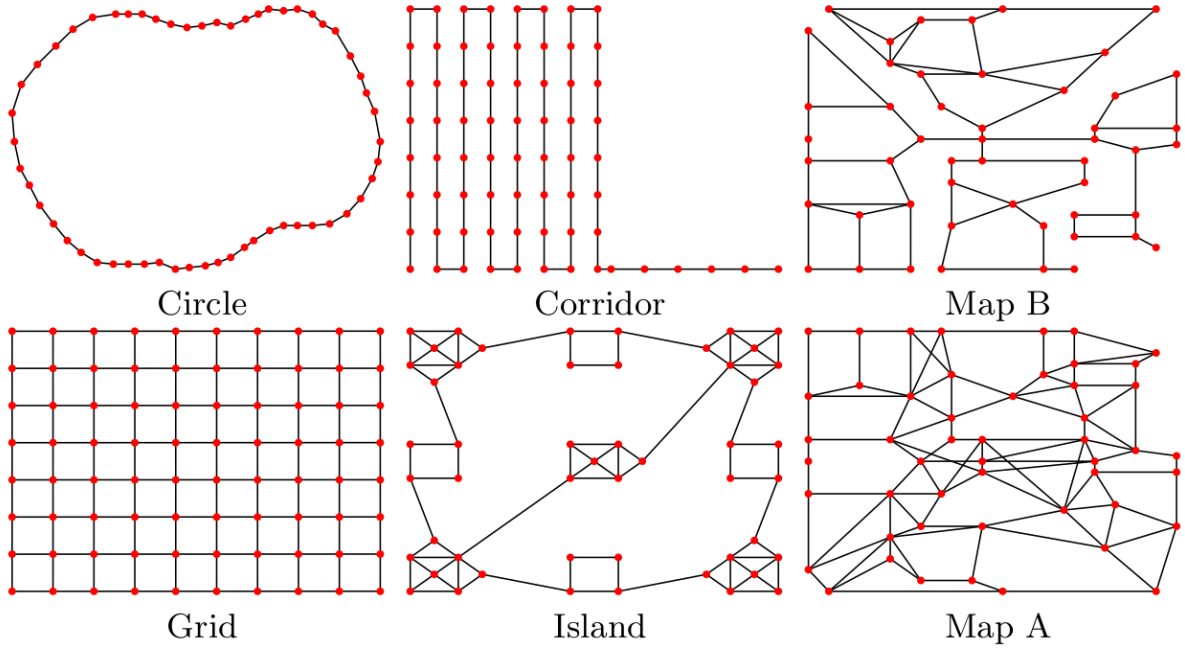

Fig. 2.1 Standard benchmark of MAP topologies [26]

that of a path of edges E', such as $E' \subseteq E$, is noted $c(E')$.

Note that for some works the transit time is a random variable with a distribution over potential travel durations [7].

Generally, in the MAP community six topologies are commonly used as benchmark to evaluate strategies. **Fig. 2.1** shows graphs similar to these topologies. These graphs are characterised as follows:

- *Islands*: a 50-node topology with islands,
- *A*: a 50-node regular topology,
- *Grid*: a 50-node topology in the shape of a grid,
- *B*: a 50-node regular topology with bottlenecks,
- *Circle*: a 50-node topology in the form of a circle,
- *Corridors*: a 49-node topology in the shape of corridors.

In order to align with previous works of the MAP community, the proposed strategies of this work are assessed using these topologies.

2.1.2 Society of agents

In early works, the society of agents was only characterised by the number of agents belonging to it [28][29]. Later the society was defined as a set of agents that are also able to leave it [42]. In this dissertation only closed societies of agents, noted A, are considered, that is societies with a constant number of agents, $N_a = card(A)$, throughout the mission.

Agents are *homogenous*, namely all agents are identical, i.e. they stand for the same type of unmanned aerial drones, and move at the same speed. Despite the fact that caracteristics of mobile robots involved in a mission may vary from one robot to another, the homogeneity hypothesis is usual in the MAP literature. Agents are *anonymous*, i.e. for every agent all the other agents are identical and it cannot distinguish them; for the designer, agents are identified by their ID, although their strategy cannot be an explicit function of the this ID. This hypothesis is implicitly assumed in some works [53], but rejected in other, where the nodes of the graph are dispatched among the agents [7].

2.1.3 Performance measures

An *evaluation criterion*, sometimes known as *quality criterion* [24], or *metric*, or *performance measure*, is a measurement which characterises a particular aspect of a MAP execution for a graph G and a duration of simulation $T \in \mathbb{N}^*$. More generally, an evaluation criterion is used to assist in making an objective and fair procurement source decision [33]. In what follows, we shall describe the more relevant ones for MAP.

2.1.3.1 Idleness

In MAP, as all places must be visited as often as possible, the time elapsed since the last visit is a natural candidate as prime and basic criterion. By doing so, Machado et al. introduced the *idleness* of a node at time $t \in \mathbb{R}^+, t \leq T$, where $T \in \mathbb{R}^+$ is the duration of the simulation, as the time elapsed since the last visit [29]. This has then led to the following criteria:

- the *instantaneous node idleness* at time $t \in \mathbb{R}$, noted $i_v(t), \forall v \in V$, corresponds to the time elapsed since its last visit by any agent at time t; if this does not lead to confusion, it will simply be termed as *idleness*,
- the *instantaneous graph idleness* at time $t \in \mathbb{R}$, noted $i_G(t)$, corresponds to the average instantaneous idleness of all nodes a time t,

- the *graph idleness* or *average idleness*, noted i_G, Iav_G, or simply Iav, being the average instantaneous graph idleness over all the execution's duration,
- the *worst idleness*, noted $i_{max,G}$, i_{max} or WI, is the biggest value of instantaneous node idleness occurred during the entire simulation [29]:

$$i_{max,G} = \max_{t \in \mathbb{R}^+ : t \leq T, v \in V} (i_v(t)) \tag{2.1.1}$$

As stated by Acevedo et al., it is worth noting that using the WI ensures that the probability an event not be detected by an agent is upper bounded to a known value [1].

2.1.3.2 Exploration time and decision cost

Machado et al. also introduced two other idleness-independent criteria:

- the *exploration time*, which consists of the time necessary to the agents to visit at least once all the nodes of the graph; such a criterion can be intuitively thought of as exploring an area in order to map it [29],
- the *decision cost*, which is given by the sum of all the durations required by every agent to choose the node to visit; assuming that agents stop at nodes to make the decision about the next node, such a criterion enables showing that the longer the decision takes time, the longer the nodes remain without being visited [28].

2.1.3.3 Qualitative measures

In addition to the MAP criteria defined above, Menezes et al. mentioned possible qualitative measures:

- *scalability*, that evaluates whether the strategy is capable of patrolling over topologies of any dimension,
- *stability*, that measures the variation on the idleness of each node, indicating whether nodes are uniformly visited,
- *adaptability*, that evaluates the ability of the strategy to deal with topological changes during the simulation,
- *reactiveness*, that evaluates the ability of the strategy to patrol over various topologies without learning time, recalibration of parameters and definition of specific strategies to perform the task for each topology [32].

However, it is regrettable that, thus far, no criterion be associated to these qualitative measures, except the stability that corresponds to the standard deviation of the instantaneous idleness of nodes [42].

2.1.3.4 Interval criteria

Thereafter, Sampaio et al. introduced the notion of *interval between visits* of a node as a new type of evaluation criterion [52]. They presented it as more germane and intuitive than the average idleness, insofar as the latter considers successive values of the instantaneous idleness of a node. An interval of a node is considered as the time elapsed between two visits of any agent, it is a real positive number. Let:

- $\forall v \in V$, J_v be the set of intervals of v during a MAP execution upon G, such that $N_J^v = |J_v|$,
- $J = \{J_1, \ \ldots \ , J_N\}$ be the set of intervals for every node resulting from a MAP execution upon G,
- $N_J = \sum_{v \in V} N_J^v$ be the total number of intervals resulting from a MAP execution upon G.

Then, this new notion gives rise to four evaluation criteria:

- the *mean interval*, noted $MI(G, T, J)$, which captures the average performances of interval between visits, such as:

$$MI(G, T, J) = \frac{\sum_{v \in V} \sum_{j_v \in J_v} |j_v|}{N_J} = \frac{N \times T}{N_J} \tag{2.1.2}$$

- the *quadratic mean interval*, also called *mean square interval* [52], noted $QMI(G, T, J)$, which captures an average value of interval which takes into account, by its *quadraticity*, the distribution of visits over the nodes of G. The lower $QMI(G, T, J)$ is, the better visits are distributed over the graph. As previously,

$$QMI(G, T, J) = \sqrt{\frac{\sum_{v \in V} \sum_{j_v \in J_v} |j_v|^2}{N_J}} \tag{2.1.3}$$

  To optimise — i.e. to minimise — QMI will lead to establish a more equitable solution — i.e. strategy — than that obtained by minimising the MI.

- the *variance of intervals*, noted $VI(G,T,J)$:

$$VI(G,T,J) = QMI(G,T,J)^2 - MI(G,T,J)^2 \tag{2.1.4}$$

- the *worst interval*, noted WI, which is the biggest interval recorded during the overall simulation. It follows:

$$WI(G,T,J) = \max_{v \in V,\ j_v \in J_v}(j_v) \tag{2.1.5}$$

  Minimising $WI(G,T,J)$ is equivalent to search for a strategy as robust as possible. There is identity between the worst idleness $i_{max}(G,T,J)$ and the worst interval $WI(G,T,J)$, by definition.

Interestingly, MI, QMI and WI are particular cases of the *generalised mean* of intervals. Let $j_1, \cdots, j_n$ be all the intervals of the graph G. The generalised mean, or *power mean*, of these intervals with exponent $p > 0$ is:

$$MI_p(G,T,J) = \left(\frac{1}{N_J} \sum_{v \in V} \sum_{j_v \in J_v} j_v^p \right)^{1/p} \tag{2.1.6}$$

It follows that,

$$MI(G,T,J) = MI_1(G,T,J) \tag{2.1.7}$$

$$QMI(G,T,J) = MI_2(G,T,J) \tag{2.1.8}$$

$$\begin{aligned} WI(G,T,J) &= MI_{+\infty}(G,T,J) \\ &= \lim_{p \to \infty} M_p(G,T,J) \\ &= \max_{j in J}(j) \end{aligned} \tag{2.1.9}$$

From the foregoing, it can be stated that minimising $MI_p(G,T,J)$ as p increases, is equivalent to find a more equitable solution, and by doing so, a more robust strategy. $QMI(G,T,J)$ can then be regarded as an intermediate criterion, among others, between $MI(G,T,J)$ and $WI(G,T,J)$.

Also, it is worth noting that the average idleness of G, $Iav(G,T,J)$, can be reformulated as a function of $MI(G,T,J)$ and $QMI(G,T,J)$ as follows:

$$Iav(G,T,J) = \frac{N_J}{2N \times T}(QMI^2(G,T,J) + MI(G,T,J)) \tag{2.1.10}$$

From the foregoing, the Iav can then be written as an expression of the MI and QMI, with a particular emphasis on QMI.

2.1.3.5 Normalisation

Any criterion C selected to evaluate a MAP execution can be normalised to compare different topologies and numbers of agents [29]. With notations of **Definition 2.1.1** and **Subsection 2.1.2**:

$$C_{norm}(G,T,J) = C(G,T,J) \times \frac{N_a}{N} \tag{2.1.11}$$

2.1.4 Discrete time model

A common specialisation of the MAP model is that for which a sample time is defined, where all transit times $c_{uv}, \forall u, v \in V$ are expressed as an integer number of sample times; such a sampling leads to a discrete-time MAP model. In this case, the formal model can be integrated step by step: agent positions and node idlenesses at time $t \in \mathbb{N}^*$, can be computed from agent positions, node idlenesses and agent decisions with the relevant heading at time $t-1$. Such a time $t \in \mathbb{N}^*$, is referred to as *period* and can be defined as follows:

Definition 2.1.2. Period. In the MAP model, when a sample time is defined, the sampling period is referred to as *period*, or *time step*, and makes up the unit of time.

Furthermore, in the specific case studied by Machado et al. [29], where $\forall u, v \in V$, $c_{u,v} = 1$, at each time step any agent is then necessarily on a node.

In our framework, we pose the hypothesis that visits are instantaneous, i.e. the action of visiting a node lasts 0 period.

2.2 Methodology

In their seminal work of MAP, Machado et al. established a general methodology to study MAP consisting of the following steps [29]:

1. specification of performance measures among those of **Section 2.1.3**,
2. definition of strategies to evaluate,

3. definition of some case studies with specific environment and societies of agents, called *patrolling scenario* or *patrolling configuration*

4. design and implementation of a simulator to perform experiments

2.3 Classical strategies

At time 0, N_a agents are located upon nodes of G. When the patrol task starts, agents move simultaneously around the nodes and edges of the graph according to a predetermined *multiagent strategy*, also termed as *multiagent architecture* [29]. When that does not lead to confusion, in the following it will simply be referred to as *strategy*, and in the context of MAP a strategy is defined as follows:

Definition 2.3.1. Strategy. A *strategy* executed by an agent is defined as a procedure $\pi : \mathbb{N} \to V$ such that $\forall n \in \mathbb{N}, \pi(n)$ is the n^{th} node visited by the agent. A *multiagent strategy* $\Pi = \{ \pi_1, ..., \pi_{N_a} \}$ is then simply defined as a set of N_a single-agent strategies [8]. Otherwise, as part of MAP, an agent strategy can be thought of as a *decision-making algorithm* generating a list of paths, one path per agent, giving rise to a *patrolling scheme*.

Note that this definition of a strategy corresponds either to a plan to execute in an open loop or to the outcome of the execution of a given procedure to choose the next node. Alternative definitions corresponding to policies could define it as a function mapping the current knowledge about the state to the next node to visit.

According to this definition, an agent a will visit the node corresponding to $\pi_a(n)$ after a time equal to the weight of the path $\pi_a(0), ..., \pi_a(n)$., which is $\sum_{k=0}^{n-1} c_{\pi_a(k)\pi_a(k+1)}$. When the strategy is deterministic, the path and its corresponding travel time are known beforehand.

Loosely speaking and informally, in the framework of MAP a good strategy is that which minimises the time lag between two passages at the same place, i.e. the interval of time, and this for all places. Besides, optimality depends always upon the considered evaluation criterion. It is thereby necessary to consider optimality according to a specific criterion among those defined in **Subsection 2.1.3**.

Machado et al. proposed four basic parameters characterising a strategy [29] that enables exploring strategies methodologically, that is to say using a bottom-up and incremental approach for designing MAS strategies. These parameter considered as essential for understanding and measuring the impact of a strategy on the dynamics of a collective problem-solving process[12]. These basic parameters are:

1. the *strategy type*: *reactive*, *cognitive*, etc.,
2. the *communication type*: none, messages, blackboard, through the environment, etc.,
3. the *selection criterion of the next node*: locally random, time without visits in the neighbour, time without visits in the whole graph, etc.,
4. the *coordination strategy*: *emergent* or *central*.

Usually, and particularly in MAS, agents are considered as reactive when they act with respect to their perceptions, and cognitive when they may pursue a goal [29].

For each one of these basic parameters it exists an infinity of possibilities. These values are only characterisations, or archetypes.

Finally, there exists a wide variety of strategies, and in what follows, only the most significant will be retained.

For example, some strategies based on negotiation mechanisms were investigated [3], [32]. In the context of MAS, negotiations can only take place when agents are able to communicate. Inside the framework established in this disseration, a realistic assumption consists in considering that agents are able to communicate sporadically according to their respective trajectory. By doing so, neither any strategy requiring static communication links will be described and, a fortiori, nor any negotiation-based strategy.

Other examples are strategies based on *Ant colony optimisation* (ACO) algorithms. They consist of agents acting as ants via pheromone-based communication. In such strategies agents are spread out over the graph, then they lay pheromones down upon nodes to determine the patrolling strategy through an ACO procedure [24] [25] [10] [17]. Although performing well in simulation, this type of algorithms assumes interactions with environment hardly practicable in real life. Therefore, given that within our framework agents can be drones, these algorithms are not described here.

2.3.1 Reactive strategies

In the context of MAP, Machado et al. defined reactive agents as those whose the field of vision is one-edge depth, i.e. reactive agents only perceive adjacent nodes referred to as their neighbourhood [29]. Therefore, they cannot plan to visit a remote node, and thereby pursue a goal. Reactive agents are also sometimes said to act based upon their perception.

Incidentally, it is worth noting that the concept of agents acting upon their perception does not pertain to the application motivating this work. The graph, indeed, is an

abstraction of the area to patrol and the perception of an agent using sensors provides only information about the detection or not of events such as fire outbreak or hostile activity at the place where the agent stands. Nonetheless, decentralised strategies can be reformulated as strategies planning only the next node to visit, that is either randomly or considering the path already travelled, respectively.

Machado et al. designed three reactive strategies [29]. First, *Random Reactive* (RR) agents do not communicate and choose randomly their next node to visit. Then, *Conscientious Reactive* (CR) agents select the next node to visit as that having the highest *indidividual idleness* in their neighbourhood. There is no communication between agents: idlenesses are estimated by each agent on the basis of their own path. CR can be thought of as a good representative and thereby a comparison strategy for the reactive ones. CR is designed following a bottom-up approach, that is, agents before organisation.

On a side note, as pointed out by Almeida et al., selecting the next node to visit according to the individual idleness can be thought of as following a gradient: in MAP, nodes with a high idleness act as valleys, or attractors, whereas those with a low idleness act as mountains, or repulsors [2].

Finally, the *Reactive with Flags* (RF) algorithm is similar to CR except that agents leave a flag on the visited nodes leading to share their idleness estimate using environment as a communication medium.

Note that with respect to **Def. 2.3.1**, reactive strategies do not correspond to plans to be executed but to results of execution of their reactive procedure.

2.3.2 Cognitive strategies

In the context of MAP, cognitive agents can perceive the whole graph [29]. Cognitive agents are thus able to pursue a goal, namely to plan to visit a distant node. As previously indicated, for the application motivating this work, the focus is on the capacity to plan a visit to a remote node, rather than on perception skills.

Machado et al. proposed the *Idleness Coordinator* (IC) strategy [29], also called *Cognitive Coordinated* (CC) by Almeida et al. [2] and in further works. In this strategy, agents can perceive the whole graph and select the next remote node to visit as that with the highest *true idleness*. The agent is instructed to visit this node by a centralising coordinator. Everything happens as if there was a perfect communication between agents. Actually, agents communicate unrestrictedly with the coordinator and idlenesses are estimated by the latter on the basis of all information it has centralised. It is responsible

for avoiding that more than one agent choose the same next remote node and responds instantaneously to provide it. When the coordinator is fully-informed, i.e. it has a global knowledge of executed and planned paths for all agents, it can be thought of as an *omniscient coordinator*. Eventually, agents use the *Floyd-Warshall* algorithm to compute the shortest path between their current node v_0 and the assigned goal v_1. This algorithm minimises the *time-to-go* from v_0 to v_1, i.e. $d(v_0, v_1)$.

Almeida et al. introduced two methods to improve the CC strategy, the first termed *Heuristic* is used to select the next remote node to visit exploiting more information than just true idleness, whereas the second one, termed *Pathfinder*, is used to compute the path to go there exploiting more information than just distance [2].

2.3.2.1 Heuristic method

The *Heuristic* method enables selecting the next goal node by taking into account not only the normalised idleness, but also the normalised *time-to-go* of a candidate goal node from the agent's current position. The time-to-go between two nodes of V is the travel time of the shortest path between these two nodes. Idleness and time to go are normalised by scaling them between 0 and 1. Because edges with high idleness values ought to be traversed first, 0 is assigned to the maximum idleness whereas a value equal to 1 is assigned to the minimum idleness. Intermediary values are calculated by means of proportions as shown in **Eq. 2.3.1**:

$$\forall v_0 \in V, \text{ if } \min_{v \in V}(i_v(t)) \neq \max_{v \in V}(i_v(t)), \forall v_0 \in V$$
$$\bar{i}_{v_0}(t) = \frac{\max_{v \in V}(i_v(t)) - i_{v_0}(t)}{\max_{v \in V}(i_v(t)) - \min_{v \in V}(i_v(t))} \tag{2.3.1}$$

where $i_v(t)$ and $\bar{i}_v(t)$ are the true and normalised idlenesses of v at time t, respectively.

Normalised time-to-go is calculated similarly. Because edges with short distances must be traversed first, 0 is assigned to the minimum time-to-go whereas 1 is assigned to the maximum time-to-go. Intermediary values are calculated by means of proportions as shown in **Eq. 2.3.2** :

$$\begin{aligned}
&\forall d(v_0, v_1) \text{ the time-to-go from } v_0 \text{ to } v_1, \\
&\quad \text{if } \min_{v,w \in V : v \neq w} (d(v,w)) \neq \max_{v,w \in V : v \neq w} (d(v,w)) \\
&\qquad \bar{d}(v_0, v_1) = \frac{d(v_0, v) - \min\limits_{v,w \in V : v \neq w} (d(v,w))}{\max\limits_{v,w \in V : v \neq w} (d(v,w)) - \min\limits_{v,w \in V : v \neq w} (d(v,w))}
\end{aligned} \tag{2.3.2}$$

Finally, for an agent at the position v_0 at time t, the values associated to nodes $v \in V$ are given by **Eq. 2.3.3**:

$$\begin{aligned}
&\forall v \in V, val_{r_H, v_0}(v, t) = \\
&\qquad r_H \times \bar{i}_t(v)) + (1 - r_H) \times \bar{d}(v_0, v))
\end{aligned} \tag{2.3.3}$$

where the weighting factor $r_H \in [0, 1]$ must be chosen by the designer. Minimising the node values according to that expression, i.e. selecting the nodes with the minimum value, allows for agents to visit nearby nodes with higher idleness first and foremost. Moreover, as for CC, there is a mechanism forbidding the coordinator to select nodes that are already assigned to other agents, for the purpose of maintaining the consistency of the node assignment.

2.3.2.2 Pathfinder method

The Pathfinder method computes the path leading to the goal node. This method takes into account the idleness of the nodes for all paths between the current location and the goal, in order to compute the best path leading there. In doing so, the normalised transit time of an edge $\bar{c}_{v_1,w_1}$ is defined from transit times of G as follows:

$$\begin{aligned}
&\forall e_1 = \{v_1, w_1\} \in E, \text{ if } \min_{e=\{v,w\} \in E} (c_{v,w}) \neq \max_{e=\{v,w\} \in E} (c_{v,w}), \\
&\qquad \bar{c}_{v_1,w_1} = \frac{c_{v_1,w_1} - \min\limits_{e=\{v,w\} \in E} (c_{v,w})}{\max\limits_{e=\{v,w\} \in E} (c_{v,w}) - \min\limits_{e=\{v,w\} \in E} (c_{v,w})}
\end{aligned} \tag{2.3.4}$$

Note that for topologies whose all the edges have the same transit time, the same arbitrary value between 0 and 1 is assigned to each edge in both directions, and the resultant graph is thereby directed. Arcs are then valued as follows:

$$\begin{aligned}
&\forall e = (v,w) \in E, \\
&\qquad c_{r_P}(e) = r_P \times \bar{i}_w(t) + (1 - r_P) \times \bar{c}_{v,w}
\end{aligned} \tag{2.3.5}$$

where the weighting factor $r_P \in [0, 1]$ must be chosen by the designer. As raised above, the resultant graph computed by the Pathfinder method is directed, resulting in turning edges $e = \{v, w\} \in E$ into $e = (v, w) \in E$.

Minimising the weights of edge according to this expression allows also agents to visit nearby nodes with higher idleness first and foremost.

2.3.2.3 Heuristic and Pathfinder methods

Applying this two methods with CC gives rise to the *Heuristic Pathfinder Cognitive Coordinated* (HPCC) strategy. The decision process of HPCC includes then two steps:

- Heuristic method, i.e. the selection of a target node that is not necessary in the neighbourhood,
- Pathfinder method, the computation of a path between the current position of the agent and the target node previously selected.

In the HPCC strategy, the coordinator makes a decision by applying these two methods that take as inputs the vector of true idleness and the vector of agent positions as shown in **Figure 2.2**.

Also, it is worth noting that both may be used separately, giving rise thereupon to either *Heuristic Cognitive Coordinated* (HCC) or *Pathfinder Cognitive Coordinated* (PCC) strategies.

2.3.2.4 Performance

Fig. 2.3, from Almeida et al., depicts for 5 and 15 agents the performance, according to the Iav averaged over the six topologies, of different MAP strategies of the state of the art, namely CR, CC, HPCC, SC, GBLA, a strategy based on reinforcement learning presented in **2.3.4.1**, and HPTB and HPMB [3], negotiation-based strategies that are not studied in this dissertation. These results highlight the following order on the performance of these strategies:

$$SC > HPCC > GBLA > HPTB > HPMB > CC > CR \tag{2.3.6}$$

HPCC, as communicating, fully-informed, coordinated, and thereby centralised strategy, is then a high-performance and the best online — namely without precalculation of paths — strategy until now, according to the results laid out by Almeida et al. [3]. It can then be regarded as a representative and thereupon a comparison strategy for

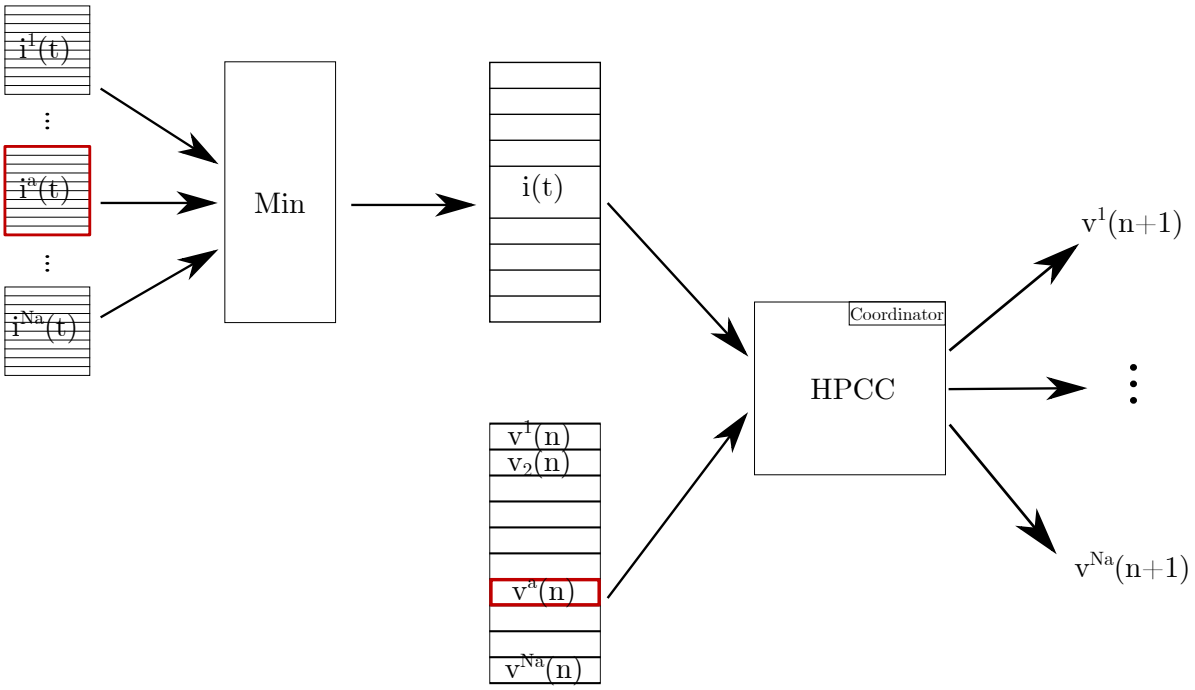

Fig. 2.2 Decision procedure of HPCC, with $n \in \mathbb{N}$ standing for the nth decision step, t the corresponding time, and $v^a(n)$ and $i^a(t)$ the node visited by agent a as well as its vector of individual ideleness, resp., at the nth decision step.

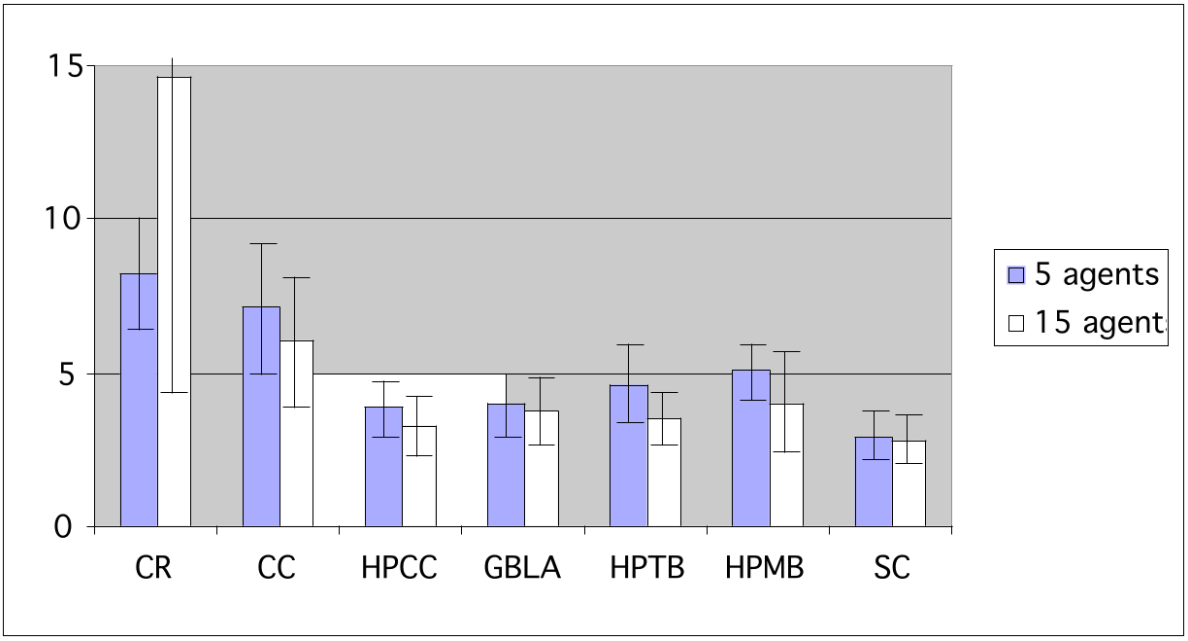

Fig. 2.3 Normalised averaged idleness of several MAP strategies averaged over the six topologies, for 5 and 15 agents [3].

coordinated and centralised strategies. In fact, HPCC follows a top-down designing, that is to say the society is considered prior to agents and influences them.

2.3.3 Graph-theory-based strategies

Numerous works have been led in the graph-theory framework, which have resulted in worthwhile theoretical results. In the following, symbols have the same meaning than in **Subsections 2.1.1** and **2.1.2**.

2.3.3.1 Single Cycle (SC)

Chevaleyre was the first to establish theoretical results for MAP, from graph theory [8]. First of all, he defined the notion of *cyclic strategy*.

Definition 2.3.2. Cyclic strategy. Let $S = (s_0, \ldots, s_n)$, with $s_n = s_0$ and $n \geq N$ be a closed-path of size $n+1$ nodes visiting all the nodes of graph G. Any strategy $\Pi = \{\pi_1, \ldots, \pi_{N_a}\}$ is a *multiagent cyclic strategy* based on S iff there exists $\{d_1, \ldots, d_{N_a}\} \in \mathbb{N}$ such that $\forall k \in \mathbb{N}$, $\pi_a(k) = s_{(k+d_a) \mod n}$.

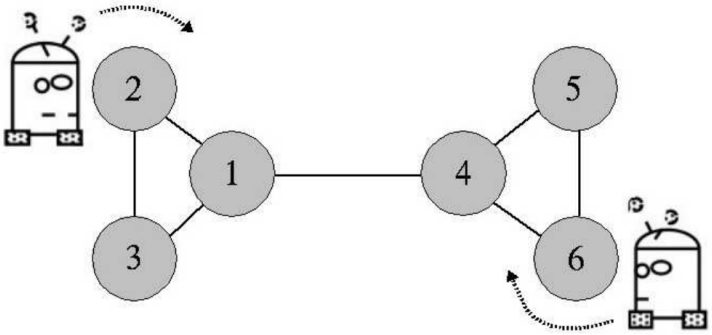

Fig. 2.4 Illustration of a multiagent cyclic strategy. The cyclic strategies of the depicted agents from their current node are $\pi_1 = 2, 1, 4, 5, 6, 4, 1, 3, 2$ and $\pi_2 = 6, 4, 1, 3, 2, 1, 4, 5, 6$, perpetually [8].

In such a definition, $\{d_1, \ldots, d_{N_a}\}$ can be thought of as the distance, in terms of nodes, with the first node of the closed path, that is s_0. For example, in **Fig. 2.4** $d_1 = 0$ and $d_2 = 4$ [8].

Then, in a theorem he has established that for a single agent, the optimal strategy in terms of worst idleness, as defined by **Eq. 2.1.1** or **2.1.5** — the worst idleness or worst interval being identical —, is the cyclic-based strategy based on the closed path which is the optimal solution to the Travelling Salesman Problem (TSP), noted S_{TSP}:

Theorem 2.3.3. *For a single agent, the optimal strategy in terms of the worst idleness is the cyclic strategy set from S_{TSP}.*

He also proved that if N_a agents follow a multiagent cyclic strategy Π and $d_1, ..., d_{N_a}$ are well-chosen, then the worst idleness will be approximatively N_a times lower than the worst idleness of single-agent patrolling on the same cycle:

Lemma 2.3.4. *Let $S = s_0, ..., s_{N_a}$ be a closed-path covering each node of G such that there exists a node $v \in V$ covered exactly once by S. Let $l = c(S)$ be the length of the closed-path. There exists a multiagent cyclic strategy $\Pi = \{\pi_1, ..., \pi_{N_a}\}$ based on this closed-path such that $\frac{l}{N_a} - \max_{(i,j)\in E_S}(c_{ij}) \leq WI_\Pi \leq \frac{l}{N_a} + \max_{(i,j)\in E_S}(c_{i,j})$. Here, E_S refers to the set of edges present in S. Note that l is also the worst idleness of the single-agent cyclic strategy based on S.*

Let S_{Chr} denote the closed path obtained with the Christofide's algorithm. Extending **Theorem 2.3.3** to the multiagent case by arranging agents on the same closed path such that they keep an approximately constant offset while walking, he then proved the following theorem:

Theorem 2.3.5. $WI_{\Pi_{Chr}} \leq 3 \times opt + \max_{e=\{i,j\}\in E}(c_{ij})$

where Π_{Chr} is the multiagent cyclic strategy based on S_{Chr}, and *opt* the worst idleness of the optimal strategy. This strategy is referred to as *Single Cycle* (SC) [8].

Thus, Chevaleyre has proven that the shortest cycle of a graph G is the optimal solution on the worst idleness criterion for a single agent. He also established performance bonds concerning the optimality of multiagent one-cycle-induced strategies. As such, he has showed that determining the optimal multiagent cyclic strategy on the worst idleness criterion is a NP-complete problem, that is to say, it is a NP problem and it is NP-hard, i.e. no polynomial-time algorithms are known to compute an optimal solution to the problem. This is thereupon equivalent to solve the TSP for the current graph G.

However, with dynamic environments or large graphs, cyclic strategies are problematic insofar as dynamic environments need frequent recomputations of the closed walk, whereas large graphs mean exponentially longer run time.

2.3.3.2 Graph-partitioning-based strategies

Chevaleyre introduced the *graph-partitioning-based strategies*, or simply *partition-based strategies* [8], which correspond to multicycle strategies. Such strategies are well-known

for their performances in term of visit frequencies. He has defined a MAP strategy $\Pi = \{\pi_1, ..., \pi_{N_a}\}$ based on a partition P, as a strategy for which each agent $k \in [|1, N_a|]$ visits the nodes of only one region of P. The class of strategies based on the partition P are referred to as Π_P.

Let $opt_{\Pi_{cycle}}$ and opt_{Π_P} be the worst idleness of the optimal cyclic strategy and the optimal strategies based on P. Then Chevaleyre demonstrated the following theorem:

Theorem 2.3.6. $opt_{\Pi_{cycle}} \leq opt_{\Pi_P} + 3 \times \max_{i,j \in V}(c_{ij})$

In this theorem, as well as in the previous one, the term $\max_{i,j \in V}(c_{ij})$ brings to our attention the fact that cyclic strategies are not suited for graphs containing long edges [8]. Cyclic strategies will thereby be preferred when graphs have not long edges connecting far regions.

Finally, from the foregoing theorem, he has laid down a "computable" version of the previous theorem in the form of the following corollary:

Corollary 2.3.7. *Let $P = \{P_1, ..., P_{N_a}\}$ a partition of V. A cyclic strategy Π_{Chr} such as $WI_{\Pi_{Chr}} \leq \frac{3}{2} opt_{\Pi_P} + 4 \times \max_{i,j \in V}(c_{ij})$ is computable in $O(N^3)$.*

Fault-Tolerant Multi-Robot Area Coverage. Fazli et al. created a new robust and fault-tolerant strategy based on graph-partitioning for continuous environment and the *Minimum Spanning Tree* (MST) problem [16]. The complexity of this strategy is $O((N + N_a)^3)$. A spanning tree over the nodes of the graph, is created by means of a graph reduction algorithm. Such a reduced graph is called a *Reduced-CDT*, and enables improving efficiency by minimising the average of total time spent by the agents to travel the graph. The *Multi-Prim* algorithm [11], extending the *Prim*'s one [43], is used to build the MST of a weighted graph in order to compute a *forest* of *partial spanning trees* from a Reduced-CDT graph as input. It is here used to compute as many trees as the number of agents, and finally returns N_a spanning trees.

The *Revised Double Minimum Spanning Tree* (Revised-DMST) algorithm, a variant of *Double Minimum Spanning Tree* (DMST) which takes a tree as input and returns a cycle whose length is double the length of the tree, proceeds as DMST except that it removes each vertex already proceeded and moves to the next vertex never visited before, connecting this one with a shortcut edge. Given that the Revised-DMST algorithm cannot be applied owing to the possibility of a shortcut edge not being in E, the author introduced another algorithm called *Constrained Spanning Tour* (CST). The latter traverses the vertices in the same way as Revised-DMST with the difference that it uses

edges of the original CDT graph as shortcut. Finally, a cycle is created for each partial spanning tree using CST.

Lastly, when an agent leaves the society, all the vertices of its assigned tree are released. Then, all of its *supportive* agents, i.e. agents whose the tree has a bridge with that of the failing agent, expand their trees through the Multi-Prim Algorithm to acquire the released vertices. The following theorem can then be stated:

Theorem 2.3.8. ***Robustness.*** *The approach is robust even if $N_a - 1$ agents leave the society, or in other words, as long as there is at least one agent in the society, N_a being the number of agents.*

This ensures that as long as at least one agent is patrolling, the whole environment will be covered.

Multilevel Subgraph Patrolling (MSP). The *Multilevel Subgraph Patrolling* (MSP) algorithm is a *multilevel* graph-partitioning-based algorithm. It assigns different regions, or subgraphs, to each agent [39]. A multilevel graph partitioning algorithm reduces the size of the graph, i.e., *coarsens* the graph, by collapsing vertices and edges, partitions the smaller graph, and then *uncoarsens* it to construct a partition of the original graph [21]. Finally, the parts are assigned to each agent, and for each one a patrolling path is computed by determining either Euler cycles, or Hamiltonian cycles using heuristics, the longest path, or non-Hamiltonian cycles.

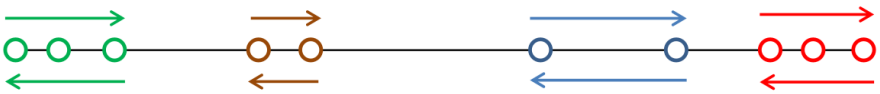

Fig. 2.5 Four optimal closed walks on a chain graph [38]

Left-Induced Partition of Generic Graphs. Portugal et al. introduced the *Min-Max Cost Closed Walk* problem, which corresponds, given a generic graph G and N_a agents, to find a partition $P = \{P_1, ..., P_{N_a}\}$ of V, the set of nodes of G, such as:

$$P^* = \arg\min_{P}(\max L(\pi_a(P_a))) \tag{2.3.7}$$

where $L(P_a)$ is the length of the optimal TSP-cycle on P_a [38]. This problem being equivalent to solving the TSP, it is NP-complete.

Pasqualetti et al. proposed an algorithm, called *Optimal Left-Induced Partition* (LIP), to determine an optimal min-max cost closed walk partition for any chain graphs, as in **Figure 2.5** [36]. Based on the latter, Portugal et al. have proposed a new algorithm called *Left-Induced Partition of Generic Graphs*, to create a strategy Π_{LIP} by extending it to generic graphs, finding first a complete open walk which is thereafter converted into a chain graph noted Γ equivalent to the open walk [38]. It turns out that the open walk must have at most $2|V| - 4$ edges. This constraint ensures the following performance guarantee with respect to the optimal solution [36]:

$$WI_{LIP} \leq 8\frac{|V|-2}{|V|}\ \eta\ WI_{\Pi^*} \tag{2.3.8}$$

with $\eta = \dfrac{\max|c_{i,j}|}{\min|c_{i,j}|}$ and WI_{Π^*} being the worst idleness of the optimal strategy.

Finally, the optimal left-induced partition of Γ, of size N_a, is computed, then each set of the optimal partition is assigned to one agent which will patrol it back and forth.

Evolutionary Heuristic to approximate the Min-Max Cost Closed Walk Problem. Considering the high dependence of the previous algorithm on η, it is expected to rarely reach an optimal solution. In doing so, the authors have then proposed an additional partition-based evolutionary algorithm they call *Evolutionary Heuristic to approximate the Min-Max Cost Closed Walk Problem* (EHP) [38]. This algorithm begins with an initial partition of G of size N_a, building with a classical multi-way graph-partitioning approach. It is then evolved using a mutation and selection scheme that randomly swaps vertices between parts of the partition. Lastly, as previously, each set of the partition is assigned to one agent.

Finally, the authors evaluated both strategies EHP and LIP on three topologies, from 1 to 20 agents, and compared them with two cyclic strategies, the first called *MST Tour*, based on an MST solving algorithm, and the second called HTSP, based on an TSP approximated-solving algorithm. EHP is always better than LIP, but in most cases it is outperformed by HTSP.

PART. *PART* is a self-organised partitioning algorithm for MAP [58]. This algorithm yield a partition between agents as optimal as possible. In the proposed model, agents do not have global knowledge nor the ability to communicate directly with all other patrolling agents. The partitioning process is self-organized so that there is no global

coordination between agents and global state known. First, the partitioning is violated by agents during the partitioning process, i.e. the parts of V do not form a partition, they are not strict nor non-overlapping, because messages induce delays in the system. Then, when agents arrive at a stable configuration each node is associated with at most one agent, and all the subgraphs in the partition are connected. Each agent works then only over its own subgraph without interfering with each other.

PART strategy was tested and compared with CR [29], GBS [40], SEBS [40], described in **Subsection 2.4.1**, and MSP [39] and has turned out to be the best strategy.

2.3.4 Markov-decision-process-based strategies

A *Markov decision process* (MDP) is a decision-making model for any discrete time stochastic control process. A MDP is represented by a tuple $< S, Ac, P, R >$, where:

- S is the set of *states*, namely for a single agent, the nodes of the graph G,
- Ac is the set of *actions*, consisting in moving among nodes,
- $P : S \times Ac \to K(S)$ — where $K(S)$ is the *credal set* of S located — i.e. the set of probability distributions over S —, is the function of probability distributions making up the *state transition function*: for each state $s \in S$, this function returns a probability distribution $P(s, a)$, where $\forall s' \in S$, $P(s, a)(s') = P(s, a, s')$ is the probability to reach the state s' from s by applying action a,
- $R : S \times A \to \mathbb{R}$, is the *immediate reward*, also known as expected immediate reward.

2.3.4.1 Multiagent patrolling and reinforcement learning

Santana et al. addressed for the first time the MAP problem as a *reinforcement learning* (RL) problem [53]. They demonstrated that an efficient cooperative behaviour can be achieved by means of Q-learning, by modelling MAP as a *Semi-Markov Decision Process* (SMDP), which is merely a generalisation of MDPs where actions may take a variable amount of time. In this model the area to patrol is still depicted as a graph, and the model being an MDP, it is defined by the tuple (S, Ac, P, R) outlined above, where $\forall t \in \mathbb{N}^*$, the state s_t corresponds to both the positions of all of the agents and the idlenesses of all the nodes of G. Then, let:

- $\forall a \in A$ an agent, $p_a(t) \in V \cup E$ be the position of a at time t,

- $\forall a \in A$, $\forall t \in \mathbb{N}^*$, $\forall p \in V \cup E$, $i_p(t)$ is the true idleness of p at time t; if p is an edge, then $i_p(t) = 0$, otherwise $i_p(t) = i_v(t)$ such as $p = v \in V$,

The idleness of $p_a(t)$ at time t, $i_{p_a(t)}(t)$, is then used as reward function, and the multiagent policy learned by the Q-learning algorithm is that which maximises the sum of the discounted reward after a T-period simulation run, $\forall T \in \mathbb{N}$:

$$\sum_{a \in A} \sum_{t=0}^{T} \gamma^t \times i_{p_a(t)}(t) \tag{2.3.9}$$

where $\gamma \in [0, 1]$ is a discount factor.

To take advantage of Q-learning the sum to maximise in **Eq. 2.3.9** should represent the real state of the system, leading the true idleness of the node visited by $a \in A$ to be used as a reward. Otherwise, agents would try to maximise an internal sum, and each agent will not make the effort to help other agents to maximise their rewards. To that end, a scheme of flag communication, which can be interpreted as communication through environment, is implemented so as each time an agent visits a node it places a flag containing the current time step which can be seen by any other agent that will stand on this node.

It is worth noting that the strategy proposed here is agent-independent: the same strategy π is executed by each agent a, so that at each time $t \in [|1, T|]$, the decision of a noted d_a is such as $d_a(t) = \pi(p_a(t), i_a(t))$.

Two Q-learning-based strategies were modelled using two different communication schemes:

- *Black-Box Learner Agent* (BBLA): agents communicate only by placing a flag on each node they visit,
- *Grey-Box Learner Agent* (GBLA): agents communicate by placing a flag on each node they visit and can also communicate their intention of actions to the other ones.

Results show that globally GBLA is the best strategy when compared with BBLA, CR and Heuristic Cognitive Coordinated.

Finally, although being an interesting and seminal work associating MAP with MDPs and RL the approach of the authors assume that agents communicate by leaving

messages on the nodes of the graph; such a strong hypothesis leads to unrealistic communication models similar to those established in ACO-based MAP, and by doing so, hardly practicable in real life.

2.3.4.2 Continuous Time Multiagent Patrolling

Marier et al. addressed the problem of continuous time MAP they cast as a *Generalised Semi-Markov Decision Process* (GSMDP) [30]. The problem state is fully observable, i.e. every agent has the same complete information to make its decision. The *freshness* is defined from idleness as being the quantity $k_v^t = b^{i_v(t)}$ with $0 < b < 1$, which enables having a bounded quantity depending on idleness. The authors point out it can be interpreted as the expected value of a Bernoulli random variable that is worth 1 if vertex v is observed correctly and 0 otherwise.

The state s_t, $\forall t \in \mathbb{R}^+$, corresponds then to:

- the position of every agent, represented by the vector of all agent positions v_t,
- the freshness of every vertex, represented by the fector of all node freshnesses k_t,

so that s_t is such as $s_t = (v_t, k_t) \in \mathcal{S} = V^{N_a} \times [0,1]^{|V|}$, where $\mathcal{S}$ is the state space.

The actions correspond to selecting a next node to visit in the neighbourhood. An agent selects an action each time it stands on a vertex, and the transit time being stochastic, the time of arrival to the goal node is not known in advance.

The immediate reward R is then defined, $\forall t \in \mathbb{R}^+$, in terms of k_t so as the rate at which the reward is gained is defined by:

$$dR = w^T k_t \; dt \tag{2.3.10}$$

where w is a vector of weights.

Finally, to solve this GSMDP problem, optimisation techniques were applied to efficiently find the optimal paths for the agents on the graph. Particularly, an algorithm with communicating agents, named *Coordinated Anytime Error Minimisation Search* (C-AEMS), was compared with a simple reactive algorithm called *Reactive Markov Decision Process* (RMDP), which has proven to be barely outperformed by C-AEMS: differences of performance are negligible. This work, in fact, highlights that simple reactive algorithms can show the same performance as more computationally complex approaches.

2.3.4.3 Cooperative Multiagent Patrolling based on DEC-POMDP

Beynier introduced a new formalisation of adversarial MAP under uncertainty based on *Decentralised Partially Observable Markovian Decision Process* (DEC-POMDP) [7]. DEC-POMDPs are convenient to model a decision problem under uncertainty with partial observability of the system. Although addressing an adversarial MAP, some aspects of this work are germane to the temporal MAP.

Agents make a decision when standing upon a node, then the decision is the next node to visit.

In this work, uncertainty arises from imperfect action executions and limited observations consisting in:

- possible transit times, refered to as travel durations by the author, as in the model of Marier et al. [30], leading to assign a probability distribution to each edge,
- detecting *illegal actions*, which correspond to the probability distribution $PI_v(t)$, that is the probability that at least one of adversaries initiate an intrusion on the node $v \in V$ at time step $t \in \mathbb{N}$.

$\forall t \in \mathbb{N}$, s_t, the state at time t, is defined such as:

- p is the position of each agent,
- $int \subset V$ is the set of nodes where an illegal action has just been observed,
- $idle$ contains the idleness of each node,
- δ contains the elapsed time of each current move.

The *transition function* corresponds to the transition probabilities which are defined from probabilities on travel durations and probabilities of detection of illegal action.

The *reward function* is defined to reward detected illegal actions. For all action $a \in A$ from a state $s_t = (p, int, idle, \delta)$ leading to a potential next state $s'_t = (p', int', idle', \delta')$, the reward $R(s'_t|a, s_t)$ is defined as:

$$R(s'_t|a, s_t) = \sum_{v \in int'} R^D(t_v) + \sum_{v \in p' and \notin int'} R^P(v, idle_v) \tag{2.3.11}$$

where $R^D(t_v) \in \mathbb{R}^*_+$ denotes the reward of detecting an illegal action on the node v and $R^P(v, idle_v) \in \mathbb{R}^*_+$ is the reward of the node $v \in V$ without detecting any illegal action.

$R^P(v, idle_v)$ is proportional to the idleness of v, leading the agents to patrol the nodes with the highest idleness first.

Interestingly, when setting $\forall v \in V, \forall t \in \mathbb{N}$, $PI_v(t) = 0$, it follows $int = \emptyset$ during all the patrolling mission, resulting in a temporal MAP with non-deterministic transit times.

Finally, a distributed evolutionary algorithm was investigated to compute the policy of the patrolling strategy. Such a distributed algorithm requires the agents to communicate their observations about detected illegal actions. Strategy resulting from this evolutionary algorithm turns out to be better than the optimal solution computed using the *Multiagent Decision Process* toolbox [35].

2.4 Machine-learning-based strategies

A first machine learning strategy based on the SMDP model is presented in **2.3.4.1**. The purpose of this section is to present a set of machine learning strategies which do not rely on MDP.

2.4.1 Bayesian learning

In simplified terms, *Bayesian learning*, also knowns as *Bayesian inference*, consists in inferring the probability of having a certain value for a parameter given a sample. In what follows, a simplified model of Bayesian learning is presented.

Let:

- X be a sample, i.e. n observed data represented by n real random vectors: $X = (X_1, \ldots, X_n)$,
- θ be the vector of parameters of the distribution of X, that we want to infer.

Following the Bayes' rule, it follows:

$$P(\theta|X) = \frac{P(X|\theta)P(\theta)}{P(X)} \tag{2.4.1}$$

where:

- $P(\theta)$, the *prior* distribution, represents the information about the parameters known beforehand,
- $P(X|\theta)$, the *likelihood* distribution, represents what is expected in terms of observed data if the parameters were known; this is what is learnt and update as part of the Bayesian learning,

- $P(\theta|X)$, the *posterior* distribution yields, in the Bayesian learning framework here laid down, the most accurate information about the value of θ.

Finally, the value of θ maximising the posterior distribution $P(\theta|X)$ is the most probable value having generated the observed data X.

2.4.1.1 Greedy Bayesian Strategy (GBS) and State Exchange Bayesian Strategy (SEBS).

Portugal and Rocha proposed two distributed MAP strategies based on the Bayesian inference formalism [40]. This method is implemented with the aim of making the MAP system more adaptive and flexible. In both strategies agents update their instantaneous idlenesses by communicating with each other each time an agent reaches a vertex

The first proposed strategy, *Greedy Bayesian Strategy* (GBS), applies Bayesian learning based on two types of Real-valued Random Variables (r.r.v.):

- $\forall v \in V$ the Bernoulli variable $move_v \in \{0, 1\}$, which represents whether the agent will move to a neighbour vertex v,
- $G_v \in \mathbb{R}^+$ which represents the gain that the agent expects to obtain whether it moves from its current vertex to the neighbour v.

In this strategy, the *prior* $P(move_v)$ is assumed to be uniform over all neighbouring vertices, while $P(G_v \mid move_v)$, the *likelihood*, follows an exponential law when $\{move_v = 1\}$:

$$\begin{aligned} &\forall G_v < M, \\ &\qquad P(G_v \mid move_v = 1) = L\ exp(\frac{ln(1/L)}{M}\ G_v) \end{aligned} \tag{2.4.2}$$

where $L, M \in \mathbb{R} : L,\ M > 0$ are given values.

Here, the event $\{move_v = 0\}$ is never considered. This is a strong assumption as far as $P(G_v \mid move_v)$ exists when $move_v = 0$, for example whether another agent moves onto v. That leads to obviate the model of $P(G_v \mid move_v = 0)$, resulting in an incomplete model. Also, according to **Eq. 2.4.2** and as stated by the authors, the likelihood distribution is set beforehand, leading to never be updated as new information becomes available, disabling any possible Bayesian learning.

As to $P(G_v)$, the *marginal likelihood*, it is regarded by the authors as "a normalisation factor which can be omitted for the sake of simplificity". This is yet again a strong

Algorithm 1: Greedy Bayesian Strategy (GBS)

while *true* **do**
  `add(`v_n *to* x_r`);` // current vertex
  `write_msg_arrival_to(`v_n`);`
  forall the $v_i \in N_G(v_n)$ **do**
    $G_i \leftarrow c\left(\frac{\mathcal{I}_{v_i}(t)-\mathcal{I}_{v_i}(t+\Delta t)}{|e_{ni}|}\right)$;
    $P(G_i|move(v_i)) \leftarrow L \cdot \exp\left(\frac{\ln(1/L)}{M} G_i\right)$;
    $P(move(v_i)|G_i) \leftarrow \frac{P(move(v_i))P(G_i|move(v_i))}{P(G_i)}$;
  // Next vertex is the neighbor of the current vertex with highest posterior probability.
  $v_{n+1} \leftarrow$ `argmax(`$P(move(v_i)|G_i)$`);`
  while *move_robot to* v_{n+1} **do**
    `read_msg_arrival_to(`$\mathcal{V}$`);`
    `update(`$\mathcal{I}_{\mathcal{V}}(t)$`);`
  $v_n \leftarrow v_{n+1}$;

Fig. 2.6 Greedy Bayesian Strategy (GBS) algorithm

assumption to the extent that the decision-making is carried out $\forall v \in \mathcal{N}_G(v_0)$, over the best probability of $\{move_v = 1\}$, with $\mathcal{N}_G(v_0)$ beging the neighbourhood of v_0, and not over the support of $move_v$, that is $\{0, 1\}$. Indeed, $P(G_v)$ can only be omitted when the decision-making is carried out over the same r.r.v. G_v, but not in any case over different r.r.v.'s, considering that $\forall v \in \mathcal{N}_G(v_0), P(G_v)$ varies, unless to assume that $\forall v \in V,\ P(G_v)$ is constant. Therefore, $\forall v \in V, P(G_v)$ should have been calculated as follows:

$$P(G_v) = P(G_v \mid move_v = 1)P(move_v = 1) + P(G_v \mid move_v = 0)P(move_v = 0) \quad (2.4.3)$$

However, it is here unclear whether the authors use $P(G_v)$ or not. First, they appear to confuse $P(G_v)$ with $P(G_v \mid move_v)$ by defining first $\forall v \in \mathcal{N}_G(v_0),\ P(G_v)$, then by using this definition for G_v given that $\{move_v = 1\}$, i.e. $P(G_v \mid move_v = 1)$. Finally, as shown on **Figure 2.6** depicting the GBS algorithm, they seem finally to use $P(G_v)$ in the Baye's rule, in spite of having stated previously that it can be omitted as a normalisation factor.

The *posterior* probability is then estimated for $move_v = 1$ via the Bayes' rule:

$$P(move_v = 1 \mid G_v) = \frac{P(move_v = 1)P(G_v \mid move_v = 1)}{P(G_v)} \quad (2.4.4)$$

leading to choose the $v \in \mathcal{N}_G(v_0)$ with the maximum $P(move_v = 1 \mid G_v)$.

Eventually, this model seems somewhat abstruse. As stated by the authors, "since the model assumes a uniform prior and considers only one likelihood function which is fixed, the decisions taken in GBS are equivalent to moving to the adjacent vertex with maximum instantaneous idleness". This raises questions about the relevance of Bayesian inference, which is, in this context, no longer necessary. Also, it is worth noting that, based on the fact set out above and as agents communicate only with each other when they reach a vertex, this strategy can be thought of as a communicating CR.

GBS agents being only interested in obtaining the best reward for themselves while neglecting the global objective of the patrolling mision, the authors have extended the strategy to reduce interferences between agents during the mission, given rise to the *State Exchange Bayesian Strategy* (SEBS). In addition to the r.r.v.'s defined previously, SEBS agents take now into account in their decision process a new discrete r.r.v., noted $S_v, \forall v \in \mathcal{N}_G(v_0)$, which represents the number of agents that intend to visit a given neighbour v. As previously, the r.r.v. G_v remains problematic in the context of the authors' model and S_v suffers from the same failings:

- the new r.r.v. S_v is set beforehand, obviating any Bayesian inference,
- $P(S_v)$ and $P(S_v \mid move_v = 1)$ are confused,
- no assumptions, and no justifications of such an assumption, about the independence of G_v and S_v, $\forall v \in \mathcal{N}_G(v_0)$, while this assumption is implicitly made in the Baye's rule used by the authors to compute the posterior $P(move_v \mid G_v, S_v)$,
- the decision is made over $P(G_v)$ and $P(S_v)$, $\forall v \in \mathcal{N}_G(v_0)$ instead of the supports of S_v and G_v,
- it remains unclear whether the authors finally omit the normalisation factors $P(G_v)$ and $P(S_v)$ or not.

Results show that, for 6 agents and more, SEBS and GBS outperform all the other strategies on the final graph average idleness criterion. However, for the same tests CR outperforms most of the time HPCC, contradicting all the domain's results [3][42]. It is regrettable that only the final graph average idleness is considered, leading to a lack of information about performances while the mission is runnng. Finally, the authors give no information about the r_H and r_P parameters for the HPCC strategy that they experimented.

2.4.1.2 Concurrent Bayesian Learner Strategy (CBLS)

Following the strategies outlined above, Portugal et al. also attempted to propose another distributed MAP strategy based on the Bayesian inference formalism [37]. In *Concurrent Bayesian Learner Strategy* (CBLS), the communication system is distributed: each agent computes its values internally, value communicated thereafter to its other teammates each time it arrives to a new vertex. CBLS agents apply as well, what is called a Bayesian learning by the authors, relying on two r.r.v.'s:

- $\forall v_0 \in V$, $\forall v \in \mathcal{N}_G(v_0)$, $move_v \in \{0,1\}$ is a Bernoulli variable which represents whether the agent will move to a neighbour vertex,
- $\theta_{0,i} \in \theta : card(\theta) = 2\ card(E)$ called *arc strength*, represents the suitability of travelling $e_{v_0,v}$ to reach v.

Here, $P(move_v)$, the *prior*, depends upon the idleness of v [41]:

$$P(move_v) = \frac{\omega \cdot \bar{i}_v + (1-\omega) \cdot \max(\bar{i}_w : w \in \mathcal{N}_G(v))}{\sum_{k \in V} \omega \cdot \bar{i}_{v_k} + (1-\omega) \cdot \max(\bar{i}_w : w \in \mathcal{N}_G(v_k))} \tag{2.4.5}$$

where $\forall v \in V$, $\bar{i}(v)$ is the average idleness of v and $\omega \in [0,1]$.

This definition is justified by the intent of selecting primordially the vertex with the higher average idleness which has itself a high average neighbourhood idleness. This method is called *vertex look-ahead*.

As the mission proceeds, the agent computes a reward for each arc $\{v_0, v\}$ as follows:

$$\gamma_{v_0,v} = S_{v,v_0}(C_v, I_V(t)) \cdot (1 - \mathcal{H}(move|\theta)) \tag{2.4.6}$$

with:

$$\mathcal{H}(move_v \mid \theta) = \frac{H(move \mid \theta)}{log_2(deg(v_0))} \tag{2.4.7}$$

being the normalised conditional entropy over the neighbours of v_0, $S_{v,0} \in \{-1, 0, 1\}$ depending upon C_v, the number of visits to v, $I_V(t)$, the vector of true idleness, and the

degrees of v_0 and v gives the reward sign.

Then, $\theta_{v_0,v}$ is updated via a 1-step-horizon reward:

$$\theta_{v_0,v}(t) = \theta_{v_0,v}(t-1) + \gamma_{v_0,v}(t) \tag{2.4.8}$$

Finally, the likelihood ditribution is learnt as follows:

$$P(\theta_{v_0,v} \mid move_v) = \frac{\theta_{v_0,v}}{\sum_{j}^{|E|}\sum_{k}^{|E|}\theta_{j,k}} \tag{2.4.9}$$

This quantity is updated at each decision step using accumulated experience. The posterior probability $P(move_v \mid \theta_{v_0,v})$ is then calculated using the Bayes's rule as in 2.4.4.

As before [40], this model has some pitfalls: the denominator term $P(\theta_{v_0,v})$ is regarded as a normalisation factor, whereas the decision-making is not carried out over the r.r.v. $\theta_{v_0,v}$, justifying this simplification, but over $\{\theta_{v_0,v}, \forall v \in \mathcal{N}_G(v_0)\}$.

Results finally showed that CBLS outperforms GBS and SEBS as well as HPCC and CR on the final average idleness criterion.

2.4.2 Neural-learning-based patrolling

For the reader not familiar with ANNs, **Appendix A** provides an overview on the basics and history of ANNs.

2.4.2.1 Modelling a dynamic landscape

Few works addressed the problematic of using ANNs in the context of MAP. Amongst related works, Guo et al. studied the use of ANN-based methods for planning a complete coverage patrolling path [19]. In that work, the area to patrol is discretised into fixed radius disks that can be thought of as nodes to visit. Then, each neuron, depicts a region activated negatively or positively as a function of either the presence of obstacle, or the absence of a visit by the patrol, respectively. The activities of all the neurons constitute a dynamic landscape such that the non-visited regions attract globally the agent in the entire space, whereas the obstacles locally repel the agent to avoid collisions. Attractive regions, which correspond to regions without visit, can be regarded as regions with a high idleness. The type of neural network used in that work is germane to estimation of global features of the environment by agents, which is one of the approach used in this disseration. However, in this work the use of ANNs by agents implies no restrictions

with regard to communication, whereas in our framework agents do not communicate systematically: either agents do not communicate, or they communicate only when they are within the range of each other; they act in a decentralised way.

2.4.2.2 Hive brain

D'Ambrosio et al. developed a new communication scheme they called *hive brain*, as part of cooperative multiagent learning [14]. In this scheme, each agent is endowed with a neural network directly connected to nodes of the others agents' internal ANN, whose weights of connections are evolved. As stated by the authors, this technique is drawn from an interesting physical phenomenon called *odd sympathy* [6], which is the tendency of pendulum clocks to synchronise when mounted near each other owing to a small amount of physical information transferred between the pendulums. Thus they elaborated the hive brain analogically, where the agents learn to synchronise by training their respective ANN in a agent simulator; the training is performed using an evolutionary algorithm. In our perspective, this work presents the same problem as the previous one, that is to say the implicit use of communications in the simulator between agents, to feed the *brain* of one agent from another one.

2.4.2.3 Predicting states of a finite-state machine

Sales et al. developed an autonomous patrolling system composed of four intelligent agents that can freely move through an indoor environment and detect intruders [51]. The agents are endowed with a localisation/navigation system composed of an ANN, used in combination with a Finite State Machine (FSM) whose the states correspond to the key features of the environment. The FSM associates a sequence of actions to execute with a sequence of states, and the ANNs process the sensors' data to identify and classify the FSM states (current and transitions), and to determine the actions to perform. After being trained offline to identify the key features of the environment such as corridors, intersections and turns, they are fed with data obtained from agents' sensors to output the FSM states. Lastly, in this work, each agent calculates the shortest path using the A^* algorithm while taking into account its teammates, to reach the intruder's position when it is detected.

2.5 Conclusion

The analysis of the state of the art presented in this chapter leads to the conclusion that the problematic consisting in learning decentralised MAP strategies from data generated by any centralised strategies is original. Moreover, it allows making up a framework for this work, whose the properties are:

- the scope is that of temporal MAP,
- assessment of studied strategies is performed using benchmark topologies experimented to a large extent by the MAP community,
- the number of agents does not vary throughout a mission,
- performance measures based on idleness and interval are retained and they are normalised with respect to the number of agents N_a to compare different numbers of agents,
- strategies cannot communicate through the environment,
- learning data can be provided by the HPCC strategy that is an excellent candidate of strategy to be decentralised considering it is a centralised, coordinated and high-performance strategy, and,
- CR can be used as a reference for the assessment of strategies without communication between agents: any new decentralised strategy ought to outperform CR.

Finally, with regard to supervised learning, models studied in the following will range from simple models with few parameters, as in Bayesian learning, to complex ANNs, such as MLPs and LSTMs.

Chapter 3

Model, methodology and implementation

This chapter describes the model of the MAP problem, as well as some definitions pertaining to it, which must be introduced in this dissertation for the sake of clarity. Then, the characteristics and structure of MAP data, namely data generated by MAP runs, are described. Finally, the general methodology used in the remainder of this dissertation is laid out, the tools developed for this purpose are described, then the data used for learning are presented.

3.1 Model and definitions

This section defines the model of the MAP problem and certain of key ideas pertaining to it. This section is a complement of what has already been exposed in **Chapter 2**.

3.1.1 Model of the MAP problem

Unlike the methodology presented in **Section 2.2**, the present model distinguishes the notions of patrolling configuration and patrolling scenario. Here, a *patrolling configuration*, *MAP configuration* or simply *configuration*, noted K, is any unordered pair $K = \{G, N_a\}$. For example, $\{Grid, 15\}$ is a patrolling configuration of 15 agents on the topology Grid. Then, a *patrolling scenario*, *MAP scenario* or simply *scenario*, is any triplet $\{\Pi, G, N_a\}$ such as Π is a MAP strategy. In fact, a MAP scenario is the pair of a MAP strategy and a MAP configuration. For example, $\{HPCC, Grid, 15\}$ is a MAP scenario of 15 HPCC agents on the topology Grid.

The model of the MAP problem can now be defined.

A MAP problem is an optimisation problem which has to be optimised according to:

- a given evaluation criterion, noted C; such a criterion defines the characteristics required by the strategy to design, and,
- a MAP configuration.

Such a problem is noted $MAP(C, K)$. In this dissertation a MAP problem is an optimisation problem whose the objective is to optimise an evaluation criterion C for a configuration K. By doing so, it follows that solving this problem is equivalent to find the optimal multiagent strategy Π with respect to C in the configuration K, i.e. the strategy Π which minimises C in K, as follows:

$$\Pi^*_K = \arg\min(\ C(K, \Pi) : \Pi \ \text{ a MAP strategy }) \tag{3.1.1}$$

By definition, such a strategy verifies:

$$C(K, \Pi^*_K) = \min(C(K, \Pi) : \Pi \ \text{ a MAP strategy }) \tag{3.1.2}$$

and Π^*_K is thus the best strategy for the problem $MAP(C, K)$.

In the remainder of this dissertation, agents are assumed to know the *a priori* topology of the environment.

Moreover, in **Chapter 7**, coordination between agents will be explicitly considered by means of an *interaction scheme*. In this work, non-centralised strategies have to manage only limited communication ranges, that is agents are able to communicate iff they are within a certain range from each other.

3.1.2 Definitions

Some fundamental notions used in the remainder of this dissertation are defined in this subsection.

3.1.2.1 Mission

In order to assess strategies for a given MAP configuration, it is relevant to test them for several *random starts*, that is to say for several instances of this configuration where

agents start the mission from different positions on the graph. Such an instance is termed *mission*.

Definition 3.1.1. Mission. A *mission*, *execution* or *run*, is an execution of a MAP scenario with specific initial conditions. These initial conditions correspond to a positioning of agents on nodes of the graph to patrol. For non-deterministic strategies the seeds of random sorting functions are also part of the initial conditions in the present work.

Initial positions on edge are not considered in the present work. Note also that for a given strategy Π, $C(K, \Pi)$ is estimated practically by averaging the performance measure C of Π over several missions.

3.1.2.2 Idleness types

In this work, three types of idleness are considered:

- $\forall a \in A, \forall v \in V$, the *individual idleness* of the node v for agent a at time $t \in \mathbb{N}^*$, noted $i_v^a(t)$, is defined as being the time elapsed since its last visit to this node,
- in a context of communication, the *shared idleness* for the node v of agent a corresponds to the minimum individual idleness of v among those the agent has received from other agents and its own individual idleness,
- the *true idleness* of the node v at time t, noted $i_v(t)$, corresponds to the time elapsed since the last visit of any agent to v; it is defined, by construction such as:

$$\forall t \in \mathbb{N}^*, \forall v \in V,\ i_v(t) = \min(i_v^a(t) : a \in A) \tag{3.1.3}$$

Then, at each time step t, individual idleness can be updated by each agent, and true idleness is updated by the simulator. Performance measures based on idleness or interval, presented in **Subsection 2.1.3**, are calculated on the basis of true idleness dynamics. It is worth noting that in the case of perfect communication between agents sharing all information concerning idleness using their interaction scheme, shared idlenesses of any agent is equal to true idlenesses.

3.1.2.3 State

In the model of the MAP problem, $\forall t \in \mathbb{R}^+$, a state at time t noted s_t is defined as being the tuple $< P(t), i(t) >$, where:

- $P(t)$ is the list of positions of all the agents at time t,
- $i(t)$ is a vector which represents the true idleness of every node of the patrolled graph G at time t, such as $i(t) = (i_1(t), \dots, i_N(t))$.

The state s_t can also be referred to as *global state* at time t.

Then, each agent $a \in A$ can have a partial knowledge of the state at time t, noted s_t^a and termed as *local state*. s_t^a is the tuple $< p^a(t), i^a(t) >$, where:

- $p^a(t) \in V \cup E$ stands for the position of agent a at time t,
- $i^a(t)$ is a vector of individual idleness computed by agent a according to its visits: $i^a(t) = (i_1^a(t), \dots, i_N^a(t))$.

3.1.2.4 Strategy types

In **Chapter 2**, an axis ranging from reactive to cognitive strategies with an infinity of strategies between this two *ideal strategy types* has been referred. One objective of this dissertation is to propose and study new strategy types.

A strategy is qualified as *centralised* when agents exchange information with a central coordinator, i.e the coordinator constitutes the centre of the changing and mobile network of communication established by agents, and more precisely, it constitutes the central place of the decision process. In this work, among the non-centralised strategies, namely strategies without centre, *decentralised* strategies are distinguished from *distributed* strategies.

Decentralised strategies consist in algorithms without communication, i.e. agents do not communicate. In such strategies, the decision-making process is totally decentralised, there is not any coordination, and thereby not any centre.

In contrast, although without centre, *distributed strategies* agents may communicate with each other, and thereby coordinate. In fact, they form a dynamic communication network, and if there exists a path between two agents in this network, then both agents can communicate; therein the strategy is distributed and, in fact, clustered: agents form communication clusters. Interestingly, a decentralised strategy can be regarded as a distributed strategy without communication, as depicted in **Fig. 3.1**. Note that a distributed algorithm is not necessarily a distributed strategy. HPCC is a distributed algorithm insofar as the coordinator needs to communicate with the patrolling agents to make decisions. However, it is not a distributed strategy.

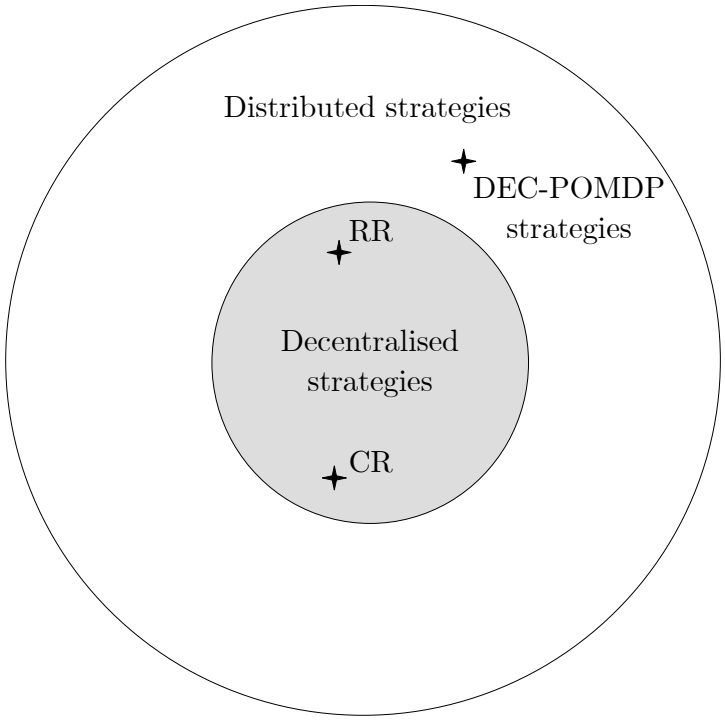

Fig. 3.1 Set of distributed strategies.

Based on the facts set out above, we propose a new axis orthogonal to the reactive-cognitive one: the decentralised-centralised axis, where distributed strategies are somewhere between the centralised and decentralised *ideal-strategies*.

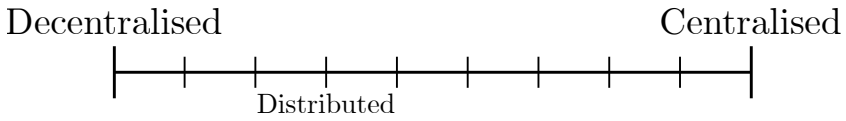

Fig. 3.2 The centralised-decentralised distinction defines a practical axis to evaluate the autonomy of agents with regard to each other.

It is then worth noting that reactive strategies are not necessarily decentralised, although in this dissertation the CR strategy, which is chosen as the decentralised representative to be compared with, is also a reactive strategy, by definition. Likewise, cognitive strategies are not necessarily centralised, and conversely, although in the following the HPCC strategy, which is the centralised representative, is also a cognitive strategy, by definition.

Finally, a *machine learning strategy* is any strategy based on a *statistical* or *machine learning model*. More specifically, when the machine learning model is an ANN such a

strategy is regarded as a *neural strategy*.

3.2 Types of stationary strategies and structure of resultant data

In this section, different types of stationary strategies are described, then the characteristics and structure of data generated by any MAP strategy during a run are laid out.

As shown in **Table 3.1**, any strategy gives rise to several types of sequences:

- *decision sequences* $d_a(t)$, that represent, $\forall t \in [|1, T|]$ such as $T \in \mathbb{N}^*$ is the duration of the mission, and for each agent $a \in A$, the decision applied by a at time t, that is to say in MAP the next node to visit,
- *state sequences*, that represent the state of the system for each time t, whether local states $s_a(t)$, i.e. those observed or computed by each agent a on its own, or the global state $s(t)$; for a certain family of MAS, local states can be regarded as a distribution of the global one, so that there exists a procedure g such as $s(t) = g(s_1(t), \ldots, s_{N_a}(t))$. For example, in MAP g corresponds to both the concatenation of agent positions and the vector of true idleness computed according to **Eq. 3.1.3**.

Agent decisions	d_1 $\vdots$ d_{N_a}	$d_1(1) \quad \ldots \quad d_1(T)$ $\vdots$ $d_{N_a}(1) \quad \ldots \quad d_{N_a}(T)$
Global states	s	$s(1) \quad \ldots \quad s(T)$
Local states	s_1 $\vdots$ s_{N_a}	$s_1(1) \quad \ldots \quad s_1(T)$ $\vdots$ $s_{N_a}(1) \quad \ldots \quad s_{N_a}(T)$

Table 3.1 Output of a MAP strategy run

Usually, a *stationary* policy or strategy Π is defined as a function mapping a state to a decision: $(d_1(t), \ldots, d_{N_a}(t)) = \Pi(s(t))$, more precisely to the decision made by every agent at time t; this is exactly what is defined in **Def. 2.3.1**, where a multiagent strategy for N_a agents is defined as N_a single-agent strategies. However, such a mapping from s

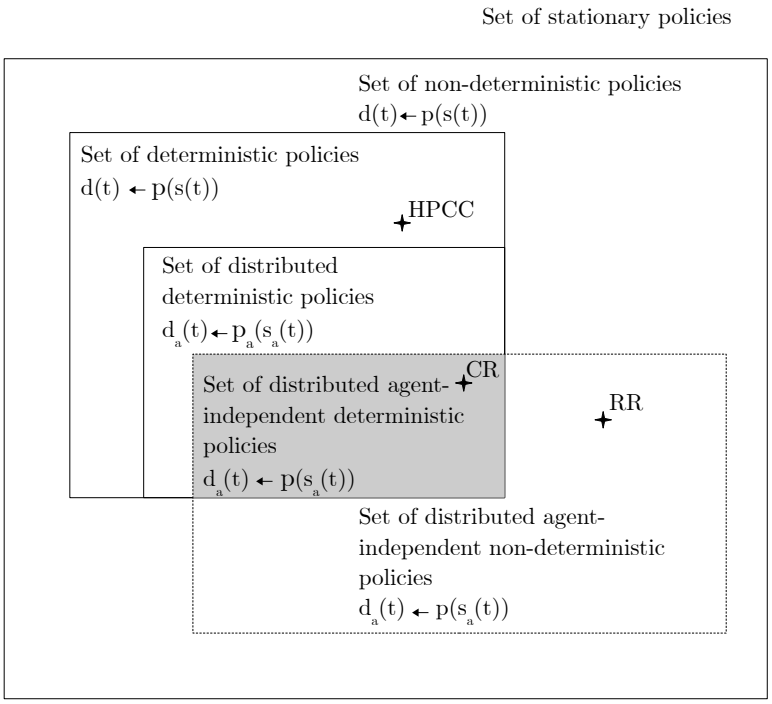

Fig. 3.3 Set of stationary policies, deterministic (π) and non-deterministic (p) where p is a random procedure.

to d is only possible for centralised strategies, where the central decision node has access to the global state $s(t)$, i.e. it is fully-informed, otherwise the global state would not be known.

Fig. 3.3 depicts some types of stationary policies. Interestingly, strategies developed in this dissertation up to **Chapter 6**, are in the form of $d_a(t) = \pi(s_a(t)), \forall a \in A$, namely they are agent-independent and decentralised, whereas in a large number of approaches of the literature — particularly in graph-partitioning-based strategies presented in **2.3.3.2** and in the DEC-POMDP model presented in **2.3.4.3** — they are in the form $d_a(t) = \pi_a(s_a(t))$, that is to say decentralised and agent-dependent.

Finally, if C can be expressed as a summation over time of individual costs depending only on state and decision, then it is known that for $T \to +\infty$, Π^* is inside the set of deterministic policies. It is true for $C = Iav$, without adding any other state variable to the model of the MAP problem; see for example **Subsection 2.3.4**, **Eq. 2.3.9** and **Eq. 2.3.11**. It is unlikely that under the same conditions Π^* be inside the set of distributed agent-independent deterministic policies. Therefore, what is searched for here is a sub-optimal policy. In order to be less constrained in this search it is interesting to consider a larger set of policies: the set of decentralised agent-independent non-deterministic policies where, as shown in **Fig. 3.3**, $d_a(t) \leftarrow p(s_a(t))$, with $p(.)$ not being a function

but a procedure involving randomness.

From the foregoing, two natural and germane use cases of machine learning models for the family of MAS here considered emerge:

- approximating an agent decision by a function of the previous decisions of the same agent $d_a(t-t'), \forall t' \geq 1$, so that there exists a procedure f such as $d_a(t) = f(d_a(t-1), \ldots, d_a(1))$; in MAP, this corresponds to approximating Π for predicting the next node to visit at time t with respect to the previous ones,
- estimating $s(t)$ from $s_a(t)$, which is partly equivalent to approximate g, and using $\Pi(\hat{s}(t))$ to derive the decision, where $\hat{s}(t)$ is the estimated global state; in MAP, this corresponds to estimating true idleness for each node with respect to the individual one, which is partly equivalent to approximate g.

Deterministic or not, decision predictors and state estimators have to be calibrated from data generated by high-performance strategies; such strategies are also generally centralised. In doing so, two types of data have been defined for the two use cases:

- *binary paths*, sequences of *binary positions*, i.e. nodes represented in one-hot[1] format; used in this dissertation to train some networks in **Chapter 4** and **Chapter 5**,
- *on-vertex idleness sequences*, sequences of idleness vectors retained only when agents stand on a vertex, that is to say when they have to make a decision, and, to each on-vertex idleness vector is associated its true idleness counterpart; this use case is used in this work to train some machine learning models in **Chapter 6** and **Chapter 7**.

From now on, any strategy Π used to calibrate a state estimator or a decision predictor as part of a decision process will be qualified as *model strategy*.

3.3 Methodology

The aim of this work being to decentralise and distribute a centralised MAP strategy, a new methodology extending that presented in **Section 2.2** is required. Such a methodology is presented in this section.

First, to decentralise the centralised strategy, the methodology is deployed as follows:

[1] A one-hot vector is a binary vector with only one 1.

1. Selecting an evaluation criterion C to optimise; the criterion, in fact, defines the needed type of strategy,
2. Selecting a centralised decision strategy to decentralise; this strategy is regarded as the model strategy,
3. Acquiring data generated by the model strategy,
4. Selecting a statistical model appropriate to the data,
5. Training the statistical model on the data previously acquired,
6. Creating the new decentralised process: the trained statistical model is deployed into agents; each agent makes a decision using the output of its statistical model,
7. Assessing the new decentralised decision-making process in simulation.

Note that because the selection performed at step 4 is evaluated at step 7, step 4 to 7 may be performed in parallel for several models. Moreover, if a resultant distributed process is aimed for, then communication between agents can be laid down, by definition of a distributed system, and domain-specific distributed algorithms may be designed to improve its performance.

In this dissertation, this methodology is concretely deployed in the context of MAP, to distribute or decentralise a high-performance centralised strategy. The concrete method that is used in the remainder is the following:

1. Choosing an evaluation criterion, e.g. the Iav or WI,
2. Choosing a model strategy, e.g. HPCC,
3. Generating data from executions of this model strategy in simulation,
4. Selecting statistical models to test, e.g. several ANN architectures,
5. Selecting one or several generic MAP configurations which stands for test scenarios used to select the best statistical model among those chosen in the previous step, e.g. {A, 15} is a general MAP configuration: A is a regular graph, and 15 agents for 50 nodes is a reasonable multiagent patrolling,
6. Training models for the test scenario, for example that resulting from steps 2 and 5: {HPCC, A, 15}, on the generated data,

7. Creating the new decentralised strategies based on these statistical model; to do this, they are in fact embedded in agents,
8. Evaluating the new machine learning strategies in simulation in the test configuration, e.g. {A, 15},
9. Choosing the statistical model with the best performance in simulation,
10. Generalising to all needed configurations.

Other annex steps could be inserted to refine this method with respect to the use aimed at.

3.4 Implementation

In this section, the main tools developed to carry out this work are presented.

3.4.1 PyTrol

PyTrol[2] is a discrete-time simulator dedicated to MAP, designed as a Python framework with the aim of performing MAP simulation. Originally, this simulator has been developed for the purpose of generating merely and rapidly a significant amount of MAP data, for the needs of this work. It has thereafter been evolved to become an easy-to-extend framework totally dedicated to MAP: the user can write custom building blocks to express and experiment new ideas for MAP research; they can also develop state-of-the-art models. It is distributed over multiple threads, and can be used sequentially or parallelly. Its temporal model is discrete and each time step, or period, is not finished as long as all the agents have not acted.

To simulate a MAP mission, that is to say an instance of a given MAP scenario {Π, G, N_a}, a JSON file containing the setting of the mission to simulate shall be provided to the simulator. More precisely, the JSON file contains the description of the graph G, the society of agents, as well as their initial positions on the graph. The duration of the mission $T \in \mathbb{N}^*$ ought also to be set. For each mission, simulation traces are recorded in JSON log files, in which at each time step the position and the individual idlenesses of each agent, as well as the true idlenesses of nodes are logged. These files are then processed to compute statistics, or even to be used as data for learning, for example.

[2] https://github.com/mothguib/pytrol

In this framework, edges are discretised. As stated in **Subsection 2.1.4**, they are sampled over time, that is to say divided into units that agents travel in one period.

In its current version, PyTrol relies on 5 variables:

- the *position of agents* in the graph,
- the *completed actions*: a boolean variable which indicates whether all agents have completed their action,
- the *communication step*: a boolean variable which indicates whether the communication step has begun, outside this step agents cannot communicate,
- the *decision step*: a boolean variable which indicates whether the decision step has begun, outside this step agents cannot decide,
- the *interaction mode*: a boolean variable which indicates whether the agents can interact, i.e. whether the interaction scheme can be used.

3.4.1.1 Main components

PyTrol comes in the form of a python package called `pytrol`, which is itself decomposed into three main subpackages, as follows:

- `control`: represents the controller, namely the component of PyTrol which executes agents and controls all operations necessary to play the simulation out,
- `model`: represents the data model of MAP, namely all concepts and objects which determine the structure of MAP necessary to simulate a MAP execution,
- `util`: utilities, that is to say annex tools, procedures and algorithms being generic enough to be used in other projects independent from PyTrol.

3.4.1.2 `pytrol.control.Communicating`

A key structure in PyTrol is the `pytrol.control.Communicating.Communicating` class. This class, which extends `threading.Thread`, provides all of the abstract methods necessary to communicate, and thereby allows creating independent threads able to communicate. Any `Communicating` object will be referred to as *communicating*. Any possible way of communication is practicable, letting the user provide an object whose the class extends the `utils.net.Connection.Connection` abstract class. Thus, the

type of needed connection is left to the discretion of the user. By default, the concrete class `utils.net.SimulatedConnection.SimulatedConnection` is used, enabling *communicatings* to communicate by reference, i.e. memory address.

3.4.1.3 `pytrol.control.agent`

Before continuing, it is worth recalling that any agent strategy is, in fact, an algorithm. Also, according to what has been set out in **Section 3.2**, in the context of this work a *multiagent strategy* is merely defined as a set of N_a single-agent strategies.

`pytrol.control.agent` contains agent strategy implementations. Any new implemented strategy shall extend the `Agent` class located in the `pytrol.control.agent.Agent` module, and be added in this package. `Agent` is an abstract class defining a template for any agent strategy. This template defines, in fact, the basic procedure that any agent must follow. This basic procedure, qualified as *main procedure of agent*, represents the life cycle of agents and consists of:

- `Agent.prepare`: any preprocessing, if necessary, the agent needs to carry out to prepare the impending main procedure,
- `Agent.perceive`: the agent perceives the position of the other ones, if required by its strategy; in the strategies studied in this dissertation only the position of the agent itself is perceived, although other types of perception are left to the discretion of the user,
- `Agent.communicate`: the agent communicates with other ones, if required by its strategy;
- `Agent.analyse`: the agent checks and processes messages he has received,
- `Agent.decide`: the agent decides; this method constitutes the core of the strategy, given that any strategy is a decision-making procedure in the context of MAP,
- `Agent.act`: the agent acts according to the decision made in the previous method.

Each agent, namely each object instantiating the `Agent` class, is a *communicating* and therefore a thread; concretely the `Agent` class extends the `Communicating` class. Any new strategy to add in PyTrol shall be implemented from the above methods, then added to the `pytrol.control.agent` package, and finally referenced in `pytrol.model.AgentTypes`. A set of strategies are already implemented in PyTrol:

- CR in `pytrol.control.agent.CR`, implemented according to **Algorithm 1**
- HPCC in `pytrol.control.agent.HPCC`, implemented according to **Algorithm 3** for the coordinated agents and **Algorithm 2** for the coordinator,
- HCC in `pytrol.control.agent.HCC`,
- strategies based on machine learning that extend the abstract class `pytrol.control.agent.MAPTrainerModelAgent`.

Algorithm 1: Conscientious Reactive (CR)

input : t = 0, a an agent, T the number of periods

while $t < T$ **do**
  if not $v^a(t) = v^a(t - \tau_0)$ **then** /* τ_0 being the transit time between the previous node and the next node to visit */
    $\text{ng} = Ng(v^a(t))$;
    $v^a(t + \tau_1) = \arg\max(i^a(t)[ng])$; /* τ_1 being the transit time between the current node and the next node to visit, and thereby $v^a(t + \tau_1)$ is the next node to visit */
  end
end

In the implementation of HPCC studied in this dissertation and coded in PyTrol, agents request a new goal node each time they arrive at a node; for each agent the Heuristic and Pathfinder algorithms are therefore executed by the coordinator each time they arrive at a node. With regard to the implementation of HCC, the Warshall's algorithm is executed at the simulation's startup to compute once and for all the shortest distances and paths between the nodes.

3.4.1.4 `pytrol.control.Ananke`

The `pytrol.control.Ananke.Ananke`[3] class is the core of PyTrol, i.e. the structure which concretely handles the simulation running. It is also a *communicating*.

The life cycle of `Ananke` starts with the mission's initialisation which takes place in `Ananke.__init__`, where it loads the graph, all information relative to the current mission, as well as the agents.

[3] *Ananke* is an ancient Greek goddess who was the personification of inevitability, compulsion and necessity, and in other terms, of what must happen.

Algorithm 2: Heuristic Pathfinder Cognitive Coordinator (HPCCor)

input : t = 0, a an agent, T the number of periods, ms the stack of received messages, asg_nodes the list of the nodes assigned to agents

```
1  while t < T do
2      m = pop(messages)
3      while m != NIL do
4          a, v^a(t) = parse(m);
5          goal_node = heuristic(i(t), v^a(t)); /* The Heuristic method is first
            applied to select the next (remote) goal node */
6          v^a(t + τ1) = pathfinder(goal_node, i(t), v^a(t), asg_nodes)[0];     /* The
            Pathfinder method is applied to select the best path to go
            there, and finally the next node to visit, of index 0 in the
            list, is chosen */
7          asg_nodes[a] = v^a(t + τ1); /* The node v^a(t + τ1) selected above to
            be assigned to agent a is recorded to avoid assigning it to
            another agent */
8          send(coordinator, message("next node", v^a(t + τ1)), a);
9      end
10 end
```

Algorithm 3: Heuristic Pathfinder Cognitive Coordinated (HPCC)

input : t = 0, a an agent, T the number of periods

```
1  while t < T do
2      if not v^a(t) = v^a(t − τ0) then
3          send(a, "next node request", coordinator);
4      end
5      m = receive(a);
6      if m is a "next node" message then
7          v^a(t + τ1) = process(m);
8      end
9  end
```

Then, in `Ananke.run` the main simulation loop over the time steps is executed. There is as many iterations in this loop as the duration T set for the current run. This loop stands for the running of simulation: at each period, the strategy of agents simulated herein is deployed. More precisely, at each iteration Ananke executes the main procedure of the strategy by calling, for every agent, the methods described above which constitutes their life cycle.

3.4.1.5 `pytrol.control.Archivist`

The `pytrol.control.Archivist.Archivist` class gives rise to a communicating object which logs the running of the simulation, that is as stated above, the positions, individual idlenesses of each agent, and true idlenesses. Complementary MAP elements or events to log might be added.

3.4.2 MAPTrainer

MAPTrainer[4] is a Python framework based on the PyTorch library[5] dedicated to machine learning research in the context of MAP. As PyTorch, it uses GPUs and CPUs. MAPTrainer is a native Python package by design, called `maptrainer`. Its functionalities are built as Python classes, which enable the integration of its code with Python packages and modules. Relying on PyTorch, it enables GPU-accelerated training, through CUDA, for machine learning, and especially neural network applications. The objective of this framework is to train any machine learning model, built with PyTorch, for any MAP scenario. Parameters defining the MAP scenario are provided to the programme.

3.4.2.1 `maptrainer.model`

`maptrainer.model` allows implementing any machine learning model in the same way as PyTorch. Any new machine learning model which is intended to be experimented shall be added in this package. Then, the model of this class will be loaded by `maptrainer.model.modelhandler`, which will check, if specified by the user, whether a previous trained version of this model for the current MAP scenario exists. Otherwise, a new model is created for the current scenario.

[4] https://github.com/mothguib/maptrainer
[5] pytorch.org.

3.4.2.2 `maptrainer.data`

This package adds support for loading MAP data necessary to train a model. The data of a MAP scenario are loaded by means of a *data loader*, which is merely an object of `maptrainer.data.MAPDataLoader.MAPDataLoader` type. `MAPDataLoader` is an abstract class defining a template for any data loader. A new data loader shall extend this class and implements its methods. Over the training stage, the data loader will then return data divided in validation and training data, and for each type of data, into input and output data. In execution, the data will then be returned fold after fold if a k-fold cross-validation is required. As part of this work, two concrete data loaders are developed:

- `BPDataLoader`: loads *binary sequences*,
- `IPDataLoader`: loads *on-vertex idleness sequences*.

3.4.2.3 `maptrainer.train`

`maptrainer.train` is used to automatically train a model. A complete training is run with the `maptrainer.train.run_epochs` method, which runs a training for the loaded model, the criterion to optimise, the data loader, the learning rate, and the number of epochs, among others.

3.4.2.4 `maptrainer.valid`

`maptrainer.valid` is used to validate the model with respect to the criterion to optimise, the data loader, the learning rate, and the number of epochs, among others.

3.4.3 MAPTor

MAPTor[6], for ***Map Editor***, is a convenient Python toolkit for preparing and processing MAP experiments. It can be regarded as an annex tool for PyTrol and MAPTrainer, although it can be used in any research work concerning MAP which uses the JSON format. By doing so, it is an editor and a processor for MAP that comes in the form of a python package called `maptor`. It offers the following functionalities:

- converting positions of agents traced in JSON logs to binary position, i.e. one-hot vectors representing the travelled nodes,

[6] https://github.com/mothguib/maptor

- processing idlenesses traced in JSON logs to convert them to on-vertex idlenesses,
- computing statistics from JSON logs,
- loading the graph to patrol; considering that `pytrol` needs to load the graph for each mission, this requires that `maptor` be a dependence of `pytrol`,
- automatically situating the graph in a frame, if it is not situated, using the *NetworkX*[7] library,
- plotting statistics in the form of curves or diagram bars.

3.5 Model strategy and databases

In the following of this work, two high-performance centralised strategies are distributed; databases constituted from this two strategies are now presented in this section.

3.5.1 First database: HCC 0.2

First, the HCC strategy with $r_H = 0.2$, abbreviated as HCC 0.2, has been chosen as model strategy without any consideration for the performance. In this case step 2 of the methodology outlined above was skipped. Its simulation traces gave rise to the first database used to train the statistical models studied in this dissertation. To constitute this preliminary database, 100 runs for each MAP scenario {HCC 0.2, G, N_a}, such as $G \in \{A, Islands, Grid\}$ and $N_a \in \{5, 10, 15, 25\}$, were executed. Then they were processed to make up the corresponding binary paths and on-vertex idleness sequences.

HPCC variant	**Iav**	**MI**	**QMI**	**WI**
HPCC 0.1	940	**147**	489	31188
HPCC 0.2	546	149	397	30025
HCC 0.2	263	232	341	13687
HPCC 0.5	**233**	184	**293**	**1216**
HPCC 0.8	319	194	353	4923
HPCC 0.9	329	199	365	3378

Table 3.2 Normalised Iav, MI, QMI and WI for several variants of HPCC averaged over the A, Islands, and Grid topologies, and over 5, 10, 15 and 25 agents.

[7] https://networkx.github.io/. Retrieved 2019-09-30.

3.5.2 Second database: HPCC 0.5

The HPCC strategy being a centralised high-performance and one of the best online strategies, it has been selected to be the model strategy. However, steps 1 and 2 of the methodology could be intertwined owing to the fact that in real life the choice of a strategy is a multicriteria decision process.

Thus, with the aim of constituting a larger database, different variants of HPCC were assessed. To determine the best one, six of it were evaluated according to the Iav, MI, QMI and WI, on the A, Islands and Grid topologies. For each graph they were assessed in simulation for r_H and r_P being both equal to 0.1, 0.2, 0.5, 0.8 and, 0.9. **Appendix B.1** shows the detail of the performance of these HPCC variants on the selected criteria and **Table 3.2** summarises the main features of them.

According to these results, HPCC 0.5, the variant of HPCC with $r_H = 0.5$ and $r_P = 0.5$, is the best variant on all criteria, except on the MI. Particularly, except for the two configurations {Islands, 10} and {A, 10} on the QMI, HPCC 0.5 is doubtless the most robust solution, i.e. the variant which has the best WI and QMI. Interestingly, in terms of performance HCC 0.2 is the best strategy after HPCC 0.5 on the QMI and Iav. HPCC 0.5 is selected as being the second model strategy in this dissertation, and it gives rise to a second database. For this second model strategy, it is decided to generate 10000 sequences for each MAP scenario, leading to:

- 2000 runs for a scenario with 5 agents,
- 1000 runs for a scenario with 10 agents,
- 667 runs for a scenario with 15 agents,
- 400 runs for a scenario with 25 agents.

Thus, for HPCC 0.5 data, the number of data no longer hinges on the number of agents N_a, as for the HCC 0.2 data, but it is identical, whatever the scenario. Finally, this database includes data for the six topologies addressed in the literature in **Subsection 2.1.1**.

3.6 Conclusion

All together, the proposed model of temporal MAP, the methodology established to address our problematic, namely decentralising and distributing a centralised multiagent strategy, as well as the tools developed to study this problematic, allows implementing

this methodology. Two databases are derived from the first steps of this implementation and might be used to conduct experiments for strategies based on learning of nodes to visit, or of true idlenesses. Details regarding those strategies and experiments are exposed in the next chapters.

Chapter 4

Path-Maker: a decentralised strategy based on node prediction

Learning temporal sequences is well adapted to the MAP problem insofar as it constitutes a temporal decision problem. In this chapter a first application of ANN to MAP is presented: a new procedure based on machine learning algorithms is proposed to decentralise a centralised decision-making process. This new procedure rests on temporal node prediction which gives rise, as implementation of this procedure, to a new generic strategy in the context of MAP. The node predictor is first trained on simulation traces generated by a fully-informed, coordinated and communicating strategy, namely the model strategy; the model strategy used in this chapter is HCC 0.2. The statistical models, by doing so, are trained on the HCC 0.2 database. Then, each agent of this new strategy uses its predictor to select the next node to visit with respect to its current node. Finally, different variants of this new strategy are evaluated in simulation and compared with other strategies.

Section 4.1 outlines this new procedure which enables decentralising a centralised decision-making process using node prediction. Then, **Section 4.2** describes the way the predictors have been trained. Finally, in **Section 4.3** experiments with the new strategy, as well as its results are described and analysed.

4.1 Path-Maker

One of the objectives of this chapter is to implement concretely the methodology outlined in **Chapter 3** into MAP strategies. More precisely the statistical model is used here as node predictor. Particularly, RNNs, and more precisely LSTM networks are used as node predictors.

4.1.1 Path-Maker

Path-Maker (PM) is a machine-learning-based generic strategy: it defines a general strategy whose the decision-making process is carried out by means of a statistical model used as node predictor. This predictor outputs the next node according to the current one. More precisely, the statistical model is used to make prediction with regard to the next node to visit. This strategy can be thought of as a reactive strategy using an artefact for guidance through the area to patrol. In fact, such a trained model takes implicitly into account both the idleness of nodes and the agents' positions. In this context, any temporal series corresponding to a sequence of successive nodes visited by an agent is called *path*. As part of MAP, as set out in **Chapter 3** such a strategy to learn is termed model strategy.

For a given scenario, the model temporally learns $\forall k \in \mathbb{N}^*, v(k+1)$ the next node to visit according to the previous ones $v(k), v(k-1), \ \dots \ , v(1)$, from data generated by a model strategy, this for all paths: each path being the path of an agent on the graph, it is fed into the network node after node. Let f be an ideal decision procedure of the model strategy, i.e. a strategy depending only upon the current node and true idlenesses which would have generated the data of such a model strategy. Then, it follows that the statistical model, noted $\tilde{f}$, for the current scenario approximating f can be defined as follows:

Let:

- $\forall t \in \mathbb{N}^*$, $i(t) = (i_{v_1}(t), \ \dots, \ i_{v_N}(t)\}$ be the vector of true idleness of the graph G at time t,
- $\forall k \in \mathbb{N}^*$, $v^a(k) \in V$ be the k^{th} node visited by agent a,
- $\forall a \in A$, $t^a : V \times \mathbb{N}^* \to \mathbb{N}^*$ be the function mapping to the k^{th} node visited by agent a the corresponding time t.

With such an ideal decision procedure f, the next node to visit would be selected such as:

$$\forall k \in \mathbb{N}^*, v^a(k+1) = f(\ v^a(k), \ i(t^a(v^a(k), k))\ ) \tag{4.1.1}$$

is the next node to visit, whereas $\tilde{f}$ is a node-predictor-based decision procedure approximating f, i.e. $\tilde{f} \approx f$. Theoretically, it then follows:

$$\forall k \in \mathbb{N}^*, \tilde{v}^a(k+1) = \tilde{f}(v^a(k), ..., v^a(1)) \tag{4.1.2}$$

and

$$\forall k \in \mathbb{N}^*, \tilde{f}(v^a(k), ..., v^a(1)) \approx f(\ v^a(k),\ i(t^a(v^a(k), k))\ ) \tag{4.1.3}$$

This equation pertains to the feature laid down previously: each output depends upon the previous outputs. At this stage, it is worth noting that a constraint of a path is that any node ought to have one of its neighbours as subsequent node.

Finally, upon the training stage's completion each agent is endowed with the same parametrised node predictor. It then uses it to find its path through the graph throughout the mission. This strategy is meant to replicate, or approach, the model strategy's behaviour in a decentralised way, as shown in **Figure 4.1**. In the particular case of Path-Maker, the vector of individual idleness being computed with respect to the successive visits of the agent, only its visited nodes are here considered.

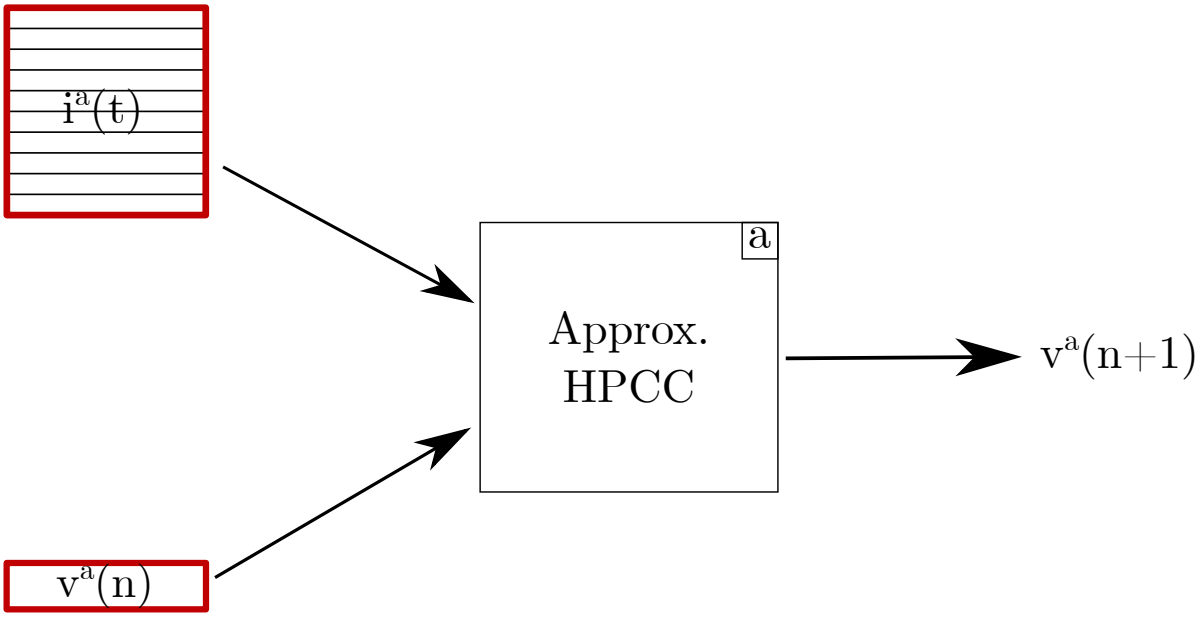

Fig. 4.1 Decision procedure of the replicated and decentralised HPCC strategy, with $n \in \mathbb{N}$ standing for the nth decision step, t the corresponding time, and $v^a(n)$ and $i^a(t)$ the node visited by agent a as well as its vector of individual ideleness, resp., at the nth decision step.

4.1.2 RNN-Path-Maker: an implementation of Path-Maker

As stated in the introduction and detailed in **Appendix A.3.3**, RNNs constitute a type of ANNs well suited to learning time series. They can therefore be used as node predictors to implement the Path-Maker strategy.

Node predictor. Let N denote the number of nodes in the graph. In order to feed the model with the most meaningful information regarding the nodes, each one is encoded as a N-dimensional one-hot vector, as described in **Section 3.2**. For example, $\forall v \in V$, all the coordinates of its corresponding one-hot vector are set to 0, except the v^{th} coordinate which is set to 1. From now on the set of one-hot vectors corresponding to the set of nodes V is denoted X^N, with $card(X^N) = N$ and $X^N \subset \{0,1\}^N$. The input of the network must thereupon be an N-dimensional vector of X^N, while its output can be specified as a probability distribution on V. To ensure that the output of the RNN is a probability distribution, i.e. the values are positive and their sum equal to 1, a softmax layer is added as output layer of the network. Finally, the next node to visit is chosen using this distribution.

In the following, only RNNs with the same width for all layers are considered.

Let (L, H) denote the *profile* of an RNN so that L and H stand for the number of layers and the number of hidden units (or memory cells) per layer, respectively. Formally, by defining $\beta : V \to X^N$ as being the function mapping all the nodes of V to their one-hot representations, the proposed architecture can then be described with:

$$\begin{aligned} &\forall t \in \mathbb{N}^*,\ x_t^1 = \beta(v^a(t)) \\ &\forall t \in \mathbb{N}^*,\ \text{if } L > 1, \forall l \in [|1, ..., L-1|]\ x_t^{l+1} = h_t^l \\ &\forall t \in \mathbb{N}^*,\ m(x_t^1; \boldsymbol{\theta}) = softmax(W^{sm} \cdot h_t^L + b^{sm}) \end{aligned} \tag{4.1.4}$$

where:

- $\forall l \in [|1, L|],\ \forall t \in \mathbb{N}^*$, x_t^l is the input of the layer l,
- h_t^l is the output of the layer l,
- $\forall a \in A$, $v^a(t)$ is the node visited by agent a at time t,
- m is the RNN,

- $H = dim(h_t^L)$,
- W^{sm} and b^{sm} are the $N \times H$-matrix of weights and the N-dimensional vector of biases, respectively, of the softmax layer,
- $\boldsymbol{\theta}$ represents the parameters of the network, including not only W^{sm} and b^{sm}, but also the parameters of the layers from 1 to $L-1$, with $\dim(\boldsymbol{\theta}) = 4(2L\ -\ 1)H^2 + (4L\ +\ 5Card(V))H$.

Finally, the output of M is a probability distribution over the nodes of the graph. From this generated distribution, any decision-making procedure can be used to choose the next node to visit.

In the following, concretely the LSTM architecture, described in **Appendix A.3.3.1**, will be used as concrete RNN architecture giving rise to the *LSTM-Path-Maker* strategy.

4.1.3 Deterministic-Path-Maker

A first practicable used of the probability distribution generated by model m is to select deterministically the next node to visit, such as that with the highest probability is selected, as follows:

Let:

- $m : \{0,1\}^N \to [0,1]^N$ the node predictor used by an agent $a \in A$,
- $Ng : [0,1]^N \times V \to [0,1]^N$, the function setting to zero the coordinates not corresponding to the neighbours of a given node.

Then,

$$\begin{aligned} &\forall t \in \mathbb{N}^*, \forall v_t \in X^N, \\ &\quad v^a(t+1) = \arg\max(\ p_1, \ldots, p_N : (p_1, \ldots, p_N) = Ng(m(\ \beta(v_t); \boldsymbol{\theta}\ ), v_t)\ ) \end{aligned} \tag{4.1.5}$$

It then follows:

$$\begin{aligned} &\forall t \in \mathbb{N}^* :\ v^a(t) \in V, \\ &\quad \tilde{f}(v_t, ..., v_1) = \arg\max(\ p_1, \ldots, p_N : (p_1, \ldots, p_N) = Ng(m(\ \beta(v^a(t)); \boldsymbol{\theta}\ ), v_t)\ ) \end{aligned} \tag{4.1.6}$$

with $\tilde{f}$ depending upon $v(1), \ldots, v(t)$ considering that their relevant features are stored in the memory of $m()$.

On a side note, it is interesting to note that $\tilde{f}$ belongs to the family of distributed agent-independent deterministic policies, namely policies in the form of $d_a(t) = \pi(s_a(t))$.

4.1.4 Random-Path-Maker

First experiments showed that selecting the next node to visit as being that with the highest probability in the neighbourhood tends to lead agents to converge indefinitely into a small set of nodes. As a consequence of this, nodes may not be visited until the end of the execution.

As indicated in **Chapter 2**, random variation may increase the robustness and adaptibility of a controller. Therefore, the decision procedure has been slightly improved: henceforth, in order to better distribute the visits of agents over the nodes, the next vertex is chosen randomly from a sample according to the probability distribution output by the model. More specifically, the probability distribution is first normalised over the probabilities of the current vertex's neighbours using Bayes' theorem, that is the probabilities of the non-neighbour nodes are set to zero and all the probabilities of the neighbours are multiplied by a common factor in order to have their sum equal to 1. Then, a random drawing is performed. This new procedure enables therefore making the decision process random when choosing the next node in the neighbourhood. This leads to increase the robustness of the system, and thereby to avoid agents to visit only a restricted group of nodes.

The new resulting strategy is called *Random-Path-Maker* and belongs to the family of agent-independent non-deterministic policies. When the LSTM architecture is concretely used, the strategy is called *Random-LSTM-Path-Maker* (RLPM).

4.2 Training procedure

According to **Chapter 3**, the training of LSTM networks is performed from logged paths of any high-performance strategy f. In fact, what is intented in the present case is to decentralise a high-performance decision process on the basis of a node predictor.

4.2.1 Pretraining

For a given graph G and all scenarios involving $\{\Pi, G, N_a\}$ with possible variations of Π and N_a, the LSTM network is first pretrained with the aim of learning the graph representing the area. This stage aims at ensuring the network to capture as far as possible the structure of the topology. To that end, in this stage each edge is provided to the network in the form of 2-length series of one-hot vectors.

4.2.2 Main training

Thereafter, the network is trained over all of the paths retrieved from the executions of any scenario $\{\Pi, G, N_a\}$, so that it learns to output with the highest probability the next node to visit in the path. The process described here can be thought of as performing sequence modelling where sequence is a path of nodes; here sequence modelling corresponds to *path generation*.

As aforementioned in 4.1.2, the network's output layer being a softmax layer, the output can be interpreted as a probability distribution. As indicated in **Appendix** A.1.2 cross-entropy is a germane criterion for learning probabilities. Thus, path generation aims at learning a probability distribution over paths by minimising the cross-entropy of a model given a set of M training sequences of length T_s:

$$\min_{\boldsymbol{\theta}} \; -\sum_{n=1}^{M}\sum_{t=1}^{T_s} \log \; p(v_n(t)|v_n(1),...,v_n(t-1);\boldsymbol{\theta}) \tag{4.2.1}$$

where:

- $\forall t \in [|1, T_s|],\ \forall n \in [|1, M|]$, $v_n(t)$ is the t^{th} node of the n^{th} sequence,
- $\boldsymbol{\theta}$ is the set of the model's parameters,
- p is the predicted probability for the current element $v_n(t)$ component of the observed sequence n, given by the $v_n(t)^{th}$ coordinate of the $m(v_n(t-1);\boldsymbol{\theta})$ vector.

4.3 Experiments and results

This section outlines the assessment of the new RLPM strategy introduced in this chapter, that is the experiments and evaluation. The conduct of experiments is first outlined, and the training settings is detailed. Finally, experiments in simulation of the RLPM

strategy are described, then the results are discussed.

4.3.1 Conduct of experiments

The conduct of experiments to study and evaluate the new RLPM strategy is now described. First the scenarios as well as the number of random initial positions for each scenario are defined. Then, statistical models are chosen then trained. Finally, the best statistical models, giving rise to variants of the new strategy, are experimented and evaluated. This conduct is detailed in the next paragraphs.

Scenarios. The RLPM strategy newly defined in this chapter has been assessed on the topologies A, Islands and Grid, shown on the **Fig. 2.1**, for 5, 10, 15 and 25 agents. Therefore, 12 scenarios were experimented.

Missions. For each scenario, 100 missions were executed, i.e. for the same topology, number of agents and strategy, 100 random starts were executed. Each mission lasted 3000 time steps.

Training of statistical models. Several LSTM architectures are selected then trained following the two training stages defined in **Section 4.2**. According to the training performance, best networks are chosen to be evaluated in simulation.

Simulation. The variants of RLPM resulting from the networks retained in the previous step are executed then evaluated.

4.3.2 Training settings

In this chapter the HCC 0.2 database is used to train the statistical models; HCC 0.2 is thereupon the model strategy for RLPM in what follows.

From the foregoing, for each MAP scenario several implementations of RLPM — each one giving rise to a variant — were trained from 100 executions of HCC 0.2, this for each scenario, using, as previously, the MAPTrainer framework. More rigorously, for each scenario seven LSTM architectures were trained with the following profiles of parameters: $(1, 1)$, $(2, 2)$, $(4, 10)$, $(1, 50)$, $(2, 50)$, $(3, 50)$, and $(50, 2)$. Number 50 has been chosen to have the same number of memory cells as the number of nodes in the studied topologies.

An end-to-end training, i.e. a non-truncated BackPropagation Through Time (BPTT) time was applied, and the LSTM networks are stateful. As described in **Section 4.2**, each architecture is trained in two steps: the pretraining step over which the network is pretrained from edges, then the main training step over which the network is trained from longer sequences of nodes. Both last 10000 steps.

The training settings are described in **Table 4.1**. Here, the objective function to optimise is the cross-entropy (CE), and the learning rate of the Gradient Descent (GD) algorithm is worth 0.1.

Settings	**All architectures**
Number of sequences	$100 * N_a$
Apportionment Training-Validation	$80\% - 20\%$
Model strategy	HCC 0.2
Batch size	$Full$
Learning rate	0.1
Algorithm	GD
Number of steps	10000
Criterion	Cross-Entropy

Table 4.1 Overview of the training settings

4.3.3 Training results

4.3.3.1 Pretraining results

For any machine learning model used in the context of MAP, the *neighbour accuracy* denotes, for a given topology, the proportion of nodes whose the output, i.e. the generated probability distribution, emphasises one of their neighbour. In other terms, a node is said *neighbour-accurate* for a model iff the output probability given by the model for that node has one of its neighbours having the maximum probability. A model generating a neighbour-accurate output for any node of a given topology will have a neighbour accuracy of 100%.

For the three studied topologies, the neighbour-accuracy is of 100% after the pretraining stage. This result is not a surprise considering that the pretraining database consisting of 2-length sequences representing the edges of the graph, it is consequently small. For example, on the A topology, the pretraining database has only 210 sequences, namely the number of edges of A. Also, with regards to the validation cost, as shown in

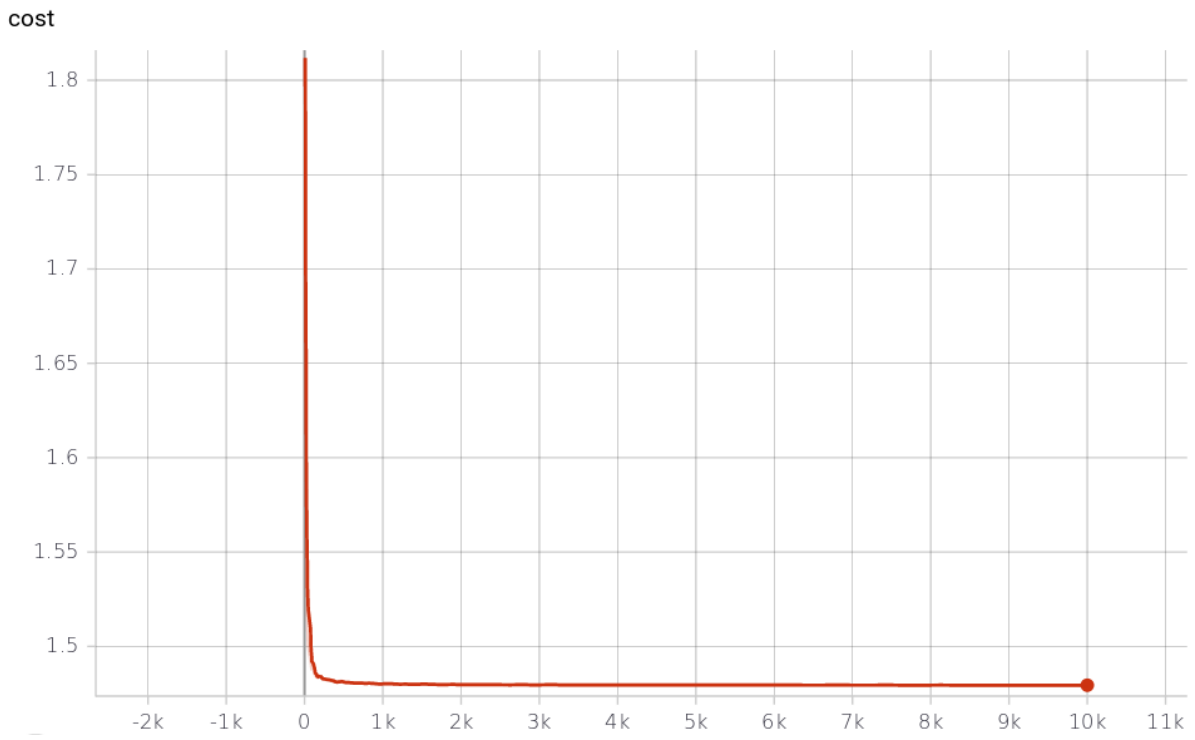

Fig. 4.2 Validation cost, in pretraining, of LSTM $(1, 50)$ on A.

Fig. 4.2 with the architecture $(1, 50)$ on A, in pretraining it decreases quickly, namely it has converged after approximately 150 iterations to 1.49

4.3.3.2 Main training results

Training performances are now compared with and without the pretraining stage.

In order to have a complete analysis of the training process, not only the cross-entropy is analysed but also the accuracy, i.e. the percentage of correct predictions of the next node, that is the node with the highest probability output by the model.

Fig. 4.3 and **Fig. 4.4** depict the validation cost and accuracy, respectively, during the main training stage on {A, 15}, without and with the pretraining stage carried out beforehand.

The validations costs on **Fig. 4.3** range from 3.934 without the pretraining stage and 4.34 with, to 1.10 for both. With regard to the accuracies on **Fig. 4.4** they both range from 2% to 55%. Also, according to these two plots the model converged after approximately 10000 steps. These curves show that adding a pretraining stage has practically no influences on the performance. Only a slight difference can be noted: a

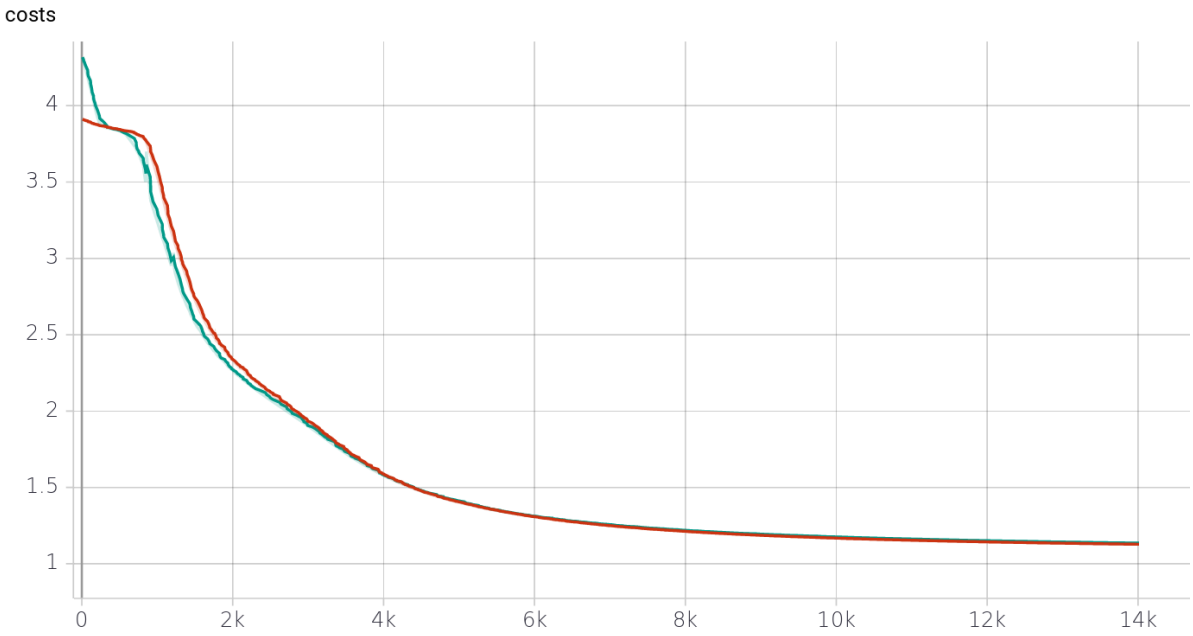

Fig. 4.3 Validation cost in main training without (red) and with (green) the pretraining stage of LSTM $(1, 50)$ on {A, 15}.

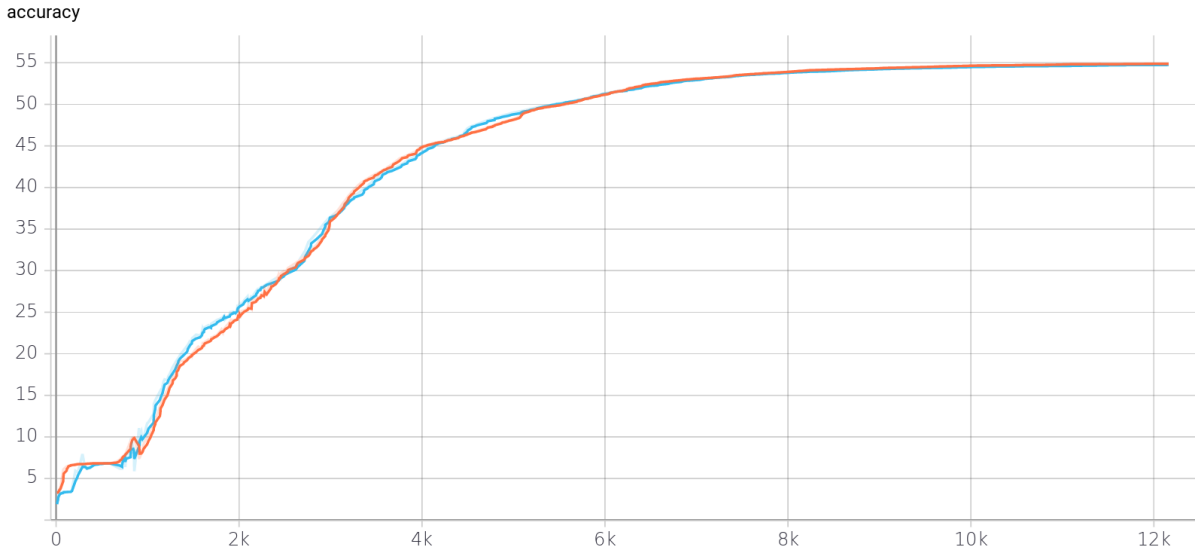

Fig. 4.4 Accuracy in main training without (orange) and with (blue) the pretraining stage of LSTM $(1, 50)$ on {A, 15}.

faster decrease, growth respectively, before epoch 1000 on the validation cost, accuracy respectively. After epoch 2000 the performance are identical on both evaluation metrics.

Fig. 4.5 depicts the initial and final values of the cost function, namely the cross-entropy, used to train the networks during the validation stage, that for each network, averaged over the A, Islands and Grid topologies and over the numbers of agents. Here, the initial cost corresponds to the validation cost after the first epoch. This figure shows that the best networks are $(2, 50)$, $(1, 50)$ and $(4, 10)$. Interestingly, the initial and final costs of architecture $(50, 2)$ are almost identical, whereas it corresponds to the worst final cost, among all of the architectures evaluated, here with a value of 3.87. This result tends to highlight that the parameters of $(50, 2)$ converged very quickly, that is in 1 epoch, due to the depth of the network. The number of parameters for $(1, 1)$, $(2, 2)$ and $(50, 2)$ are 258, 564 and 2484 respectively. It seems that these numbers are too low for a satisfactory approximation of the sequences. Conversely, $(3, 50)$ has 63100 parameters, which tends to consider this number as being too large to avoid overfitting, leading to a relatively bad performance in term of validation cost. Indeed, for 1 agent the size of training data is approximately 250000, namely not an order of magnitude higher than the number of parameters of the $(3, 50)$-network.

Considering the low performances of $(50, 2)$, $(1, 1)$ and $(2, 2)$ in training, with final costs of 3.87, 3.03, and 2.08 respectively, four LSTM architectures are tested in simulation: $(4, 10)$, $(1, 50)$, $(2, 50)$, $(3, 50)$. Then, each architecture gives rise to four variants of RLPM named $\forall(L, H) \in \{(4, 10), (1, 50), (2, 50), (3, 50)\}$, RLPM-$L$-$H$.

4.3.4 Simulation results

The RLPM variants studied here were tested and compared in simulation with CR, the reactive and decentralised representative, HCC 0.2, the cognitive and centralised representative from which the LSTM networks were trained. First the average idleness (Iav), then the MI and QMI are here retained to evaluate these strategies.

Fig. 4.6 depicts the normalised average idleness for the Islands topology. For the sake of clarity only the best variant of RLPM on this criterion, i.e. the variant with the lowest value, has been plotted for each scenario. Not surprisingly HCC 0.2 outperforms all of the other strategies in all cases and for all the number of agents studied here. On this graph the RLPM variants show low performance: they are always worse than the representatives CR and HCC 0.2 with values of 550 (RLPM-1-50) for 5 agents, 1078 (RLPM-4-10) for 10 agents, 1793 (RLPM-1-50) for 15 agents, and 1153 (RLPM-1-50) for 25 agents. Besides, a peak for 15 agents can be noticed. This result suggests, that from 5 to 15 agents the RLPM variants do not exploit the advantage afforded by additional

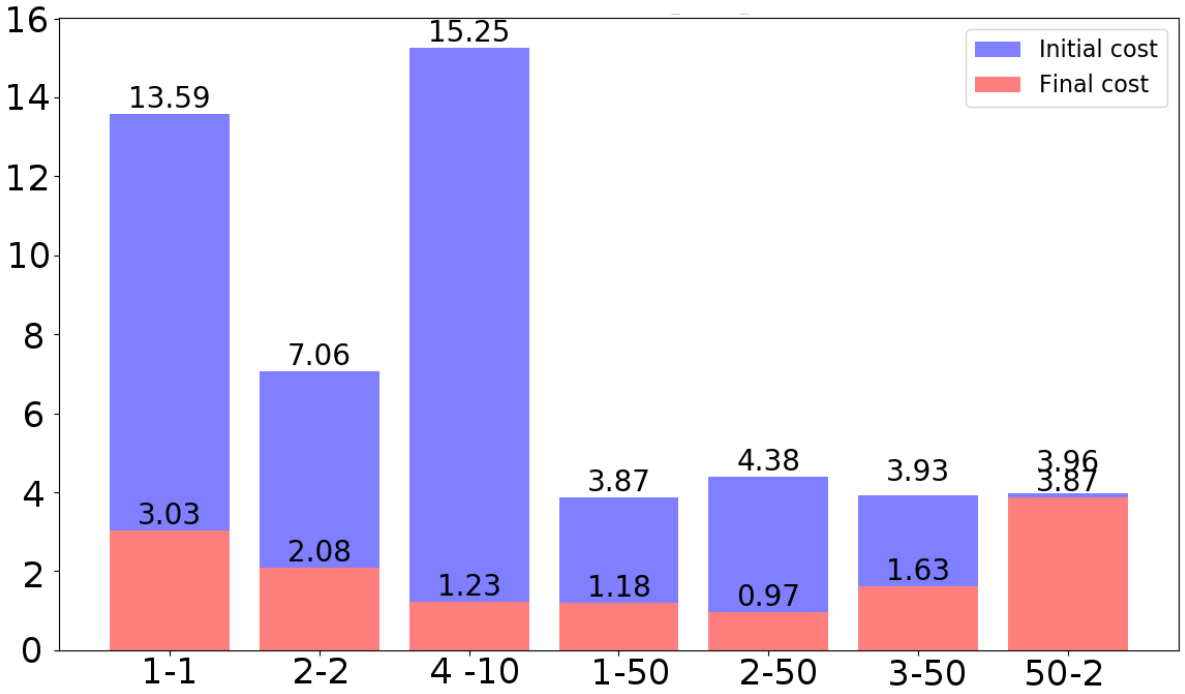

Fig. 4.5 Validation cost averaged over the A, Islands and Grid topologies and the numbers of agents for each LSTM architecture.

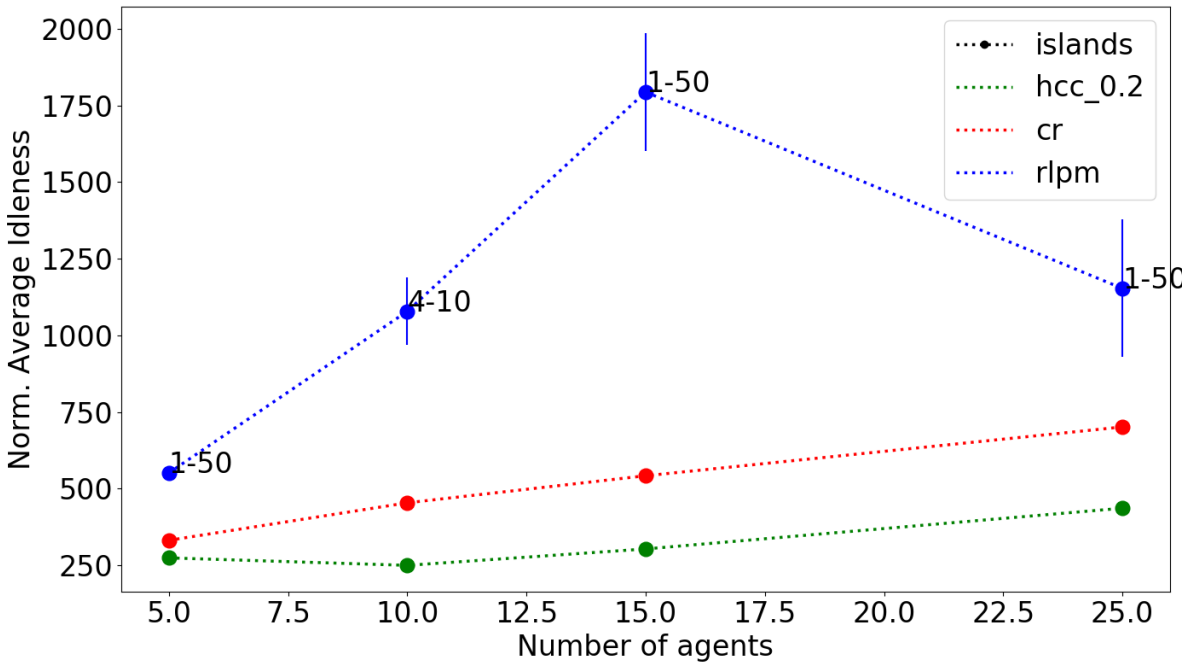

Fig. 4.6 Normalised average idleness, averaged over 100 runs, of the best variant of RLPM w.r.t. the size of agents on Islands.

agents. However, for 25 agents the performances are slightly better, which could be explained by considering that 25 agents visiting 50 nodes, the number of nodes of this topology, is equivalent to assign 2 places per agent.

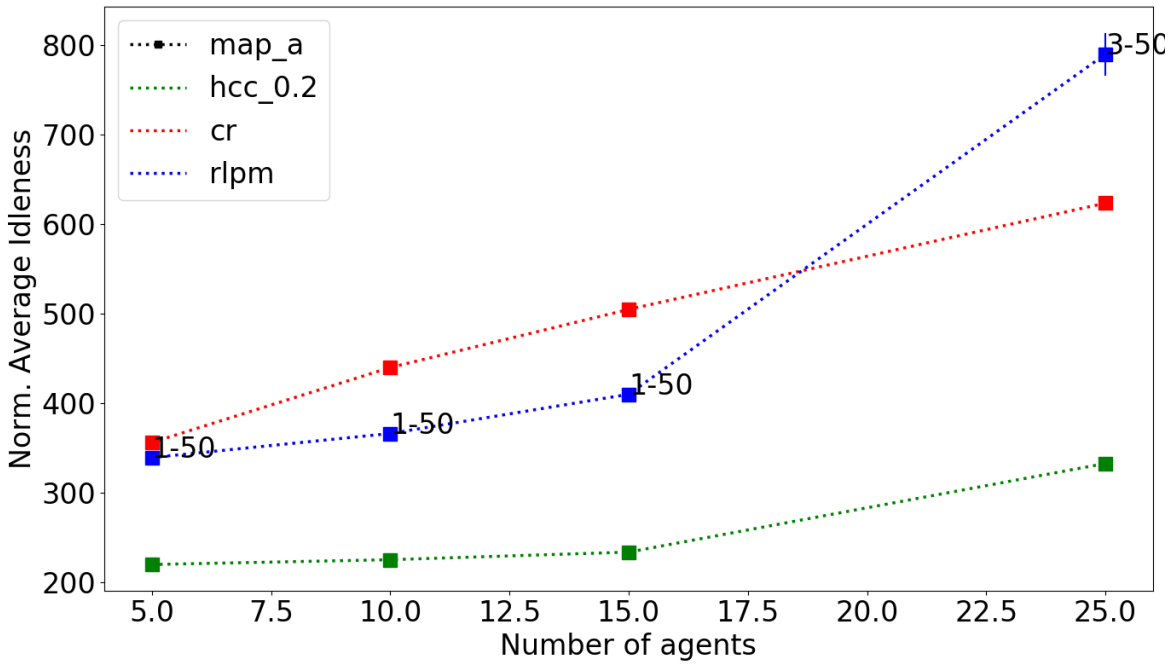

Fig. 4.7 Normalised average idleness, averaged over 100 runs, of the best variant of RLPM w.r.t. the size of agents on A.

The results on the Iav upon the A topology depicted on **Fig. 4.7** have shown that the best variant of RLPM, that is to say RLPM-1-50, is better than CR by 60 periods in average for 5, 10 and 15 agents, but worse by 166 periods for 25 agents with RLPM-3-50. Further investigation have tended to show that the RLPM strategy leads agents to visit very frequently only some nodes. Particularly, for 25 agents it has been figured out that the poor distribution of visits over the nodes is amplified by the inflow of additional agents not being taken into account: many agents visit the same set of nodes leading the normalised criterion to be penalised.

Finally, **Fig. 4.8** shows that, on the Iav, RLPM-1-50 remains the best variant of RLPM on Grid. It is better than CR for 10 and 15 agents by 43 periods in average, whereas for 5 agents RLPM-1-50 and CR are approximately equal. For 25 agents the best RLPM is worse than CR with a value of 716 periods against 558.

Globally, RLPM-1-50 is thus the best variant of RLPM on the average idleness, although for some scenarios it does not outperform CR, the decentralised representative.

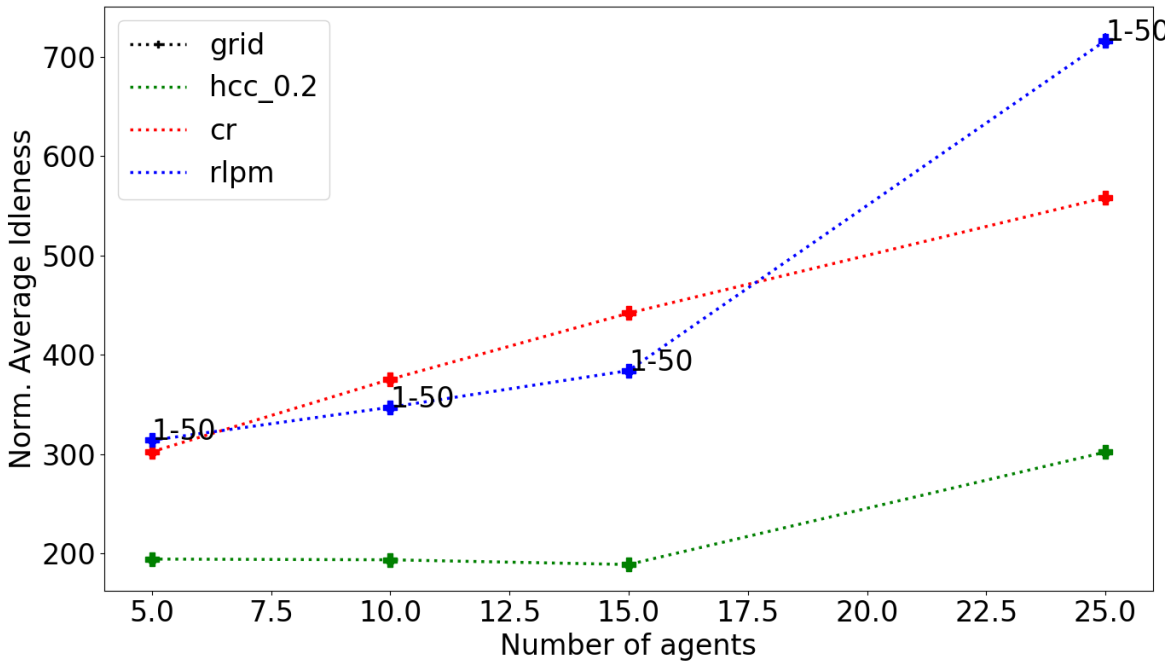

Fig. 4.8 Normalised average idleness, averaged over 100 runs, of the best variant of RLPM w.r.t. the size of agents on Grid.

As indicated by **Eq. 2.1.10**, the average idleness can be written as an expression of MI and QMI.

Fig. 4.9 depicts the normalised MI for the scenarios studied here. On this criterion the best RLPM outperforms significantly the reactive strategy CR while it is close to HPCC. Moreover, the evolution of the best variant of RLPM over MI with respect to the number of agents fits that of HCC 0.2 with an average difference, over all the numbers of agents and the graphs, of 15.

For the Islands and A topologies, the architecture $(2, 50)$ is the best on the normalised MI, whereas for Grid it is $(1, 50)$ for 5, 10 and 25 agents, but for 15 agents it is $(4, 10)$. Further analyses have shown that for 15 agents architecture $(1, 50)$ is only worse than $(4, 10)$ of 1 time step, they can then be regarded as equal. RLPM-1-50 is thus globally the best strategy on this graph. Also, for the same graph the average difference of performances over the numbers of agents between the best architectures enumerated above and the architecture $(2, 50)$ is only of 2 time steps. This leads to consider RLPM-2-50 as being globally the best RLPM strategy on the MI.

Lastly, **Fig. 4.10** shows the results for the normalised QMI. For Islands, the QMI of the best variants of RLPM is worse than HPCC and CR for all of the numbers of agents except for 25 agents where CR is worse by 28 periods. The best architectures are

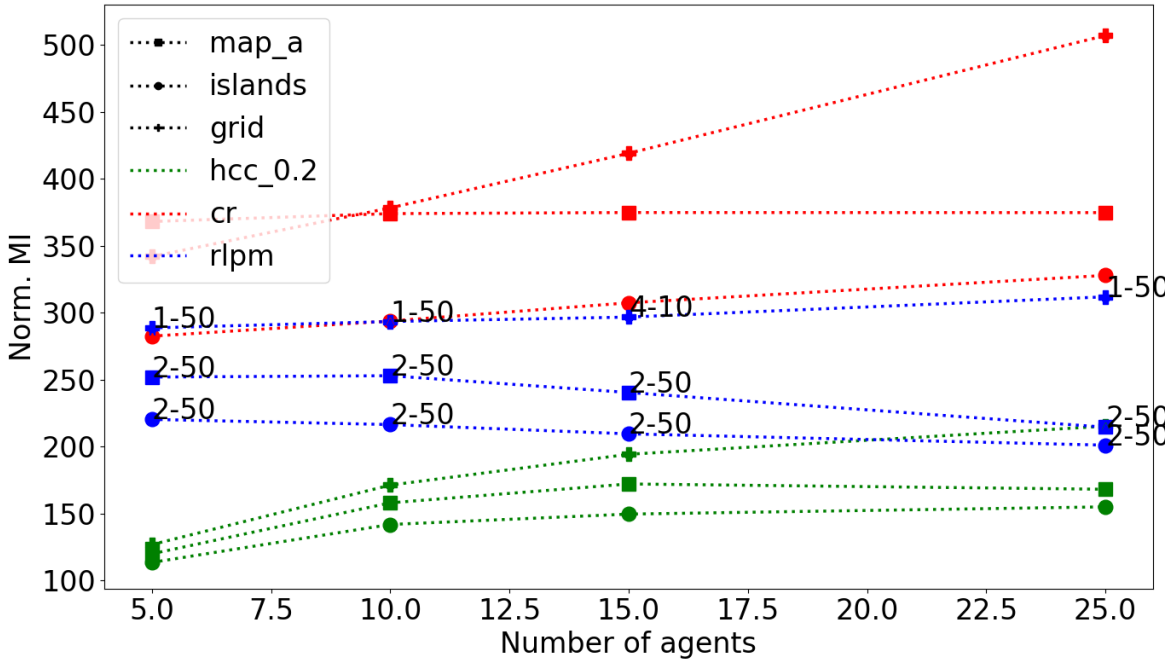

Fig. 4.9 Normalised MI of the best variant of RLPM averaged over 100 runs w.r.t. the number of agents on A, Islands and Grid.

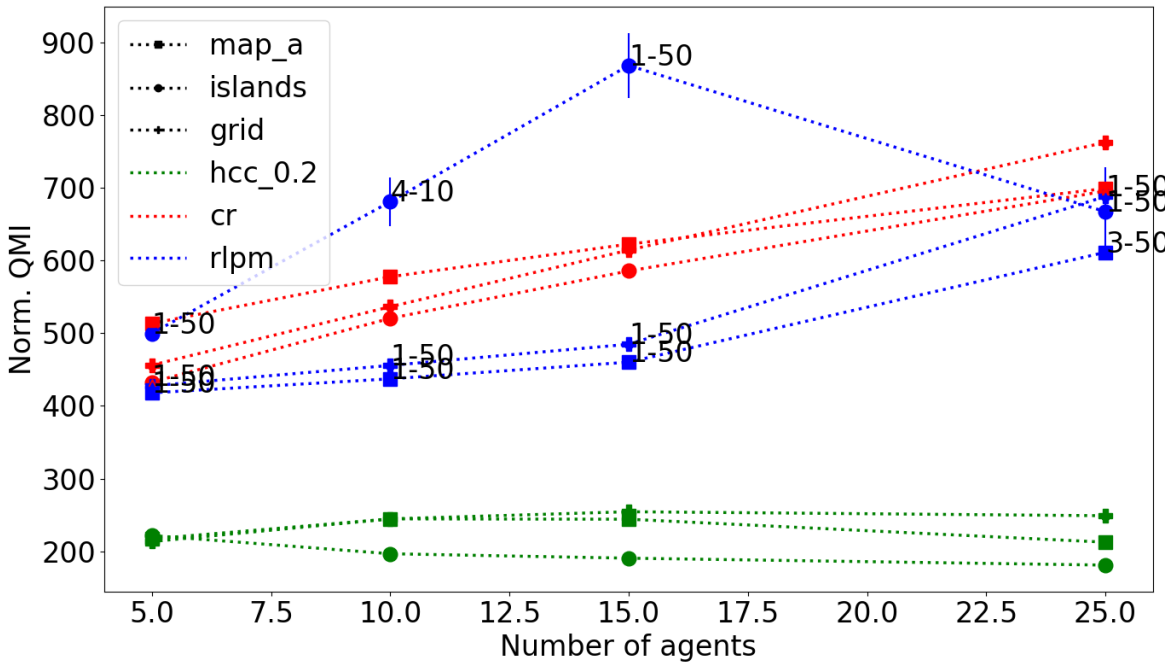

Fig. 4.10 Normalised QMI of the best variant of RLPM averaged over 100 runs w.r.t. the number of agents on A, Islands and Grid.

$(1, 50)$ for 5, 15 and 25 agents, and $(4, 10)$ for 10 agents on this criterion. In average, on Islands the best variants of RLPM are worse than HPCC by 256 periods with a significant difference of 533 for 15 agents. For the graph A they are better than CR but worse than HPCC, and except for 25 agents where RLPM-3-50 is the best RLPM strategy, RLPM-1-50 is always the best one. However, RLPM-2-50 has turned out to be the best strategy for QMI when being averaged over the number of agents with a value of 481 periods. Moreover, in average the best variants of RLPM are worse than HPCC by 152 periods. Lastly, for Grid the best variants of RLPM are worse than HPCC but better than CR. The best architecture is $(1, 50)$ for 5, 10, 15 and 25 agents. As for the A graph, in average RLPM-2-50 is the best strategy, although the best RLPM variants are worse than HPCC by 105 periods.

Finally, the $(2, 50)$ architecture tends to be the best RLPM strategy on the MI, except for Grid for which $(1, 50)$ is slightly better, whereas on the QMI $(1, 50)$ is irremediably and globally the best strategy, confirming thereby the first conclusions laid down on the Iav.

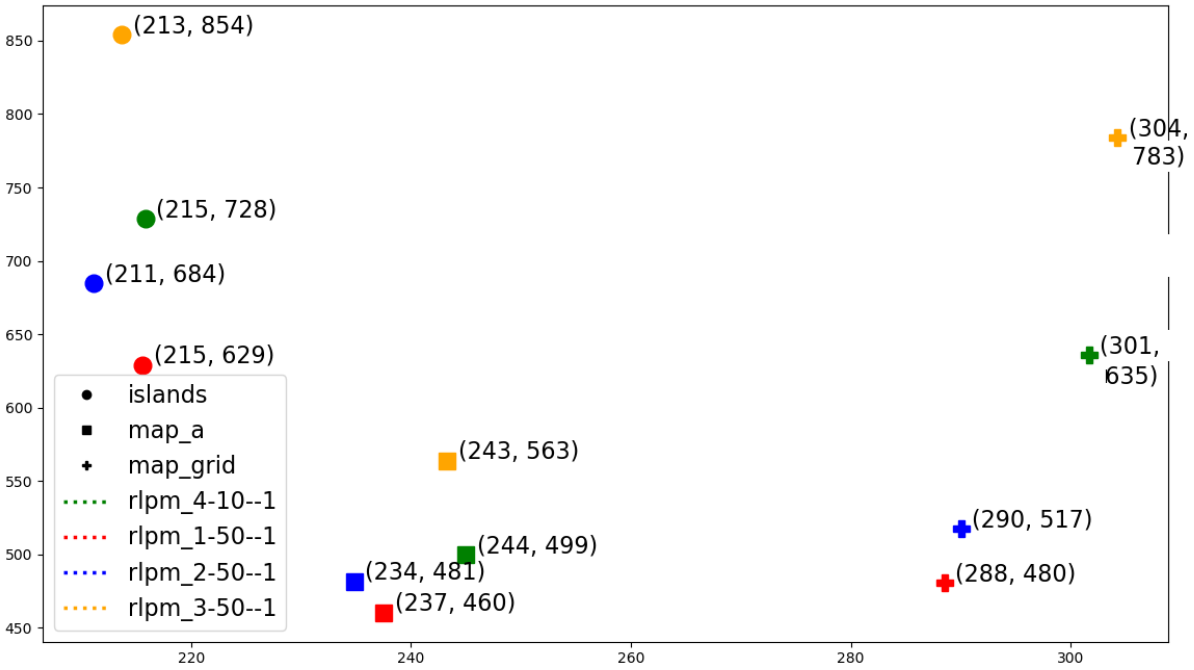

Fig. 4.11 Normalised MI and QMI of the best variant of RLPM averaged over 300 execution w.r.t. the number of agents on A, Islands and Grid.

Fig. 4.11 depicts the performance in the (MI, QMI) *criterion space* of the RLPM variants averaged over the numbers of agents. By doing so the variants of RLPM constitutes here the *decision space*.

This figure highlights the $(2, 50)$ and $(1, 50)$ architectures are two Pareto optimal solutions in the criterion space for Islands and A, where the former is the best on MI with a value of 211 on Islands and 234 on A, whereas the latter is the best on QMI with a value of 629 on Islands and 480 on A. On Grid the $(2, 50)$ is the sole Pareto optimal solution with a value of $(290, 517)$. This result tends to confirm the preliminary conclusions regarding $(2, 50)$ and $(1, 50)$, as being globally the best variants, where the former tends to be better on the MI whereas the latter is better on the QMI, although $(1, 50)$ can be considered as being the best architecture considering its performance on the Iav.

Interestingly, $(3, 50)$, the most complex architecture assessed here, gives rise to the worst strategy on the QMI criterion. As developed in **Subsection 2.1.3**, QMI, as quadratic mean, takes better into account the difference of time intervals between the nodes and thereby measures the tendency of nodes to be equitably visited throughout an execution. Based on that, it seems that RLPM-3-50 distributes agents over the nodes inefficiently. Similarly, the $(4, 10)$ architecture is a bad strategy on both the MI and QMI, that is to say in the (MI, QMI) criterion space. The performance according to the QMI of these strategies tends to show that they visit perpetually the same small set of nodes, leading to a poor distribution of visits over the nodes. It is likely that $(4, 10)$ presents also a too small number of parameters to learn the behaviour of HCC 0.2, whereas $(3, 50)$, conversely, could have a too large number of parameters to avoid overfitting.

4.4 Conclusion

In this chapter a new application of ANNs to MAP has been proposed and evaluated. More precisely, LSTM, a RNN architecture, has been used as node predictor to allow agents to navigate through the area to patrol without communicating; such a new strategy is thereupon decentralised according to the definition of a decentralised strategy in **3.1.2.4**. This new application has thus given rise to a new LSTM-based strategy for MAP consisting first, in training an LSTM network for a specific scenario on traces of a high-performance strategy, then, to embed this network in patrolling agents. Seven architectures of LSTM have been analysed in this chapter.

More concretely, as outlined in **Chapter 3**, data were first generated using PyTrol, then statistical models were trained using MAPTrainer. The new strategies were then

run in PyTrol, then confronted to the representative centralised and model strategy, namely HCC 0.2, a variant of HPCC, and CR, the decentralised representative.

First experiments have shown that, choosing the maximum probability node in the distribution generated by the LSTM network leads the patrolling system to visit perpetually the same set of nodes, resulting in a non-robust system. In order to remedy that, the procedure of node selection has been turned into a directed-random selection where a random drawing is carried out according to the probability distribution generated by the LSTM network, giving rise to the RLPM strategy.

The evaluation demonstrated that RLPM-2-50 and RLPM-1-50, the strategies resulting from the LSTM architectures with 2 layers and 50 neurons, and 1 layers and 50 neurons, respectively, are globally the best ones. RLPM-2-50 is the best upon the MI — a central tendency measure —, whereas RLPM-1-50 is the best upon the QMI — a measure which tends to emphasise the node with long time intervals without visits — and the average idleness — measure which can be expressed as a function of the MI and QMI.

As far as for each topology the proper architecture is chosen, the experiments have shown mitigated results. In an extreme mission where communications are prohibited, a learning strategy based on the LSTM architecture can perform missions in a context of crisis with reasonable performance which are, for some topologies and numbers of agents, even better than CR, the decentralised representative. Moreover, although CR and RLPM are decentralised strategies by design, the decision procedure of RLPM hinges on a random step, in other words, without this additional step the system, being too rigid it tends to lead agents to converge indefinitely towards a small set of nodes. This entails that the learning system studied here that rests upon the LSTM architecture is not adaptive.

Also, it could be argued that LSTM networks better learn to navigate through the area to patrol if the structure of the graph, that is to say the topology on its own, was strongly integrated into the network, in other terms if the network was wired so that it would not be able to predict a node which is not a neighbour of that provided to it as input. Therefore, in the next chapter an extension of this application will be studied, where instead of pretraining the LSTM networks, an analytical initialisation which takes into account the topology of the area to patrol will be performed.

It is worth noting that beyond the narrow frame of MAP, the procedure laid down here, which has given rise to a new MAP strategy, can be applied to any decision-making problem which has a central communication node.

Chapter 5

RAMPAGER: a strategy relying on structure-guided LSTM initialisation

In the previous chapter, RLPM, a concrete implementation of the generic strategy Path-Maker, has been introduced. Until now, the structure of the graph to patrol has been integrated in the LSTM network wiring from a pretraining stage where edges of the graph are provided to the network. This chapter introduces an analytical procedure to initialise any LSTM network of the same width as the number of nodes of the studied topology with respect to its structure, in place of the pretraining stage used in the previous chapter. Such an approach constitutes a *topology-guided initialisation*, or more generally, a *structure-guided initialisation*. The adjective "guided" has to be understood here as an instructional sequence of settings to carry out in order to set the considered statistical model so that it be consistent with the problem's constraints. The constraint here, is the structure of the topology to patrol.

The strategy consisting in RLPM agents using an LSTM initialised analytically according to the topology is named ***RA****ndom* ***M****uliagent* ***PA****trollin****G*** *LSTM-Path-Mak****ER*** (RAMPAGER). First, a formal and theoretical setting for any topology will be determined; concrete values are also proposed. Then, as in the previous chapters, preliminary experiments allow selecting the best LSTM architecture are carried out. Finally, those models are evaluated in training, on the HPCC 0.5 database, and in simulation.

5.1 Procedure of training

Unlike the previous chapter, there is no pretraining stage here. In fact, structure-guided initialisation is supposed to remedy to the pitfalls of the pretraining. Also, the main training remains unchanged.

5.1.1 Structure-guided initialisation: an analytical initialisation

What follows establishes a demonstration aiming at enabling any LSTM network which satisfies some features detailed below to integrate the structure of the graph to patrol. Namely, it is constrained by the graph depicting the area, i.e. it is not able to predict a next node to visit which is not a neighbour of the input. Two cases are distinguished in this procedure: architectures with one LSTM layer — here, there is identity between the notion of LSTM layer and LSTM block which are used interchangeably —, and ones with more than one LSTM layer.

The structure of the problem — here the structure of the topology depicting the area to patrol — confines us to only consider LSTM networks with a width equal to the number of nodes, N, of the graph. In other terms, all of its blocks are N-dimensional, with $n_y = n_x = N$. $L \in \mathbb{N}^*$ stands for the number of LSTM layers of the network.

Then, $\forall l \in [|1, L|]$, $W^{l,ix}, W^{l,ih}, W^{l,fx}, W^{l,fh}, W^{l,ox}, W^{l,oh}, W^{l,cx}$ and $W^{l,ch}$ are of dimension $N \times N$, x_t^l, and h_{t-1}^l of dimension N, and $b^{l,i}, b^{l,f}, b^{l,o}$ and $b^{l,c}$ of dimension N.

In the following, let:

- X^N be the space of one-hot vectors of dimension N, i.e. $\forall x \in X^N$, $x \in \{0,1\}^N$ and $\sum_{i=1}^{N} x_i = 1$; a consequence is that $card(X^N) = N$,
- $n_x \in \mathbb{N}^*$ be the input dimension, and $n_y \in \mathbb{N}^*$ the output dimension; as stated above $n_x = n_y = N$ is the number of nodes in the graph G,
- $\forall t \in \mathbb{N}^*, \forall l \in [|1, L|]$, $x_t^l \in X^N$ and $h_{t-1}^l \in [0,1]^N$ be the input and the hidden state, respectively, of the layer l,
- $\gamma : X^N \to V$ the function mapping one-hot vectors with their corresponding nodes in V, such as $\gamma = \beta^{-1}$, where β is the function mapping V to X^N described in **Subsection 4.1.2**.

In the following, $\forall a \in \overline{\mathbb{R}}$, and $\forall n, p \in \mathbb{N} : \forall A \in \mathcal{M}_{n\times p}(\overline{\mathbb{R}})$:

- $A = a$ means that $\forall i = 1, \cdots, n$, $\forall j = 1, \cdots, p$, $A_{i,j} = a$.
- $A_{i_0} = a$, such as $i_0 \in [|1, n|]$, means that $\forall j = 1, \cdots, p$, $A_{i_0,j} = a$.

Lastly, the initialisation will be performed theoretically, i.e. $\forall t \in \mathbb{N}^*$, as if it was needed that constraints hold for any time step.

5.1.1.1 Case 1: one LSTM block.

For the sake of simplicity, in this particular case vectors and parameter matrices are indexed by the unique layer.

The initialisation is carried out using backward induction from the output of the LSTM block towards its input while initialising analytically the least number of parameters. In doing so, the LSTM block's output, simply noted h_t, is first considered:

$$\begin{aligned} \forall t \in \mathbb{N}^* : x_t \in X^N,\ h_t &= o_t * \tanh(c_t) \\ &= \sigma(W^{ox}\ x_t + W^{oh}\ h_{t-1} + b^o) * \tanh(c_t) \end{aligned} \tag{5.1.1}$$

W^{ox}, W^{oh}, b^o. In the specific framework of MAP, given that theoretically the output is to be a neighbour of x_t, i.e. $\forall t \in \mathbb{N}^* : x_t \in X^N,\ h_{t-1} = x_t$ the information carried by h_{t-1} is accordingly not taken into account. This then leads to: $W^{oh} = 0$.

According to **Eq. A.3.6**, let, $\forall x_t \in X^N$, $\forall v \in V$, $z_{t,v} = W_v^{ox}\ x_t + b_v^o$ be the v^{th} component of the signal coming into the output gate's activation. To constrain the LSTM block to generate values as high as possible for the neighbours of x_t, according to **Eq. 5.1.1** the output gate must be opened for them:

$$\forall t \in \mathbb{N}^* : x_t \in X^N,\ \forall v \in V,\ h_{t-1,v} = \begin{cases} 1 \text{ if } v \text{ is a neighbourg of } \gamma(x_t) \\ 0 \text{ otherwhise} \end{cases} \tag{5.1.2}$$

On the output gate, this leads to:

$$\forall t \in \mathbb{N}^* : x_t \in X^N,\ \forall v \in V,\ o_{t,v} \approx \begin{cases} 1 \text{ if } v \text{ is a neighbour of } \gamma(x_t) \\ 0 \text{ otherwhise} \end{cases}$$

$$\forall t \in \mathbb{N}^* : x_t \in X^N,\ \forall v \in V,\ \sigma(z_{t,v}) \approx \begin{cases} 1 \text{ if } v \text{ is a neighbour of } \gamma(x_t) \\ 0 \text{ otherwhise} \end{cases}$$

$$\iff \forall t \in \mathbb{N}^* : x_t \in X^N,\ \forall v \in V,\ z_{t,v} \approx \begin{cases} +\infty \text{ if } v \text{ is a neighbour of } \gamma(x_t) \\ -\infty \text{ otherwhise} \end{cases}$$

$$\implies \forall t \in \mathbb{N}^* : x_t \in X^N,\ \forall v \in V,\ W_v^{ox}\ x_t + b_v^o \approx \begin{cases} +\infty \text{ if } v \text{ is a neighbour of } \gamma(x_t) \\ -\infty \text{ otherwhise} \end{cases}$$

with $\forall v \in V$, W_v^{ox} denoting the v^{th} line of W^{ox}.
$\forall t \in \mathbb{N}^* : x_t \in X^N$, a feasible solution is:

$$\forall t \in \mathbb{N}^* : x_t \in X^N,\ \forall v \in V,\ b^o \in \mathbb{R}^N,\ W_{v,\gamma(x_t)}^{ox} \approx \begin{cases} +\infty \text{ if } v \text{ is a neighbour of } \gamma(x_t) \\ -\infty \text{ otherwhise} \end{cases} \tag{5.1.3}$$

leading to:

$$\forall u, v \in V,\ b^o \in \mathbb{R}^N,\ W^{oh} = 0,\ W^{ox} \in \mathcal{M}_{N,N}(\overline{\mathbb{R}}) : W_{v,u}^{ox} = \begin{cases} +\infty \text{ if } v \text{ is a neighbour of } u \\ -\infty \text{ otherwhise} \end{cases} \tag{5.1.4}$$

For a graph where the nodes 2 and N are neighbours of 1, and 1 and $N-1$ are neighbours of N, an extended form of W^{ox} is:

$$W^{ox} = \begin{bmatrix} -\infty & \dots & \dots & -\infty & +\infty \\ +\infty & -\infty & & & \\ \vdots & & -\infty & & \\ \vdots & & & \ddots & \\ +\infty & -\infty & \dots & +\infty & -\infty \end{bmatrix} \tag{5.1.5}$$

$\mathbf{W^{cx}}$, $\mathbf{W^{ch}}$, $\mathbf{b^c}$. According to **Eq. A.3.8**, the components of the flow c_t coming from the cell corresponding to the neighbours of $\gamma(x_t)$ must be positive to ensure the output

gate set previously to let the neighbours components traverse *positively* the output gate. Otherwise, the LSTM block's output may generate a distribution whose the neighbours of x_t have a lower probability than the non-neighbour nodes. It then follows that:

$$\begin{aligned}
&\forall t \in \mathbb{N}^* : x_t \in X^N,\ \forall v \in V :\ v \text{ neighbour of } \gamma(x_t),\ \tanh(c_{t,v}) \approx 1 \\
\iff &\forall t \in \mathbb{N}^* : x_t \in X^N,\ \forall v \in V :\ v \text{ neighbour of } \gamma(x_t), \\
&\quad \tanh(\underbrace{i_{t,v}}_{>0} * \tanh(W_v^{cx}\ x_t + W_v^{ch}\ h_{t-1,v} + b_v^c) + \underbrace{f_{t,v} * c_{t-1,v}}_{>0 \text{ if } c_{t-1,v} > 0}) \approx 1 \\
\implies &\forall t \in \mathbb{N}^* : x_t \in X^N,\ \forall v \in V :\ v \text{ neighbour of } \gamma(x_t), \\
&\quad W_v^{cx}\ x_t + \underbrace{W_v^{ch}}_{=0}\ h_{t-1,v} + b_v^c \approx +\infty \\
\implies &W^{ch} = 0,\ b^c \in \mathbb{R}^N,\ \forall t \in \mathbb{N}^* : x_t \in X^N,\ \forall v \in V :\ v \text{ neighbour of } \gamma(x_t), \\
&\quad W_v^{cx}\ x_t \approx +\infty
\end{aligned} \tag{5.1.6}$$

where, as previously, the information carried by h_{t-1} is not taken into account, leading to $W^{ch} = 0$. Considering $\forall t \in \mathbb{N}^*$, $x_t \in X^N$ is a one-hot vector, it follows:

$$\begin{aligned}
&W^{ch} = 0,\ b^c \in \mathbb{R}^N,\ W^{cx} \in \mathcal{M}_{N,N}(\mathbb{R}) : \\
&\quad \forall u, v \in V :\ v \text{ neighbour of } u, W_{v,u}^{cx} \approx +\infty
\end{aligned} \tag{5.1.7}$$

To avoid any step backward, it is necessary to ensure the LSTM block will not generate a distribution which emphasises the current node $\gamma(x_t)$ in the next step. In fact, by nature the current node $\gamma(x_t)$ will be a neighbour in the next decision step. Therefore, the flow stemming from c_t and traversing the second tanh activation present in **Eq. A.3.8** has to penalise the current node, namely it has to be negative. To do so, considering that the codomain of tanh is $[-1, 1]$, it is first needed that the output of the first tanh activation present in **Eq. A.3.7** be always close to -1 for the current node. According to the same equation, it then follows:

$$
\begin{aligned}
&\forall t \in \mathbb{N}^* : x_t \in X^N : u = \gamma(x_t),\ \tanh(c_{t,u}) \approx -1 \\
\iff &\forall t \in \mathbb{N}^* : x_t \in X^N : u = \gamma(x_t), \\
&\qquad \tanh(\underbrace{i_{t,u}}_{>0} * \tanh(W_u^{cx}\, x_t + W_u^{ch}\, h_{t-1,u} + b_u^c) + \underbrace{f_{t,u} * c_{t-1,u}}_{>0 \text{ if } c_{t-1,u}>0}) \approx -1 \\
\implies &\forall t \in \mathbb{N}^* : x_t \in X^N : u = \gamma(x_t), \\
&\qquad W_u^{cx}\, x_t + \underbrace{W_u^{ch}}_{=0}\, h_{t-1,u} + b_u^c \approx -\infty \\
\implies &W^{ch} = 0,\ b^c \in \mathbb{R}^N, \forall t \in \mathbb{N}^* : x_t \in X^N, u = \gamma(x_t), \\
&\qquad W_u^{cx}\, x_t \approx -\infty
\end{aligned}
\tag{5.1.8}
$$

Considering $\forall t \in \mathbb{N}^* : x_t \in X^N$, x_t is a one-hot vector, it follows:

$$
\begin{gathered}
W^{ch} = 0,\ b^c \in \mathbb{R}^N, W^{cx} \in \mathcal{M}_{N,N}(\overline{\mathbb{R}}) : \\
\forall u \in V, W_{u,u}^{cx} \approx -\infty
\end{gathered}
$$

Thus, $W^{ch} = 0,\ b^c \in \mathbb{R}^N,\ W^{cx} \in \mathcal{M}_{N,N}(\overline{\mathbb{R}}) : \forall u, v \in V,\ W_{v,u}^{cx} = +\infty$ if v is a neighbour of u apart from $W_{u,u}^{cx} \approx -\infty$, otherwise $W_{v,u}^{cx} = w \in \mathbb{R} : \tanh(w) << \tanh(+\infty)$, is a feasible solution.

As previously, for a graph where the nodes 2 and N are neighbours of 1, while 1 and $N-1$ are neighbours of N, an extended form of W^{cx} is:

$$
W^{cx} = \begin{bmatrix}
-\infty & +\infty & w_{1,3} & \dots & +\infty \\
+\infty & -\infty & \dots & & \\
w_{3,1} & & -\infty & & \\
\vdots & & & \ddots & \\
+\infty & w_{N,2} & \dots & +\infty & -\infty
\end{bmatrix}
\tag{5.1.9}
$$

$\mathbf{W^{ix}}$, $\mathbf{W^{ih}}$, $\mathbf{b^i}$. To preserve the integrity of the values output by the tanh activation at the entrance to the LSTM block, always according to **Eq. A.3.7** the input gate must be beforehand opened for these values, values which correpond to those of the neighbour of $\gamma(x_t)$, that is to say they must not be filtered by the input gate:

$$
\begin{aligned}
&\forall t \in \mathbb{N}^* : x_t \in X^N : v \text{ neighbour of } \gamma(x_t),\ i_{t,v} \approx 1 \\
\iff &\forall t \in \mathbb{N}^* : x_t \in X^N : v \text{ neighbour of } \gamma(x_t), \\
&\qquad \sigma(W_v^{ix}\ x_t + W_v^{ih}\ h_{t-1} + b_v^i) \approx 1 \\
\implies &\forall t \in \mathbb{N}^* : x_t \in X^N : v \text{ neighbour of } \gamma(x_t), \\
&\qquad W_v^{ix}\ x_t + \underbrace{W_v^{ih}}_{=0}\ h_{t-1} + b_v^i \approx +\infty \\
\implies &\forall t \in \mathbb{N}^* : x_t \in X^N : v \text{ neighbour of } \gamma(x_t),\ W^{ih} = 0,\ b^i \in \mathbb{R}^N \\
&\qquad W_v^{ix}\ x_t \approx +\infty
\end{aligned}
\tag{5.1.10}
$$

Considering $\forall t \in \mathbb{N}^* : x_t \in X^N$, x_t is a one-hot vector, it follows:

$$
\begin{aligned}
W^{ih} = 0,\ b^i \in \mathbb{R}, W^{ix} \in \mathcal{M}_{N,N}(\overline{\mathbb{R}}) : \\
\forall u, v \in V : v \text{ neighbour of } v,\ W_{v,u}^{ix} = +\infty
\end{aligned}
$$

$W^{ih} = 0,\ b^i \in \mathbb{R}^N$, $W^{ix} \in \mathcal{M}_{N,N}(\overline{\mathbb{R}}) : \forall u, v \in V$, $W_{v,u}^{ix} = +\infty$ if v is a neighbour of v, otherwise $W_{v,u}^{ix} = w \in \mathbb{R} : \sigma(w) << \sigma(+\infty)$ is a feasible solution.

As previously, for a graph where the nodes 2 and N are neighbours of 1, and 1 and $N-1$ are neighbours of N, an extended form of W^{ix} is:

$$
W^{ix} = \begin{bmatrix}
w_{1,1} & +\infty & w_{1,3} & \dots & +\infty \\
+\infty & w_{2,2} & w_{2,3} & \dots & \vdots \\
\vdots & \ddots & & & \\
\vdots & & \ddots & & \\
+\infty & w_{N,2} & \dots & +\infty & w_{N,N}
\end{bmatrix}
\tag{5.1.11}
$$

$\mathbf{W^{fx}}$, $\mathbf{W^{fh}}$, $\mathbf{b^f}$. Lastly, on the forget gate, as for the input gate and according to **Eq. A.3.7**, the flow of the current node's neighbours must also be opened:

$$
\begin{aligned}
&\forall t \in \mathbb{N}^* : x_t \in X^N : v \text{ neighbour of } \gamma(x_t),\ f_{t,v} \approx 1 \\
\iff &\forall t \in \mathbb{N}^* : x_t \in X^N : v \text{ neighbour of } \gamma(x_t), \\
&\qquad \sigma(W_v^{fx}\ x_t + W_v^{fh}\ h_{t-1} + b_v^f) \approx 1 \\
\implies &\forall t \in \mathbb{N}^* : x_t \in X^N : v \text{ neighbour of } \gamma(x_t), \\
&\qquad W_v^{fx}\ x_t + \underbrace{W_v^{fh}}_{=0}\ h_{t-1} + b_v^f \approx +\infty \\
\implies &\forall t \in \mathbb{N}^* : x_t \in X^N : v \text{ neighbour of } \gamma(x_t),\ W^{fh} = 0,\ b^f \in \mathbb{R}^N \\
&\qquad W_v^{fx}\ x_t \approx +\infty
\end{aligned}
\tag{5.1.12}
$$

Considering $\forall t \in \mathbb{N}^* : x_t \in X^N$, x_t is a one-hot vector, it follows:

$$
\begin{aligned}
W^{fh} = 0,\ b^f \in \mathbb{R}, W^{fx} \in \mathcal{M}_{N,N}(\overline{\mathbb{R}}) : \\
\forall u, v \in V : v \text{ neighbour of } v,\ W_{v,u}^{fx} = +\infty
\end{aligned}
$$

$W^{fh} = 0,\ b^f \in \mathbb{R}^N, W^{fx} \in \mathcal{M}_{N,N}(\overline{\mathbb{R}}) : \forall u, v \in V, W_{v,u}^{fx} = +\infty$ if v is a neighbour of v, otherwise $W_{v,u}^{fx} = w \in \mathbb{R} : \sigma(w) << \sigma(+\infty)$ is a feasible solution.

As previously, for a graph where the nodes 2 and N are neighbours of 1, and 1 and $N-1$ are neighbours of N, an extended form of W^{fx} is:

$$
W^{fx} = \begin{bmatrix}
w_{1,1} & +\infty & w_{1,3} & \dots & +\infty \\
+\infty & w_{2,2} & w_{2,3} & \dots & \vdots \\
\vdots & \ddots & & & \\
\vdots & & \ddots & & \\
+\infty & w_{N,2} & \dots & +\infty & w_{N,N}
\end{bmatrix}
\tag{5.1.13}
$$

Finally, in the case of one LSTM block a theoretical feasible solution could be:

Case 1: The last LSTM block, theoretical setting

$$W^{ix} \in \mathcal{M}_{N,N}(\overline{\mathbb{R}}) : \forall u, v \in V, W^{ix}_{v,u} = \begin{cases} +\infty \text{ if } v \text{ is a neighbour of } u \\ w \in \mathbb{R} : \sigma(w) << \sigma(+\infty) \text{ otherwise} \end{cases}$$

$$W^{fx} \in \mathcal{M}_{N,N}(\overline{\mathbb{R}}) : \forall u, v \in V, W^{fx}_{v,u} = \begin{cases} +\infty \text{ if } v \text{ is a neighbour of } u \\ w \in \mathbb{R} : \sigma(w) << \sigma(+\infty) \text{ otherwise} \end{cases}$$

$$W^{cx} \in \mathcal{M}_{N,N}(\overline{\mathbb{R}}) : \forall u, v \in V,\ W^{cx}_{v,u} = \begin{cases} -\infty \text{ if } u = v \\ +\infty \text{ if } v \text{ is a neighbour of } u \\ w \in \mathbb{R} : \sigma(-\infty) << \sigma(w) << \sigma(+\infty) \text{ else} \end{cases}$$

$$W^{ox} \in \mathcal{M}_{N,N}(\overline{\mathbb{R}}) : \forall u, v \in V,\ W^{ox}_{v,u} = \begin{cases} +\infty \text{ if } v \text{ is a neighbour of } u \\ -\infty \text{ otherwise,} \end{cases}$$

$$W^{ih} = 0, W^{fh} = 0, W^{ch} = 0, W^{oh} = 0,$$

$$b_i \in \mathbb{R}^N, b_f \in \mathbb{R}^N, b_c \in \mathbb{R}^N, b_o \in \mathbb{R}^N \tag{5.1.14}$$

5.1.1.2 Case 2: Several LSTM blocks

In the case of multilayered LSTM networks, the initialisation consists in setting first, the last LSTM block in the same way as in the case of one LSTM block, then in setting intermediate LSTM blocks behaving as identity function, i.e. they output their input: $\forall l \in [|1, L-1|],\ \forall t \in \mathbb{N}^*,\ z^l_t = x^l_t$. As previously, the initialisation will be inferred backwardly from the output gate.

According to **Eq. 5.1.1**, $\forall t \in \mathbb{N}^*,\ \forall l \in [|1, L-1|],\ h^l_t = o^l_t * \tanh(c^l_t)$. It is then necessary to open the output gate for components corresponding to the node itself, that is the flow coming from a node toward itself remains unchanged. **Eq. 5.1.1** also highlights that, given that $[-1, 1]$ is the codomain of tanh, $\forall t \in \mathbb{N}^*, \forall l \in [|1, L-1|], \tanh(c^l_t)$ must convey x^l_t itself.

$\mathbf{W^{l,ox}}, \mathbf{W^{l,oh}}, \mathbf{b^{l,o}}$. Let $l \in [|1, L-1|]$. $\forall t \in \mathbb{N}^*, \forall l \in [|1, L-1|],\ o^l_t$ must output 1 for the node itself and 0 otherwise:

$$\forall t \in \mathbb{N}^* : x_t^l \in X^N,\ \forall v \in V,\ h_{t-1,v}^l = \begin{cases} 1 \text{ if } v = \gamma(x_t^l) \\ 0 \text{ otherwhise} \end{cases} \tag{5.1.15}$$

On the output gate by putting $W^{l,oh} = 0$ as in the previous case, it follows:

$$\begin{aligned}
\forall t \in \mathbb{N}^* : x_t \in X^N,\ \forall v \in V,\ o_{t,v}^l &\approx \begin{cases} 1 \text{ if } v = \gamma(x_t^l) \\ 0 \text{ otherwhise} \end{cases} \\
\forall t \in \mathbb{N}^* : x_t^l \in X^N,\ \forall v \in V,\ \sigma(z_{t,v}^l) &\approx \begin{cases} 1 \text{ if } v = \gamma(x_t^l) \\ 0 \text{ otherwhise} \end{cases} \\
\iff \forall t \in \mathbb{N}^* : x_t^l \in X^N,\ \forall v \in V,\ z_{t,v}^l &\approx \begin{cases} +\infty \text{ if } v = \gamma(x_t^l) \\ -\infty \text{ otherwhise} \end{cases} \\
\implies \forall t \in \mathbb{N}^* : x_t^l \in X^N,\ \forall v \in V,\ W_v^{l,ox}\ x_t^l + b_v^{l,o} &\approx \begin{cases} +\infty \text{ if } v = \gamma(x_t^l) \\ -\infty \text{ otherwhise} \end{cases} \\
\implies \forall t \in \mathbb{N}^* : x_t^l \in X^N,\ b_o \in \mathbb{R}^N, W^{l,ox} &\approx \begin{cases} +\infty \text{ if } v = \gamma(x_t^l) \\ -\infty \text{ otherwhise} \end{cases}
\end{aligned}$$

$\forall l \in [|1, L-1|], W^{l,oh} = 0$, $b^o \in \mathbb{R}^N$, and $W^{l,ox} \in \mathcal{M}_{N,N}(\overline{\mathbb{R}}) : \forall u, v \in V,\ W_{vu}^{l,ox} \approx +\infty$, if v is a neighbour of v, otherwise $W_{v,u}^{l,ox} \approx -\infty$, is then a feasible solution.

An extended form of $W^{l,ox}$ is:

$$W^{l,ox} = \begin{bmatrix} +\infty & -\infty & \dots & -\infty \\ -\infty & +\infty & \dots & -\infty \\ \vdots & & \ddots & \\ -\infty & -\infty & \dots & +\infty \end{bmatrix} \tag{5.1.16}$$

$\mathbf{W^{l,cx}}$, $\mathbf{W^{l,ch}}$, $\mathbf{b^{l,c}}$. Let $l \in [|1, L-1|]$. According to **Eq. A.3.8**, to ensure $\tanh(c_t^l) \to 1$ the cell must output the highest possible value for the current node $v \in V$. Therefore, the first tanh activation — present in **Eq. A.3.7** — has to be as close as possible to 1 for the current node $v \in V$:

$$
\begin{aligned}
&\forall t \in \mathbb{N}^* : x_t^l \in X^N, v = \gamma(x_t^l),\ \tanh(c_{t,v}^l) \approx 1 \\
\iff &\forall t \in \mathbb{N}^* : x_t^l \in X^N, v = \gamma(x_t^l), \\
&\qquad \tanh(\underbrace{i_{t,v}^l}_{>0} * \tanh(W_v^{l,xc}\ x_t^l + W_v^{l,hc}\ h_{t-1,v}^l + b_v^{l,c}) + \underbrace{f_{t,v}^l * c_{t-1,v}^l}_{>0 \text{ if } c_{t-1,v}^l > 0}) \approx 1 \\
\implies &\forall t \in \mathbb{N}^* : x_t^l \in X^N, v = \gamma(x_t^l), \\
&\qquad W_v^{l,cx}\ x_t^l + \underbrace{W_v^{l,ch}}_{=0}\ h_{t-1,v}^l + b_v^{l,c} \approx +\infty \\
\implies &\forall t \in \mathbb{N}^* : x_t^l \in X^N, v = \gamma(x_t^l),\ W^{l,ch} = 0,\ b^{l,c} \in \mathbb{R}^N \\
&\qquad W_v^{l,cx}\ x_t^l \approx +\infty
\end{aligned}
$$

Likewise, the first tanh activation must be equal to 0 for all nodes different from the current node v, which leads similarly to:

$$
\forall t \in \mathbb{N}^* : x_t^l \in X^N, v \neq \gamma(x_t^l), W_v^{l,cx}\ x_t^l \approx 0 \tag{5.1.17}
$$

Considering $\forall t \in \mathbb{N}^* : x_t^l \in X^N$, x_t^l is a one-hot vector, it follows:

$$
\begin{aligned}
W^{l,ch} = 0,\ b^{l,c} \in \mathbb{R}^N,\ W^{l,cx} \in \mathcal{M}_{N,N}(\overline{\mathbb{R}}) : \\
\forall u, v \in V,\ W_{v,u}^{l,cx} \approx \begin{cases} +\infty \text{ if } v = u \\ 0 \text{ otherwhise} \end{cases}
\end{aligned} \tag{5.1.18}
$$

$\forall l \in [|1, L-1|],\ W^{l,ch} = 0$, $b^{l,c} \in \mathbb{R}^N$ and $W^{l,cx} \in \mathcal{M}_{N,N}(\overline{\mathbb{R}}) : \forall u, v \in V,\ W_{v,u}^{l,cx} \approx +\infty$ if $u = v$, otherwise $W_{v,u}^{l,cx} \approx 0$, is then a feasible solution.

An extended form of $W^{l,cx}$ is:

$$\begin{bmatrix} +\infty & 0 & \dots & 0 \\ 0 & +\infty & \dots & 0 \\ \vdots & & \ddots & \\ 0 & 0 & \dots & +\infty \end{bmatrix} \tag{5.1.19}$$

$\mathbf{W^{l,ix}}$, $\mathbf{W^{l,ih}}$, $\mathbf{b^{l,i}}$. Let $l \in [|1, L-1|]$. Then, according to **Eq. A.3.7**, the input gate must be set with the aim of letting the signal coming from the tanh activation pass through. It is then opened for the current node:

$$\begin{aligned} & \forall t \in \mathbb{N}^* : x_t^l \in X^N : v = \gamma(x_t^l),\ i_{v,t}^l \approx 1 \\ & \forall t \in \mathbb{N}^* : x_t^l \in X^N : v = \gamma(x_t^l), \\ & \qquad \sigma(W_v^{l,ix}\ x_t + W_v^{l,ih}\ h_{t-1}^l + b_v^{l,i}) \approx 1 \\ \iff & \forall t \in \mathbb{N}^* : x_t^l \in X^N : v = \gamma(x_t^l), \\ & \qquad W_v^{l,ix}\ x_t + W_v^{l,ih}\ h_{t-1}^l + b_v^{l,i} \approx 1 \\ \implies & \forall t \in \mathbb{N}^* : x_t^l \in X^N, v = \gamma(x_t^l),\ W^{l,ih} = 0,\ b^{l,i} \in \mathbb{R}^N \\ & \qquad W_v^{l,ix}\ x_t^l \approx +\infty \end{aligned} \tag{5.1.20}$$

Likewise, the input gate must be equal to 0 for all nodes different from the current node v, which leads similarly to:

$$\forall t \in \mathbb{N}^* : x_t^l \in X^N, v \neq \gamma(x_t^l), W_v^{l,ix}\ x_t^l \approx -\infty \tag{5.1.21}$$

Considering $\forall t \in \mathbb{N}^* : x_t^l \in X^N$, x_t^l is a one-hot vector, it follows:

$$\begin{aligned} & W^{l,ih} = 0,\ b^{l,i} \in \mathbb{R}^N,\ W^{l,ix} \in \mathcal{M}_{N,N}(\overline{\mathbb{R}}) : \\ & \qquad \forall u, v \in V,\ W_{v,u}^{l,ix} \approx \begin{cases} +\infty \text{ if } v = u \\ -\infty \text{ otherwhise} \end{cases} \end{aligned} \tag{5.1.22}$$

$\forall l \in [|1, L-1|]$, $W^{l,ih} = 0$, $b^{l,i} \in \mathbb{R}^N$ and $W^{l,ix} \in \mathcal{M}_{N,N}(\overline{\mathbb{R}}) : \forall u, v \in V$, $W_{v,u}^{l,ix} \approx +\infty$ if $u = v$, otherwise $W_{v,u}^{l,ix} \approx -\infty$, is then a feasible solution.

An extended form of $W^{l,ix}$ is:

$$\begin{bmatrix} +\infty & -\infty & \dots & -\infty \\ -\infty & +\infty & \dots & -\infty \\ \vdots & & \ddots & \\ -\infty & -\infty & \dots & +\infty \end{bmatrix} \tag{5.1.23}$$

$\mathbf{W^{l,fx}}$, $\mathbf{W^{l,fh}}$, $\mathbf{b^{l,f}}$. Let $l \in [|1, L-1|]$. Lastly, always according to **Eq. A.3.7**, $\forall t \in \mathbb{N}^*$, with the aim that c_t outputs a maximum value for the the current node $v \in V$, the forget gate f_t must let the signal coming from the previous cell state c_{t-1} pass through for this node v, i.e. $f_{t,v}$ must be approximately equal to 1:

$$\begin{aligned} & \forall t \in \mathbb{N}^* : x_t^l \in X^N : v = \gamma(x_t^l),\ f_{v,t}^l \approx 1 \\ & \forall t \in \mathbb{N}^* : x_t^l \in X^N : v = \gamma(x_t^l), \\ & \qquad \sigma(W_v^{l,fx}\ x_t + W_v^{l,fh}\ h_{t-1}^l + b_v^{l,f}) \approx 1 \\ \iff & \forall t \in \mathbb{N}^* : x_t^l \in X^N : v = \gamma(x_t^l), \\ & \qquad W_v^{l,fx}\ x_t + W_v^{l,fh}\ h_{t-1}^l + b_v^{l,f} \approx 1 \\ \implies & \forall t \in \mathbb{N}^* : x_t^l \in X^N, v = \gamma(x_t^l),\ W^{l,fh} = 0,\ b^{l,f} \in \mathbb{R}^N \\ & \qquad W_v^{l,fx}\ x_t^l \approx +\infty \end{aligned} \tag{5.1.24}$$

Likewise, the forget gate must be equal to 0 for all nodes different from the current node v, which leads similarly to:

$$\forall t \in \mathbb{N}^* : x_t^l \in X^N, v \neq \gamma(x_t^l), W_v^{l,fx}\ x_t^l \approx -\infty \tag{5.1.25}$$

Considering $\forall t \in \mathbb{N}^* : x_t^l \in X^N$, x_t^l is a one-hot vector, it follows:

$$\begin{aligned} & W^{l,fh} = 0,\ b^{l,f} \in \mathbb{R}^N,\ W^{l,fx} \in \mathcal{M}_{N,N}(\overline{\mathbb{R}}) : \\ & \qquad \forall u, v \in V,\ W_{v,u}^{l,fx} \approx \begin{cases} +\infty & \text{if } v = u \\ -\infty & \text{otherwhise} \end{cases} \end{aligned} \tag{5.1.26}$$

$\forall l \in [|1, L-1|],\ W^{l,fh} = 0$, $b^{l,f} \in \mathbb{R}^N$ and $W^{l,fx} \in \mathcal{M}_{N,N}(\overline{\mathbb{R}}) : \forall u, v \in V,\ W^{l,fx}_{v,u} \approx +\infty$ if $u = v$, otherwise $W^{l,fx}_{v,u} \approx -\infty$, is then a feasible solution.

An extended form of $W^{l,fx}$ is:

$$W^{fx} = \begin{bmatrix} +\infty & -\infty & \dots & -\infty \\ -\infty & +\infty & \dots & -\infty \\ \vdots & & \ddots & \\ -\infty & -\infty & \dots & +\infty \end{bmatrix} \tag{5.1.27}$$

Finally, in the case of several LSTM blocks, for each intermediate block a theoretical feasible solution could be:

Case 2: Several LSTM blocks, theoretical setting for the intermediate LSTM layers

Let $l \in [|1, L-1|]$,

$$\begin{aligned}
&W^{l,ix} \in \mathcal{M}_{N,N}(\overline{\mathbb{R}}) : \forall u, v \in V,\ W^{l,ix}_{v,u} = \begin{cases} +\infty \text{ if } u = v \\ -\infty \text{ otherwise} \end{cases} \\
&W^{l,fx} \in \mathcal{M}_{N,N}(\overline{\mathbb{R}}) : \forall u, v \in V,\ W^{l,fx}_{v,u} = \begin{cases} +\infty \text{ if } u = v \\ -\infty \text{ otherwise} \end{cases} \\
&W^{l,cx} \in \mathcal{M}_{N,N}(\overline{\mathbb{R}}) : \forall u, v \in V,\ W^{l,cx}_{v,u} = \begin{cases} +\infty \text{ if } u = v \\ 0 \text{ otherwise} \end{cases} \\
&W^{l,ox} \in \mathcal{M}_{N,N}(\overline{\mathbb{R}}) : \forall u, v \in V,\ W^{l,ox}_{v,u} = \begin{cases} +\infty \text{ if } u = v \\ -\infty \text{ otherwhise} \end{cases} \\
&W^{l,ih} = 0,\ W^{l,fh} = 0,\ W^{l,ch} = 0,\ W^{l,oh} = 0, \\
&b^l_i \in \mathbb{R}^N, b^l_f \in \mathbb{R}^N, b^l_c \in \mathbb{R}^N, b^l_o \in \mathbb{R}^N
\end{aligned} \tag{5.1.28}$$

5.1.1.3 Softmax layer

The softmax layer is then parameterised so that the output of the last LSTM block is rescaled, while the order of its components is preserved. Considering the equation of a softmax layer:

$$\forall t \in \mathbb{N}^* : h_t \in \mathbb{R}^N,\ y_t = \sigma_{sm}(W^{sm} h_t + b^{sm}) \tag{5.1.29}$$

where σ_{sm} is the softmax function, then $W^{sm} = k \cdot Id$ where k is a scale factor.

5.1.1.4 Concrete values

Concrete values are set so that $\forall a, b \in \overline{\mathbb{R}}$, $\forall M \in \mathcal{M}_{N,N}(\overline{\mathbb{R}})$ such as a, b are elements of M, if $a \approx +\infty$, $a \approx -\infty$ respectively, then b is chosen so as $\sigma(a) >> \sigma(b)$ or $\tanh(a) >> \tanh(b)$, $\sigma(a) << \sigma(b)$ or $\tanh(a) << \tanh(b)$ respectively.

After having tested computationally different values with some LSTM models, it has turned out that:

- $\sigma(-7) \approx 0.000911 \approx 0 \implies -7 \approx -\infty$ w.r.t. σ,
- $\tanh(0) = 0$,
- $\sigma(7) \approx 0.999 \approx 1 \implies 7 \approx +\infty$ w.r.t. σ,
- $\tanh(7) \approx 0.999998 \approx 1 \implies 7 \approx +\infty$ w.r.t. $\tanh$,

For the last LSTM block, it then follows:

Case 1: The last LSTM block, concrete setting

$W^{ix} = 0$ and $\forall u, v \in V, v$ is a neighbour of $u,\ W^{ix}_{v,u} = 7$
$W^{fx} = 0$ and $\forall u, v \in V, v$ is a neighbour of $u,\ W^{fx}_{v,u} = 7$
$W^{cx} = 0$ and $\forall u, v \in V, v$ is a neighbour of $u,\ W^{cx}_{v,u} = 7$ apart from $W^{cx}_{u,u} = -7$
$W^{ox} = -7$ and $\forall u, v \in V, v$ is a neighbour of $u, W^{ox}_{v,u} = 7$
$W^{ih} = 0, W^{fh} = 0, W^{ch} = 0, W^{oh} = 0,$
$b_i = 0, b_f = 0, b_c = 0, b_o = 0$

For an intermediate LSTM block:

Case 2: Several LSTM blocks, concrete setting for the intermediate LSTM layers

$$
\begin{aligned}
&\text{Let } l \in [|1, L-1|], \\
&\qquad W^{l,ix} = -7 \text{ and } \forall v \in V,\ W^{l,ix}_{v,v} = 7 \\
&\qquad W^{l,fx} = -7 \text{ and } W^{l,fx}_{v,v} = 7 \\
&\qquad W^{l,cx} = 0 \text{ and } W^{l,cx}_{v,v} = 7 \\
&\qquad W^{l,ox} = -7 \text{ and } W^{l,ox}_{v,v} = 7 \\
&\qquad W^{l,ih} = 0,\ W^{l,fh} = 0,\ W^{l,ch} = 0,\ W^{l,oh} = 0, \\
&\qquad b^l_i = 0, b^l_f = 0, b^l_c = 0, b^l_o = 0
\end{aligned}
$$

Finally for the softmax layer:

The sofmax layer

$$W^{sm} = 5 \cdot Id$$

5.2 Experiments and results

This section describes the assessment of RLPM when it is preinitialised following the setting established in the previous section.

5.2.1 Conduct of experiments

Apart from an additional step consisting in selecting the best LSTM architecture to assess in simulation, the conduct of experiments is identical to that presented in the previous chapter in **Section 4.3.1**. This additional step is carried out just before the final simulation step, namely step 5, and is described in the next paragraph. The conduct becomes accordingly:

1. Scenarios,
2. Missions,
3. Training of statistical models,
4. Choice of the LSTM setting,

5. Simulation.

4. Choice of the LSTM setting. In this step, some LSTM parameters are calibrated and different LSTM architectures are assessed in simulation to select the best one according to relevant criteria.

5.2.2 Preliminary experiments: selection of the LSTM setting

5.2.2.1 Choice of the horizon h in the TBPTT algorithm

In order to determine the best value for the parameter h of the TBPTT algorithm presented in **Apprendix A.3.4.1**, several values of the latter were tested in training for all of the scenarios studied in this chapter. For examples, values for the LSTM network $(2, 50)$ trained on scenario {HPCC 0.5, A, 15} are presented in **Table 5.1**. Considering hardware constraints regarding the capacity of the GPU, the choice of h influences that of the batch size. Therefore, the latter is also noted for each tested value of h. The highest batch size, evaluated on the training database of size 8000 sequences, is preferred to foster the quality of training.

h	Batch size	Validation cost	Accuracy
Full	5154	1.17	54.16
Half	8004	1.14	55.29
200	8004	1.16	54.32
100	8004	1.11	55.96
50	8004	1.06	57.80
25	8004	1.00	59.77
10	8004	0.94	61.82
5	8004	**0.93**	**62.03**
1	8004	0.95	60.86

Table 5.1 Parameters and evaluation metrics for differents values of h, the horizon of the TBPTT algorithm, for the LSTM network $(2, 50)$ on {HPCC 0.5, A, 15}.

Results on all of the topologies have shown that, generally, $h = 5$ is the best value in training; it is thereupon the value which has been retained to train the LSTM networks in the following.

5.2.2.2 Architecture choice

Four candidates of LSTM architecture have then been considered: $(1, 50)$, $(2, 50)$, $(3, 50)$, $(4, 50)$, giving rise to the corresponding RLPM variants. They are assessed on A, for 5, 10, 15 and 25 agents, then the best network is selected to be trained for other scenarios.

Fig. 5.1, **Fig. 5.2**, and **Fig. 5.3** show the performances of the four evaluated networks, on the normalised WI, QMI and average idleness, respectively. Besides, **Table 5.2** shows these criteria averaged over the numbers of agents for the four LSTM networks.

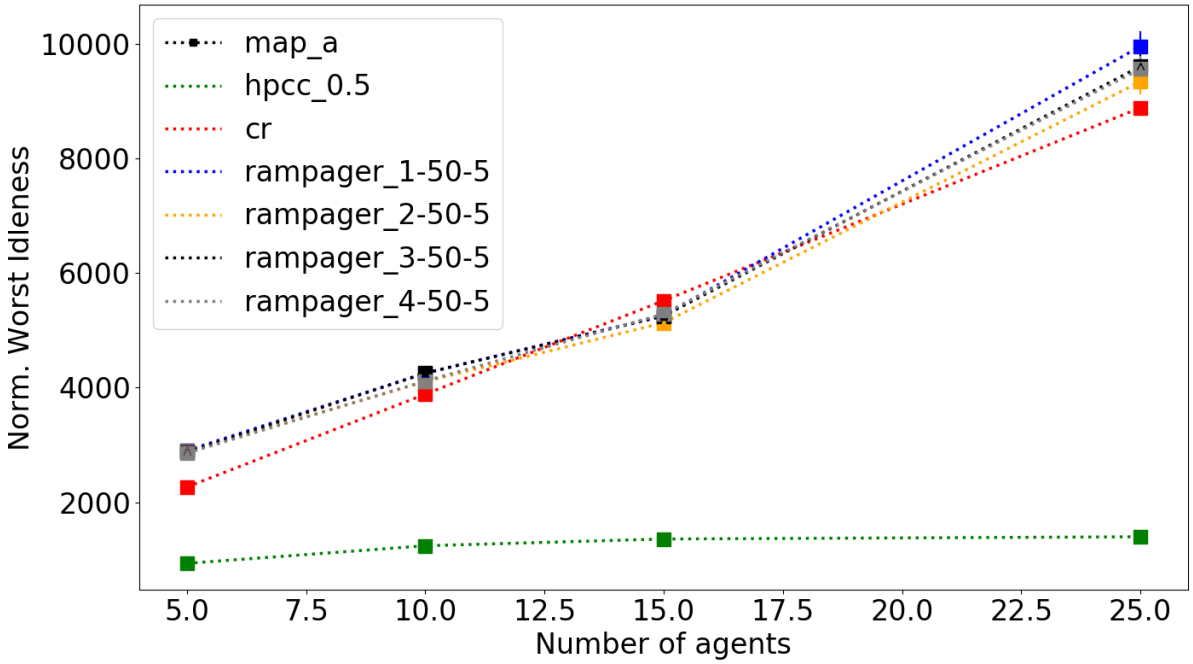

Fig. 5.1 Normalised worst idleness for 4 RLPM variants on the graph A

Architectures	Norm. WI	Norm. QMI	Norm. Iav
(1, 50)	5590	462	391
(2, 50)	**5363**	**458**	**380**
(3, 50)	5493	466	390
(4, 50)	5453	465	389

Table 5.2 Evaluation criteria averaged over the numbers of agents of some LSTM architectures evaluated in simulation on A.

It appears undoubtedly that the LSTM network $(2, 50)$ is the best one for all of the criteria, although the other ones be scarcely higher. Therefore, $(2, 50)$ is chosen to be

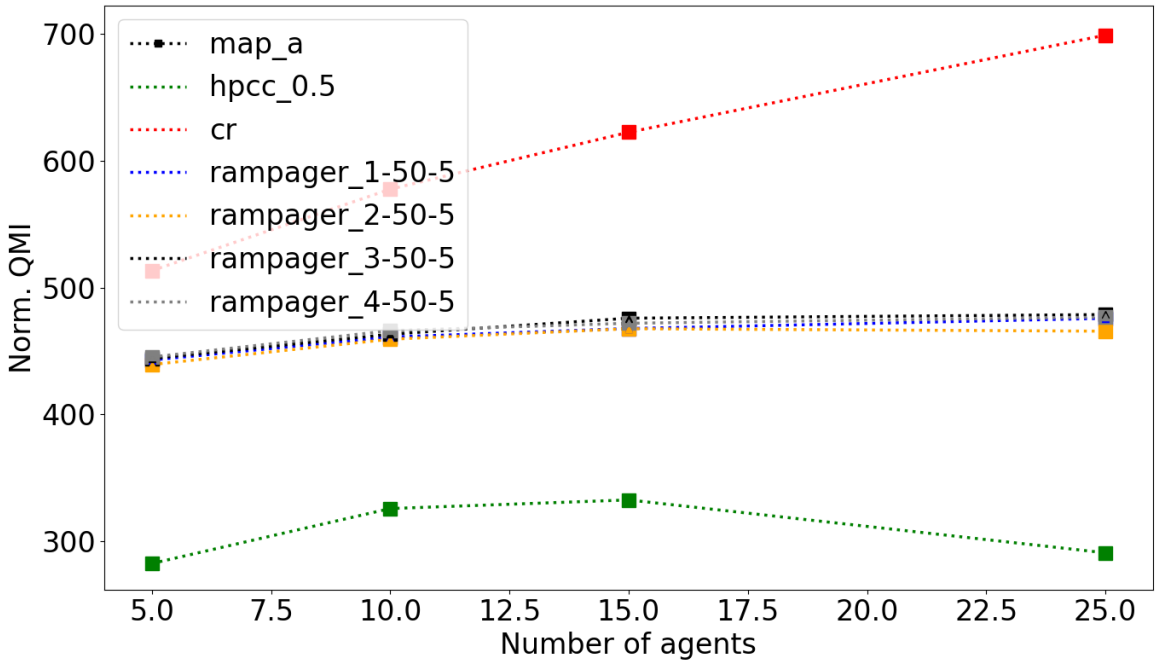

Fig. 5.2 Normalised QMI for 4 RLPM variants on the graph A

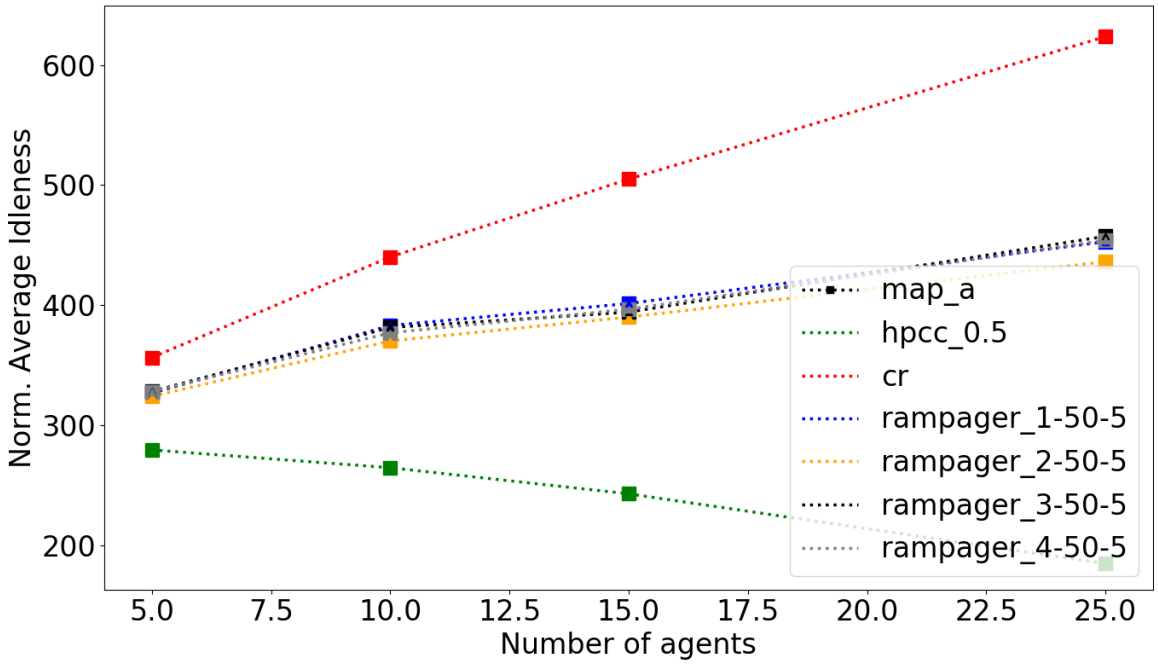

Fig. 5.3 Normalised average idleness for 4 RLPM variants on the graph A

trained on all of the topologies and experimented in simulation. RAMPAGER 2-50 is then retained to be assessed.

5.2.3 Training settings

In this chapter, unlike the previous one, the HPCC 0.5 database is used to train the statistical model retained; HPCC 0.5 is thereupon the model strategy for RAMPAGER 2-50 in what follows. In addition, 24 scenarios are considered, which entails $24\ 2-50$ LSTM networks to train and assess in simulation. Here, these networks are trained using the *Adagrad* algorithm with a learning rate of 0.1, over 500 epochs.

5.2.4 Training results

The total cost, i.e. the cost over the whole database before starting the training stage, and the corresponding accuracy, are of 2.65 and 20% respectively. Then, **Fig. 5.4** and **Fig. 5.5** show the validation cost, namely the cross-entropy over the validation database, and the accuracy, respectively, of the $(2, 50)$-LSTM network during the training on {HPCC 0.5, A, 15}. The accuracy is also calculated over the validation database. The validation cost ranges from 1.09, after the first epoch of training, to 0.86, after the last one, and the accuracy from 55.82% to 64.31%. Moreover, according to these two charts, the model converged after approximately 200 epochs.

Fig. 5.4 Validation cost of the $(2, 50)$-LSTM network trained on {HPCC 0.5, A, 15}

Fig. 5.6, **Fig. 5.7**, **Fig. 5.8**, **Fig. 5.9** and **Fig. 5.9** show the distribution of:

- the weights of all the gates of the input x_t,

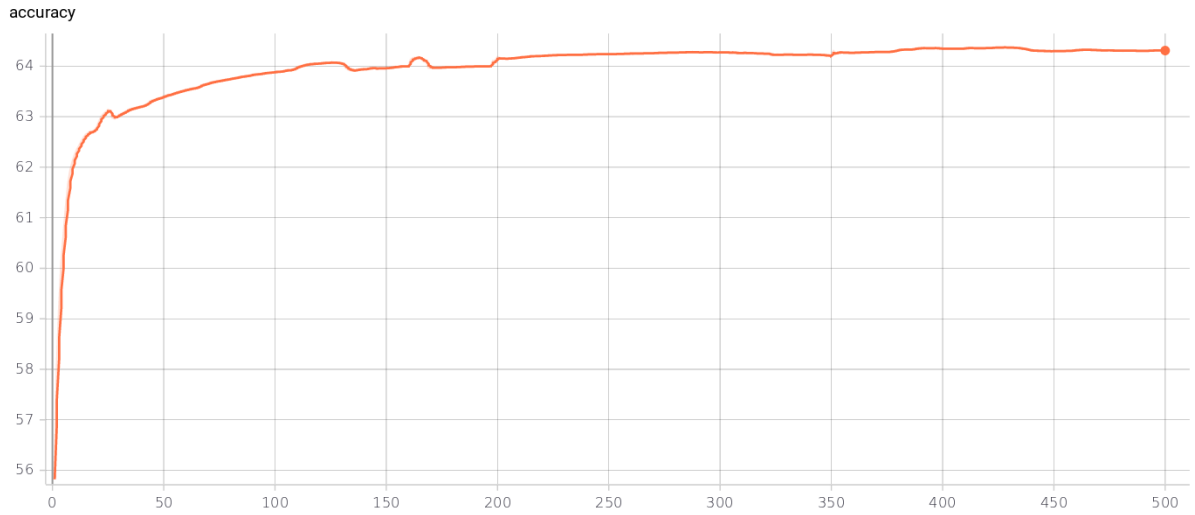

Fig. 5.5 Accuracy of the $(2, 50)$-LSTM network trained on {HPCC 0.5, A, 15}

- the biases of all the gates of the input x_t,
- the weights of all the gates of the hidden state h_{t-1},
- the biases of all the gates of the hidden state h_{t-1},
- the weights and biases of the last layer, namely the sofmax layer,

respectively. Considering parameters in the intermediate layer barely varied, their evolution is not analysed here and are left to the reader in **Appendix B.2.0.1**.

For all of the weight matrices of the last LSTM layer, both at epoch 0 and at the last epoch, parameters are roughly centred around one or two values, they are characterised by a peak around this value. After the training stage, the distributions of these parameters show the same pattern while remaining centred around the same values, although they spread over a larger range of values around their peak and be more flattened. This evolution is the sign of the tendency of the model to have slightly evolved parameters initialised analytically to take into account the paths of nodes presented to the network, from the topology captured in the network over the analytical initialisation stage.

With regard to the bias vectors, their distributions evolved sporadically with no specific central values. The patterns of each distribution at epoch 0 and 500, the first and the last ones respectively, are different. This leads to conclude that biases varied to a large extent, and unlike to weights, they were largely subject to learning.

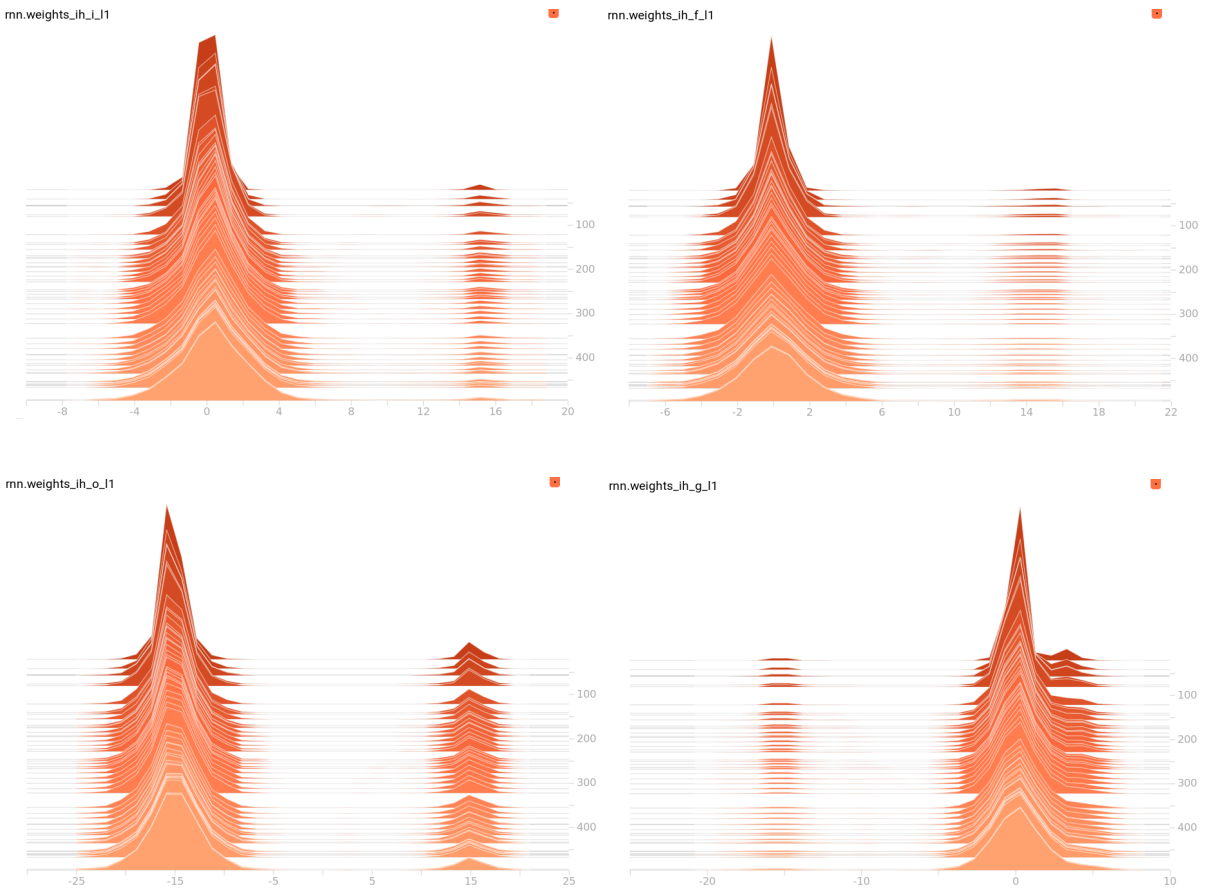

Fig. 5.6 Distribution during the training of the weights of the input x_t, upon the input (i_t), forget (f_t), output (o_t) gates and the cell state (g_t), respectively, for the $(2, 50)$-LSTM network trained on {HPCC 0.5, A, 15}

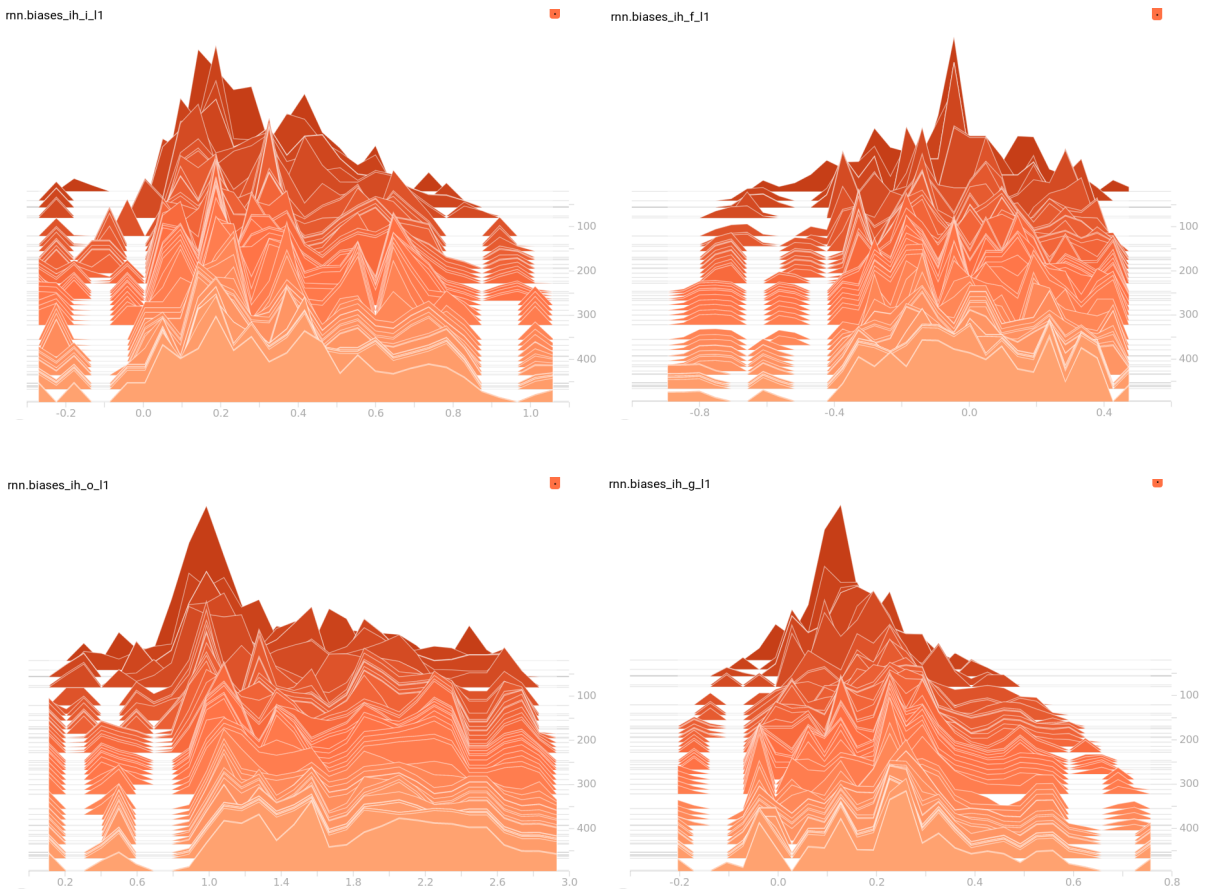

Fig. 5.7 Distribution during the training of the biases of the input x_t, upon the input (i_t), forget (f_t), output (o_t) gates and the cell state (g_t), respectively, for the $(2, 50)$-LSTM network trained on {HPCC 0.5, A, 15}

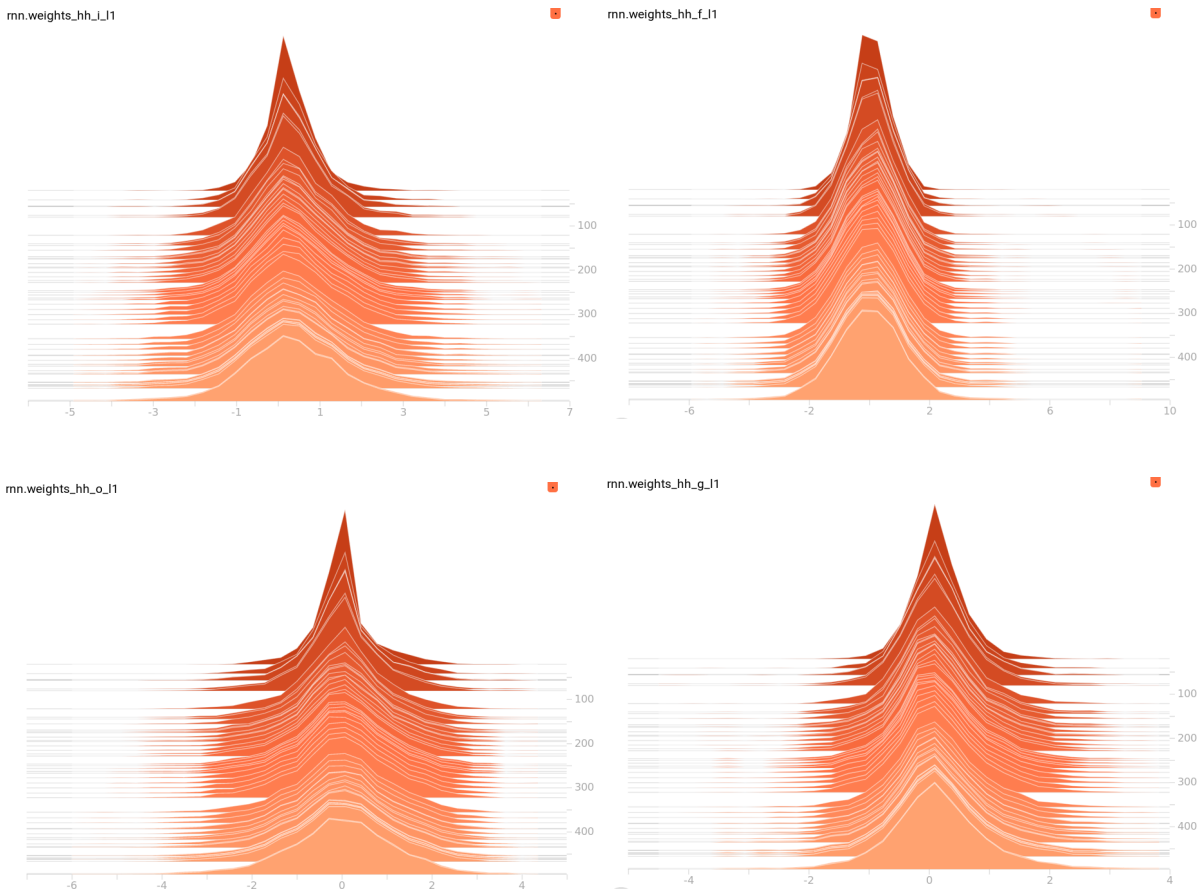

Fig. 5.8 Distribution during the training of the weights of the input h_t, upon the input (i_t), forget (f_t), output (o_t) gates and the cell state (g_t), respectively, for the $(2, 50)$-LSTM network trained on {HPCC 0.5, A, 15}

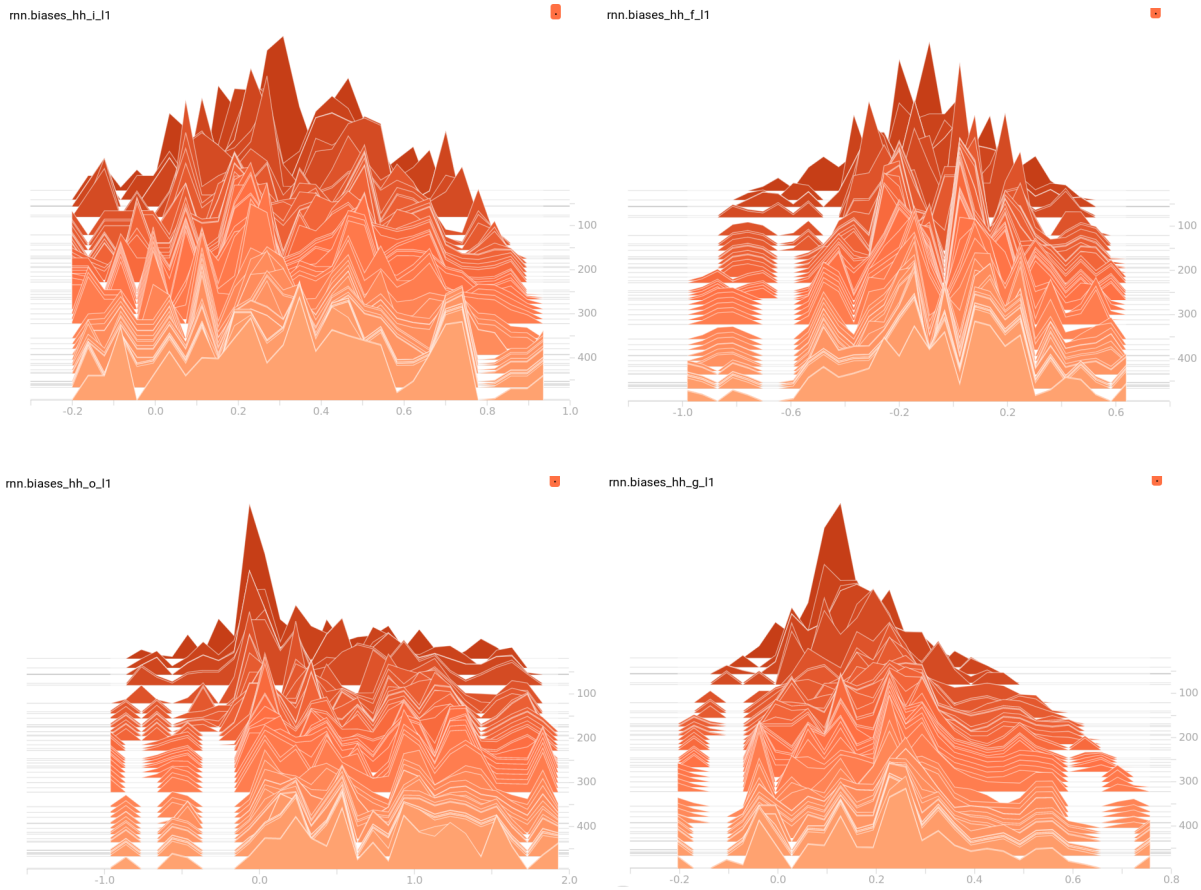

Fig. 5.9 Distribution during the training of the biases of the hidden state h_{t-1}, upon the input (i_t), forget (f_t), output (o_t) gates and the cell state (g_t), respectively, for the $(2, 50)$-LSTM network trained on {HPCC 0.5, A, 15}

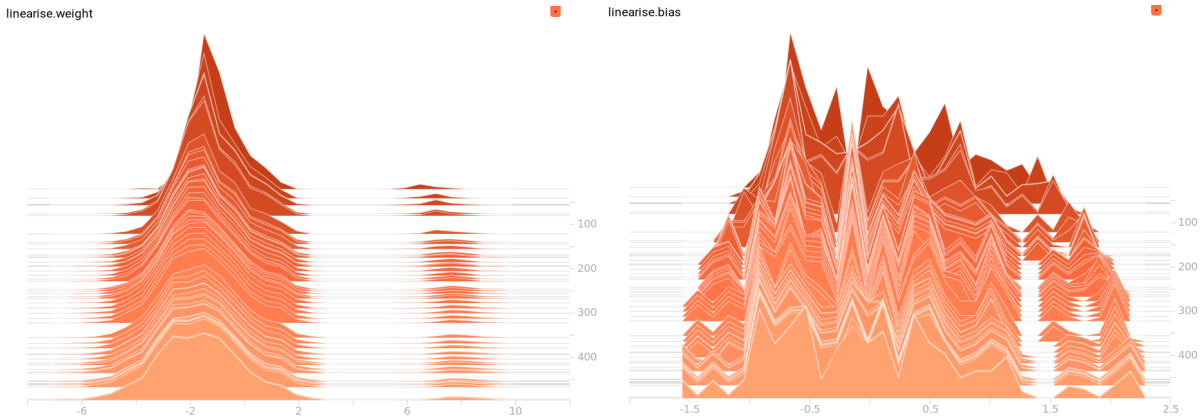

Fig. 5.10 Distribution during the training of the weights and biases respectively, of the sofmax layer (the last layer), for the $(2, 50)$-LSTM network trained on {HPCC 0.5, A, 15}

5.2.5 Simulation results

In this section the results of experiments with RAMPAGER 2-50 are presented and discussed. The focus is placed on the WI and Iav criteria, the former to evaluate the robustess of RAMPAGER 2-50, and the latter to acquire the information carried by the MI and QMI in an aggregated way.

The six topologies outlined in **Chapter 2** and the four usual numbers of agents, 5, 10 15 and 25, were experimented. 24 scenarios with RAMPAGER, for which one scenario gives rise to 100 runs, were therefore experimented, hence a total of 2400 runs. On the first three topologies, Islands, A and Grid, RAMPAGER is compared with the best RLPM variant. Then, on B, Circle and Corridor, it is only compared with CR, the decentralised representative, and HPCC 0.5, the model strategy, because experiments involving RLPM laid out in **Chapter 4** were not carried out on these topologies.

Figs. 5.11 to **5.22** show the performance according to the WI, QMI and Iav, on Islands, A, Grid, B, Circle and Corridor, respectively, of the RAMPAGER and the best RLPM variant, averaged over 100 runs for each scenario.

Globally, HPCC 0.5 is always the best strategy on both criteria, except on the Circle and Corridor topologies where HPCC is outperformed by CR on the Iav and WI for certain of configurations.

First, according to **Fig. 5.11** and **Fig. 5.12**, on Islands RAMPAGER is by far better than the best RLPM variant. On WI it remains outperformed by, but close to, CR, whereas according to Iav its performance is between CR and HPCC in term of performance.

Then, **Fig. 5.13** and **Fig. 5.14** show that on A, RAMPAGER is better than the best RLPM variant on both criteria, especially for 25 agents, and on the WI slightly worse than, but close to, CR, except for 15 agents where it is better. On the Iav it always outperforms CR.

On Grid, **Fig. 5.15** and **Fig. 5.16** highlight that RAMPAGER is better than the best RLPM variant. On the WI, it is close to CR for 5 and 10 agents, equal to it for 15 agents, and better than it for 25 agents. It can then be argued that RAMPAGER is globally equal to CR on the WI, with an average WI over the numbers of agents of 4313 for RAMPAGER against 4450 for CR. By doing so, on Grid RAMPAGER slightly outperforms CR in average on the WI. On the Iav, RAMPAGER outperforms both CR and the best RLPM variant, especially the latter for 25 agents.

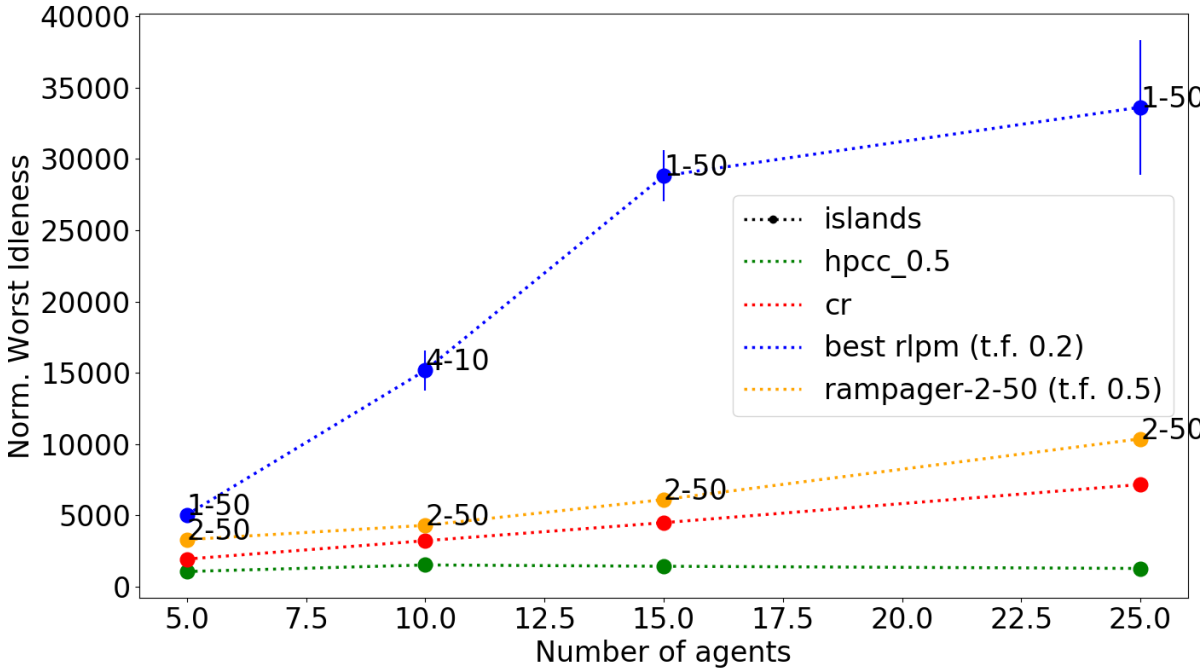

Fig. 5.11 Normalised WI, averaged over 100 execution, of the RAMPAGER 2-50 and the best variant of RLPM w.r.t. the number of agents on Islands.

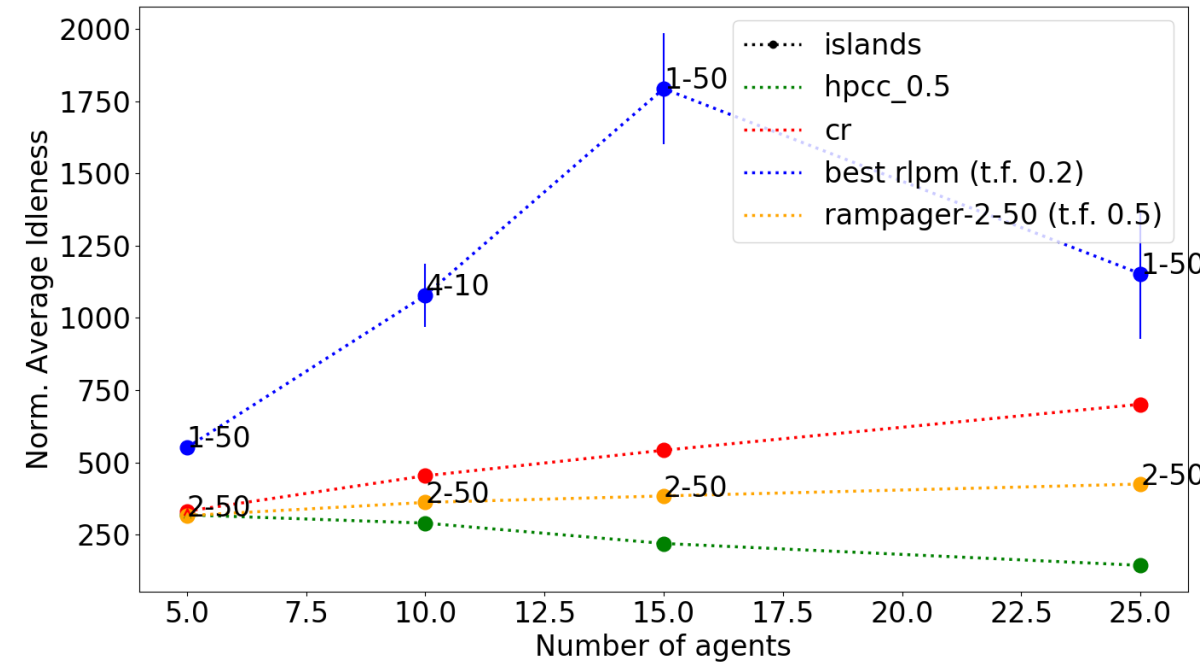

Fig. 5.12 Normalised Iav, averaged over 100 execution, of the RAMPAGER 2-50 and the best variant of RLPM w.r.t. the number of agents on Islands.

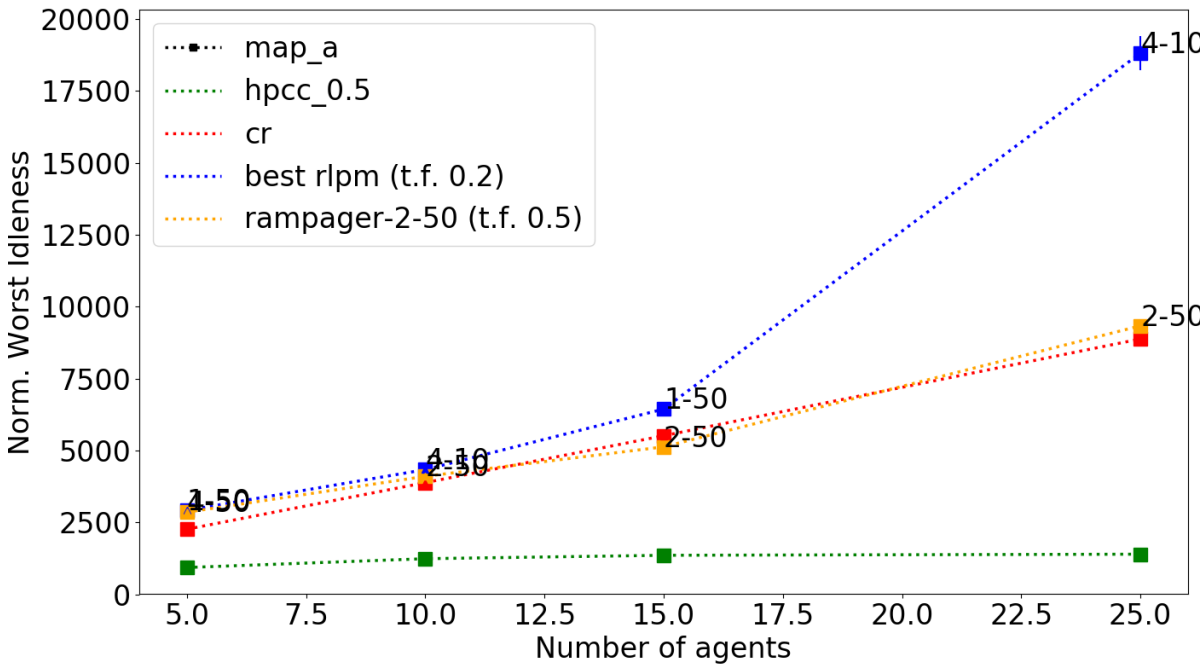

Fig. 5.13 Normalised WI, averaged over 100 execution, of the RAMPAGER 2-50 and the best variant of RLPM w.r.t. the number of agents on A.

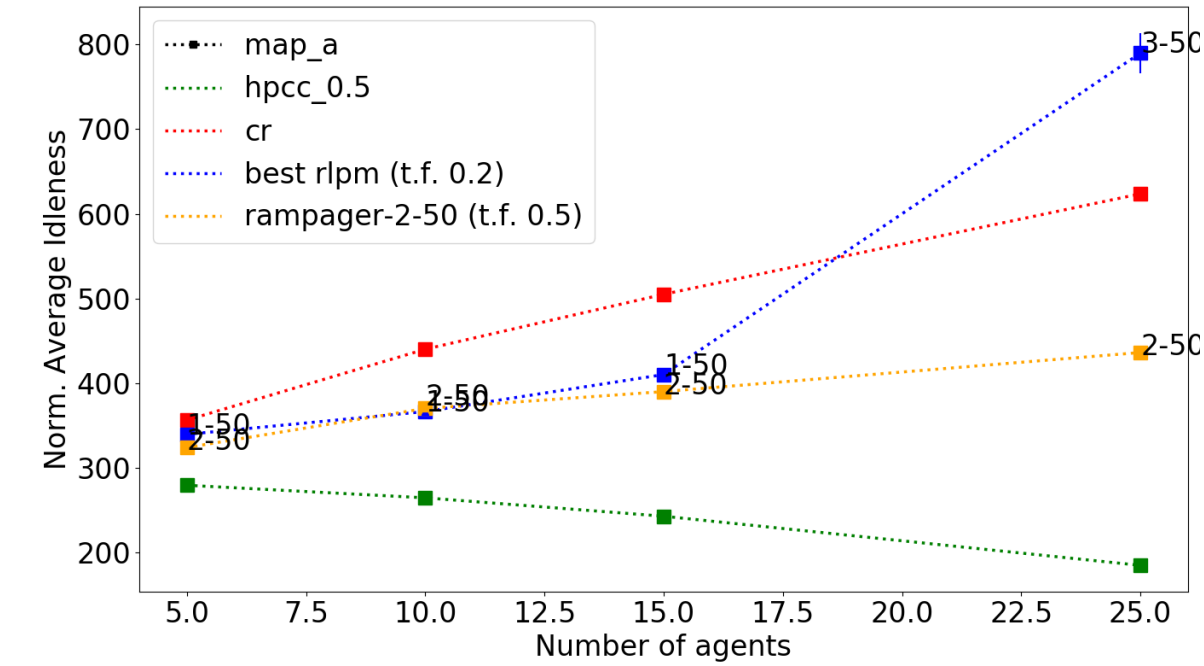

Fig. 5.14 Normalised Iav, averaged over 100 execution, of the RAMPAGER 2-50 and the best variant of RLPM w.r.t. the number of agents on A.

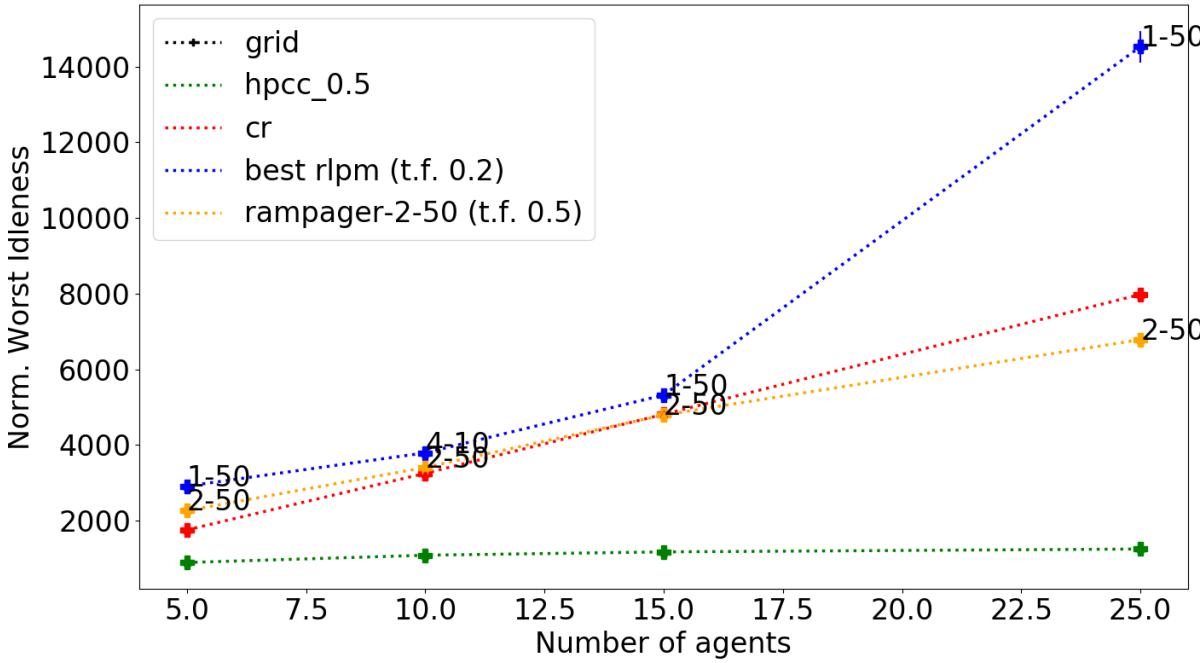

Fig. 5.15 Normalised WI, averaged over 100 execution, of the RAMPAGER 2-50 and the best variant of RLPM w.r.t. the number of agents on Grid.

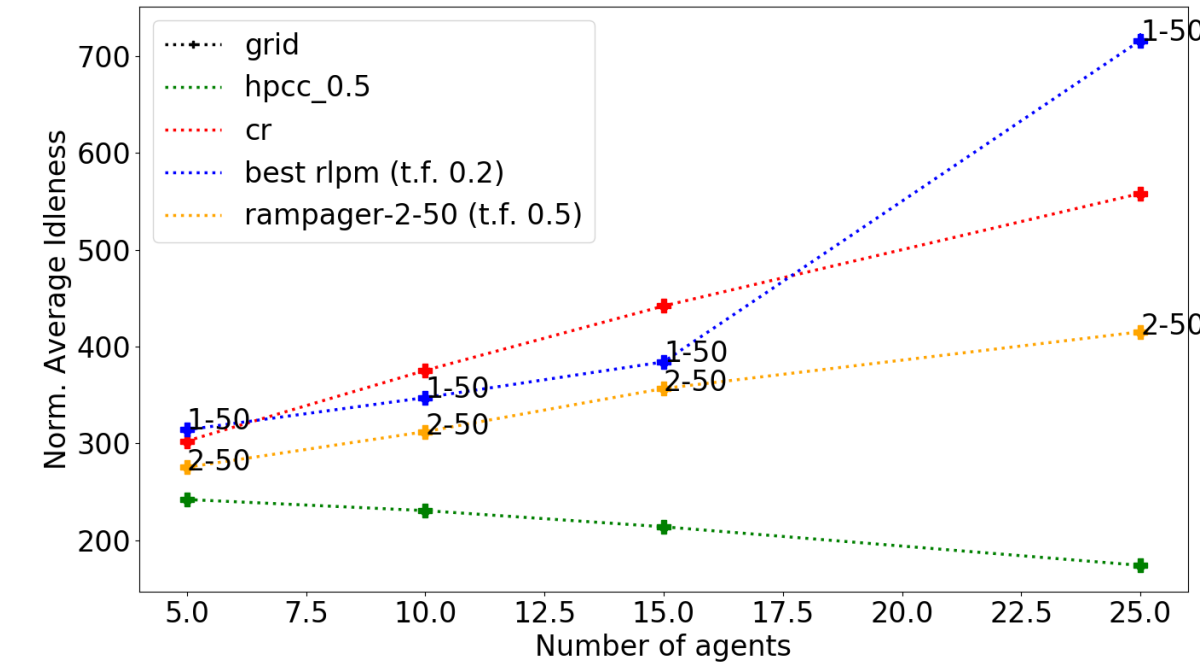

Fig. 5.16 Normalised Iav, averaged over 100 execution, of the RAMPAGER 2-50 and the best variant of RLPM w.r.t. the number of agents on Grid.

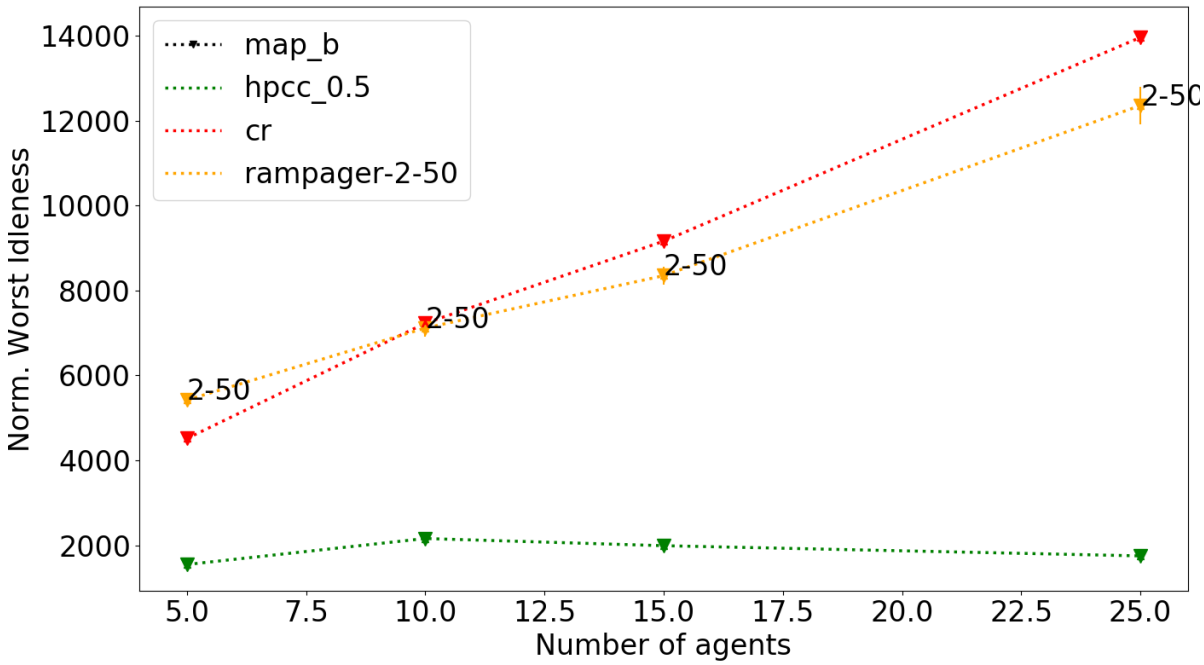

Fig. 5.17 Normalised WI of RAMPAGER 2-50 averaged over 100 execution w.r.t. the number of agents on B.

On B, **Fig. 5.17** and **Fig. 5.18** show that on the WI and Iav RAMPAGER outperforms CR, especially on the Iav, except for 5 agents where they are close.

On Circle, **Fig. 5.19** and **Fig. 5.20** show that CR is the best strategy. RAMPAGER is worse than CR on the WI and Iav, although it be better, by far, than HPCC 0.5, its model strategy, for 5 and 15 on the Iav, and approximately equal to it for 5 and 15 agents on the WI.

Finally, on Corridor, a graph with 49 vertices, **Fig. 5.21** and **Fig. 5.22** show that RAMPAGER is strongly outperformed by CR and HPCC 0.5 on the WI, although it outperforms HPCC 0.5 for 5 and 15 agents, and CR for all of the numbers of agents.

Note that for scenarios on Circle, {Corridor 5} and {Corridor 10}, learning from traces of HPCC 0.5 with the aim of developing a distributed strategy makes little sense considering the performance of HPCC 0.5 is worse than that of CR. Therefore, on Circle and Corridor, RAMPAGER is generally outperformed by CR or HPCC 0.5 on the WI and Iav, except for 5 and 10 agents where it outperforms HPCC 0.5 on the Iav.

It is worth noting that RAMPAGER has proven to be better than HPCC 0.5 on the Iav for some configurations — {Circle, 5}, {Circle, 10}, {Corridor, 5}, and {Corridor, 10} —, whereas it is, interestingly enough, the decision process which is aimed at capturing and reconstructing in a decentralised way by RAMPAGER. As stated above, a first

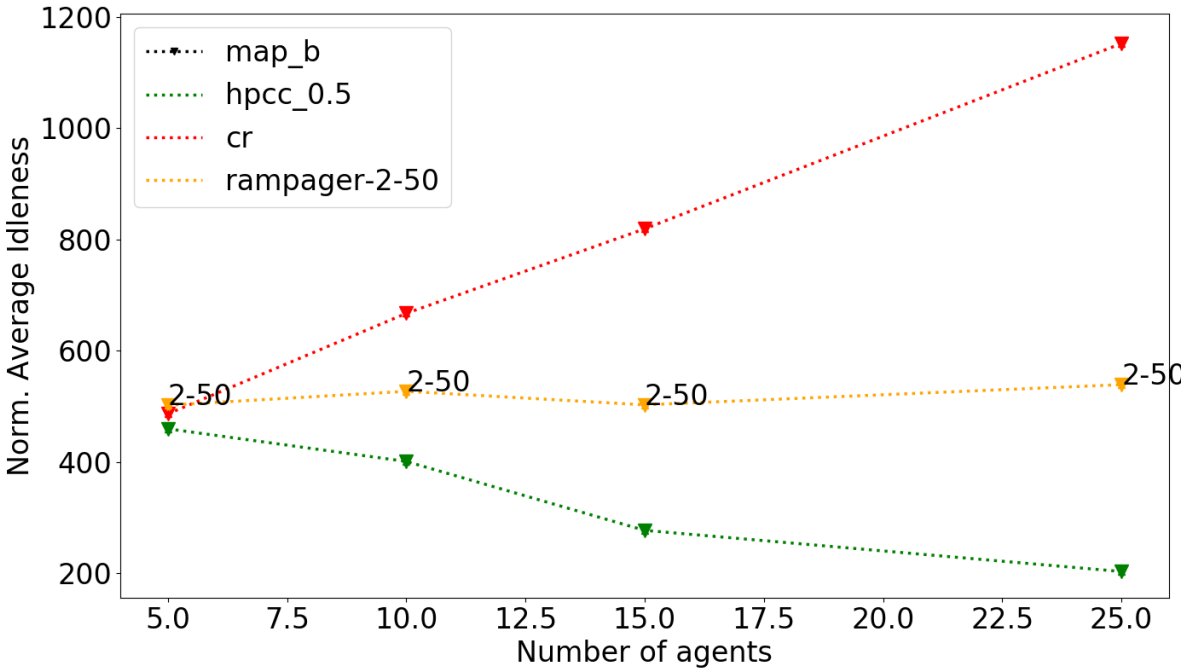

Fig. 5.18 Normalised Iav of RAMPAGER 2-50 averaged over 100 execution w.r.t. the number of agents on B.

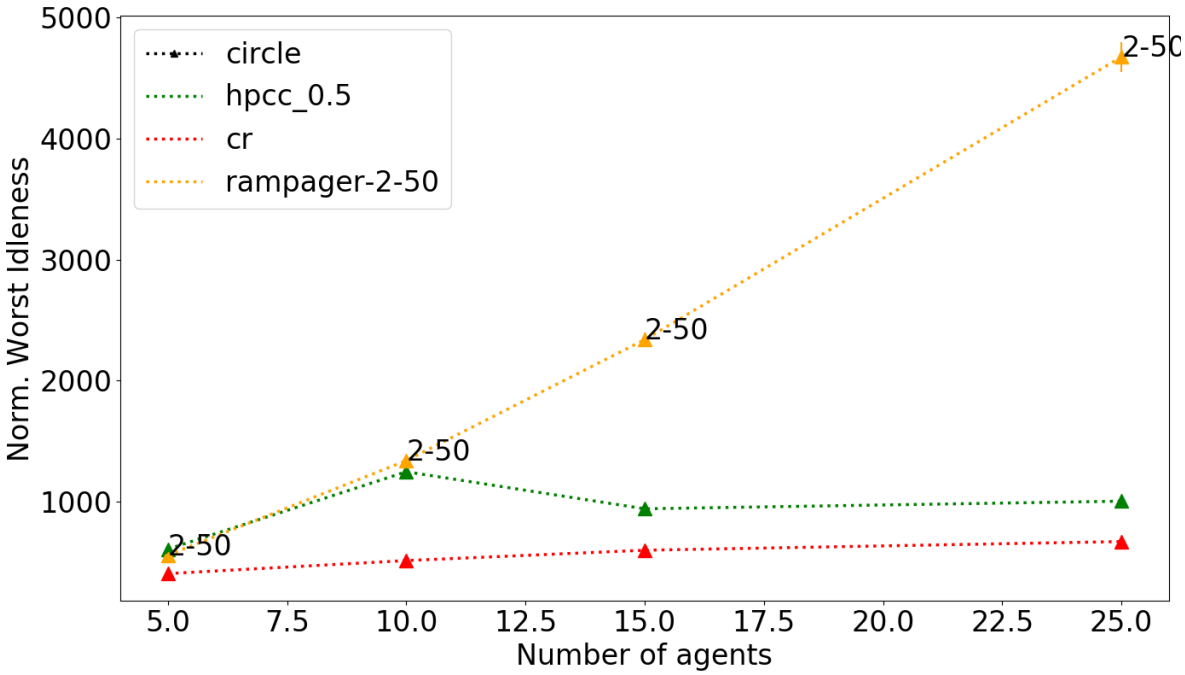

Fig. 5.19 Normalised WI of RAMPAGER 2-50 averaged over 100 execution w.r.t. the number of agents on Circle.

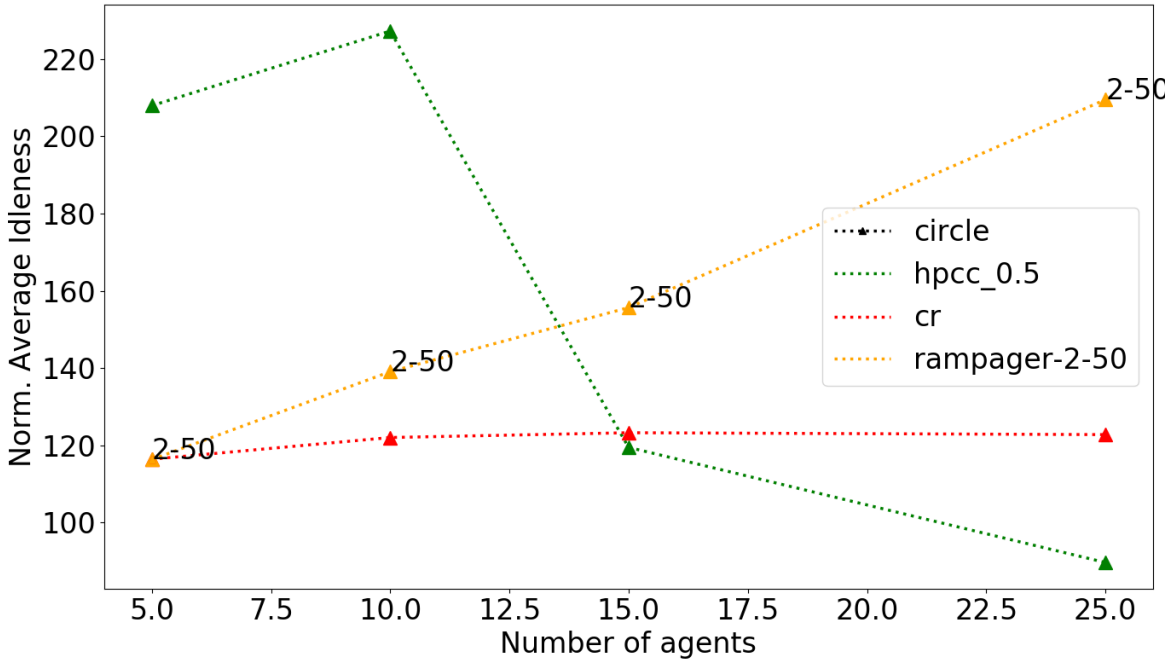

Fig. 5.20 Normalised Iav of RAMPAGER 2-50 averaged over 100 execution w.r.t. the number of agents on Circle.

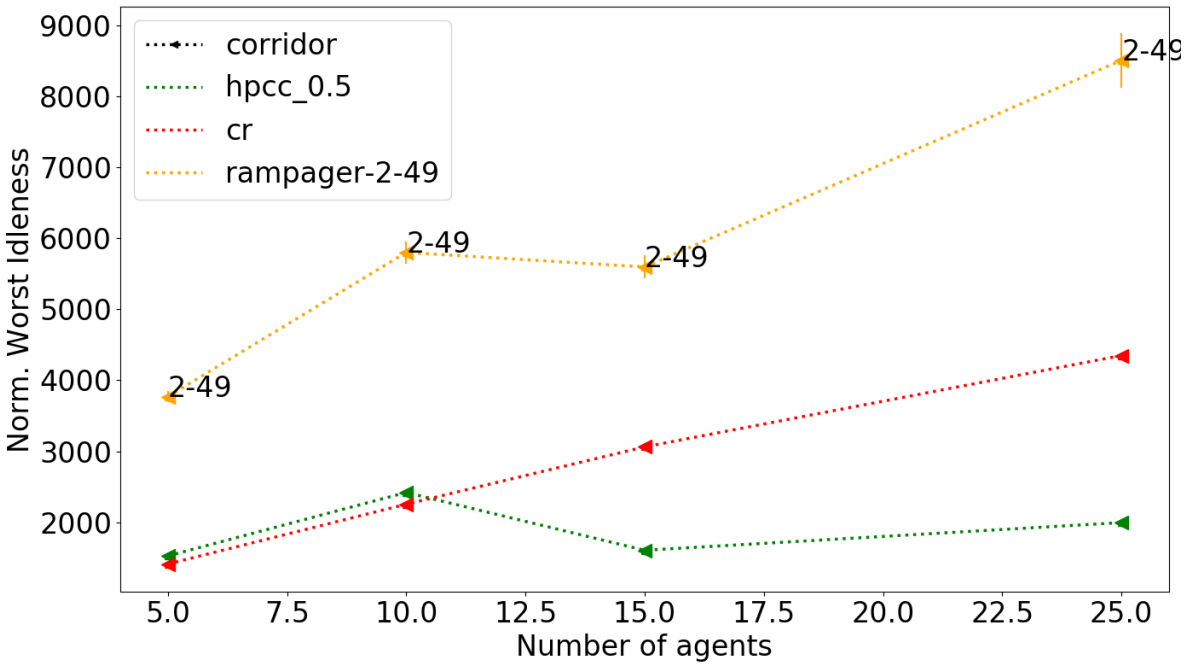

Fig. 5.21 Normalised WI of RAMPAGER 2-50 and over 100 execution w.r.t. the number of agents on Corridor.

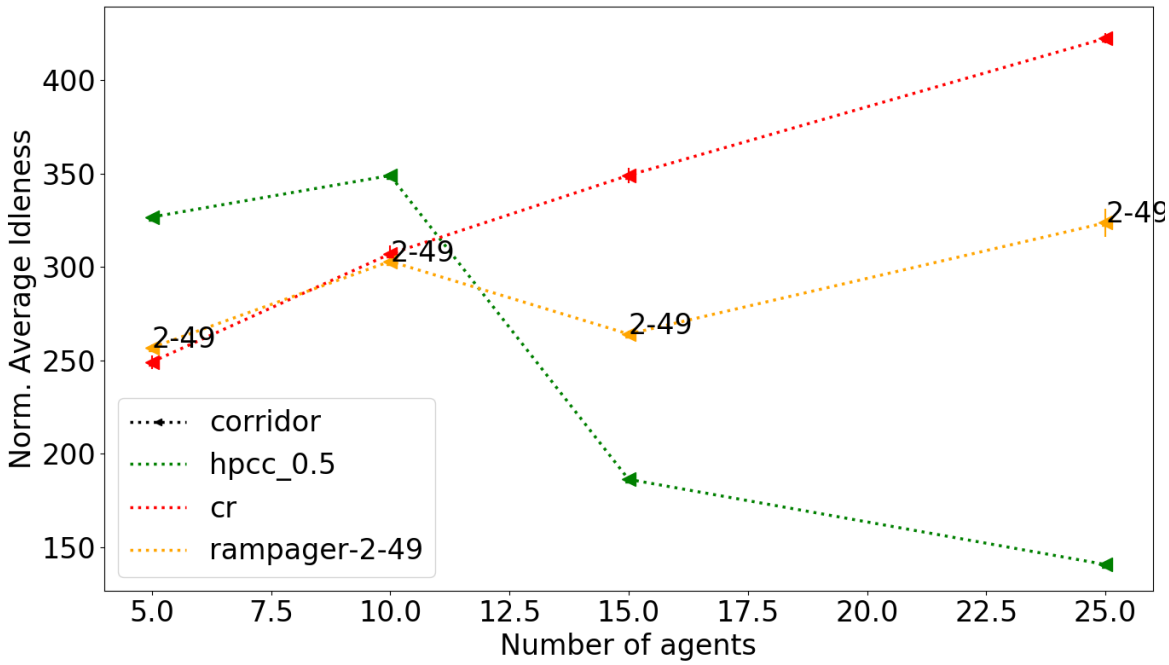

Fig. 5.22 Normalised Iav of RAMPAGER 2-50 and over 100 execution w.r.t. the number of agents on Corridor.

rationale is owing to the performance of HPCC 0.5, which has highlighted HPCC 0.5 is not robust for these scenarios. Further investigations have also shown that on Circle the QMI component of Iav in **Eq. 2.1.10** gives rise to this result, RAMPAGER has a lower QMI, whereas for Corridor it is the number of intervals N_J^G that is higher for HPCC 0.5. HPCC 0.5 is thus less robust than RAMPAGER on Circle, namely the agents are less evenly distributed over all of the nodes, whereas on Corridor the higher number of intervals of HPCC 0.5 for a better MI and QMI than RAMPAGER involves the paths taken by agents are less efficient.

5.3 Conclusion

This chapter has aimed at improving the RLPM strategy introduced in the previous chapter. The problem here was to integrate the structure of the topology to patrol in the LSTM networks used by agents to predict the next node to visit. Therefore, instead of the pretraining step carried out in the previous chapter, a new analytical procedure has been established. Following this analytical procedure enables initialising any LSTM network of same width as the studied topology's, with respect to its structure. This new analytical procedure, qualified as structure-guided initialisation, gave rise to the new

RAndom **M**uliagent **PA**trollin**G** LSTM-Path-Mak**ER** (RAMPAGER) strategy, which is nothing but an improvement of the RLPM strategy.

This procedure was then established formally according to the equations of the LSTM model, first with theoretical values, then with concrete values for its implementation in MAPTrainer. Afterwards, four LSTM networks initialised according to this new procedure were assessed in simulation. The best one, $2 - 50$, was retained to be assessed in all of the scenarios studied in this chapter, that is to say 24 scenarios with RAMPAGER 2-50.

Globally, RAMPAGER 2-50 outperforms to a large extent the best RLPM variant on both evaluation criteria. Thus, using HPCC 0.5 data, higher quality data in term of robustness, and initialising LSTM networks based on the structure of the problem, and more precisely here the graph to patrol, seem to have improved RLPM. Moreover, on the four first topologies, according to the WI RAMPAGER is generally equal to, or better than, CR, except on Islands.

Chapter 6

Idleness estimator: a decentralised strategy based on idleness estimation

The two previous chapters addressed the use of RNNs with the aim of learning paths of agents generated by a high-performance and centralised strategy. In the current chapter, the problem of using statistical models, and furthermore certain of ANN architectures, to estimate true idlenesses in real time during a patrolling mission is addressed. New generic strategies based on idleness estimation are defined. Several statistical models are studied and three of them are evaluated in simulation: the arithmetic mean, a linear-regression model and a MLP network. These models are trained from the HCC 0.2 database, then used by agents as part of the decision process as idleness estimators to choose the next node to visit according to the estimated true idlenesses. The main hypothesis of this application is that, if agents learn to reconstruct true idleness in average from individual idleness by using a statistical model, then they would be able to approach the performance reached by the model strategy.

The objective of this chapter being to define a new strategy based on machine learning models used to estimate the true idleness at each time step, in what follows the definition of such a strategy is established.

As part of MAP, a strategy based on an estimator to make a decision is named *Idleness Estimator* (IE): the decision process unfolds so that an estimate of the *true idleness* is first computed, then the decision regarding the next node to visit is made with respect to this estimate.

Each statistical model gives rise to two strategy types based on idleness estimation:

- a deterministic strategy type where idleness estimation is considered as such,
- a stochastic strategy type where idleness estimation is regarded as the average of a discrete probability distribution for the idleness to estimate.

In fact, the stochastic strategy type is an improvement of the deterministic one: random variations are introduced in the decision process to make the strategy more variable and robust.

Therefore, **Section 6.1** lays down the objective of this chapter, namely defining the procedure of idleness estimation in the framework of MAP, which thereby gives rise to the *generic type of strategy* mentioned above, that is IE. **Section 6.2** is devoted to the use of estimated true idleness by the agents to derive decisions. In **Section 6.3** three concrete examples of the new generic type of strategy introduced there are addressed. Then, in **Section 6.4** the training procedure of the estimators selected to be assessed is established; such a procedure is simple and straightforward. Finally, the strategies based on these estimators are experimented and evaluated in **Section 6.5**.

6.1 Idleness estimation

The *estimator* is noted $m(.,.)$, and is defined as follows:

According to the notations of **Section 3.1**, let $\forall t \in \mathbb{N}^*,\ \forall a \in A, i^a(t) = (i_1^a(t), ..., i_N^a(t)) \in (\mathbb{N}^*)^N$ be the vectors of individual idlenesses and $\hat{i}^{-a}(t) = (\hat{i}_1^{-a}(t), ..., \hat{i}_N^{-a}(t)) \in \mathbb{N}^{*N}$ be the corresponding vector of true idleness estimated by agent a, at time t. Then,

$$\forall t \in \mathbb{N}^*, \forall a \in A, \hat{i}^{-a}(t) = m(i^a(t), \boldsymbol{\theta}) \tag{6.1.1}$$

where $\boldsymbol{\theta}$ is the set of the estimator's parameters.

For any agent, such an estimator outputs an estimate of the current vector of true idlenesses, called *estimated idlenesses*, with respect to the current individual idlenesses provided as input to the estimator. All of the agents embed the same estimator, i.e. the same parametrised estimator. IE can be thought of as a reactive strategy using an artefact for estimating missing information concerning the area to patrol, and that takes into account the idleness of nodes and thereby implicitly the agents' positions. In this context, a temporal series representing the successive idlenesses each time an agent stands upon a node is called an *idleness sequence*. Also, by definition of the true

idleness, $\forall a \in A$ an agent, any individual idleness $i_v^a(t)$ of the node v must be mapped to an estimate of true idleness lower than or equal to the individual one as follows:

Let $i(t) = (i_1(t), ..., i_N(t))$ be the vector of true idleness at time t. Then, according to **Eq. 3.1.3** the following property holds:

$$\forall t \in \mathbb{N}^*, \forall v \in V, \forall a \in A, i_v(t) \in [0, i_v^a(t)] \tag{6.1.2}$$

However, this property does not necessary hold for $\hat{i}^{-a}(t)$. Hence the possibility to improve the estimation by bounding it, in a component-wise manner $\forall v \in V$, by the interval $[0, i_v^a(t)]$. This leads to the improved vector of estimated idleness $\hat{i}^a(t) = (\hat{i}_1^a(t), \ldots, \hat{i}_N^a(t))$ such as:

$$\forall t \in \mathbb{N}^*, \forall v \in V, \forall a \in A, \hat{i}_v^a(t) = \min(\ \max(\hat{i}_v^{-a}(t), 0),\ i_v^a(t)\ ) \tag{6.1.3}$$

This new estimator, termed as *enhanced estimator*, can be regarded as a procedure producing an estimation of true idlenesses corrected by the data available to the agent; such an estimation complies with the features of true idleness. In our context, this procedure is equivalent to apply **Eq. 6.1.1** and **6.1.3**. In the following, such an estimator will be simply termed as estimator.

For a given scenario, the corresponding estimator is parametrised in the most efficient way to predict the vector of true idleness with respect to its corresponding vector of individual idleness, this for any agent. To that end, each time an agent stands upon a node, it provides its vector of individual idleness to the estimator; in other terms, at each time step of next node choice, the agent estimates the vector of true idleness from its vector of individual idleness. Using an estimator as an information processing system, allows the agent to infer knowledge about its environment, and leads thereupon to the step of decision-making.

6.2 Decision-making based on idleness estimation

In the decision-making step, agents apply the two Heuristic and Pathfinder methods described in **Subsection 2.3.2** to their vector of estimated idleness: the Heuristic method is applied to select the next node to visit, whereas the Pathfinder method to choose the path to go there.

However, unlike HPCC, there is a slight difference for the Heuristic method used here: there is no mechanisms forbidding to select nodes already selected by other agents, due

to the absence of communication.

Let f be the ideal decision procedure of the model strategy, defined in **Subsection 4.1.1** and especially characterised by **Eq. 4.1.1**, and $\tilde{f}$ be the statistical-model-based decision procedure which is an approximated function of f using:

- $i^a(t)$, the vector of individual idleness of $a \in A$, instead of $i(t)$ the vector of true idleness,
- $v^a(t) \in V$ the node visited by a at time $t \in \mathbb{N}^*$,
- $v^a(t+\tau)$, the node resulting from the decision of a at t, where τ is the transit time between $v^a(t)$ and $v^a(t+\tau)$.

Then $\tilde{f}$ can then be characterised as follows:

$$\forall t \in \mathbb{N}^*,\ \forall a \in A,\ v^a(t+\tau) = \tilde{f}(v^a(t), i^a(t)) \tag{6.2.1}$$

This equation pertains to the formal definition of an idleness-estimator strategy: at one hand, each output depends upon both the current node and the individual idleness vector, and on the other hand, the final decision made according to $\tilde{f}$ corresponds to the next node to visit chosen with respect to the idleness estimation.

The decision procedure followed in the latter step can be more or less complex; it can be deterministic or stochastic.

6.2.1 Deterministic approach

In the deterministic approach, each agent is merely endowed with the same parametrised idleness estimator that they will use to estimate the true idleness vector throughout the mission to select the next node to visit, as shown in **Figure 6.1**. Such as in the two previous chapters, this strategy is meant to replicate, or at the very least to approach, the model strategy's behaviour in a decentralised way.

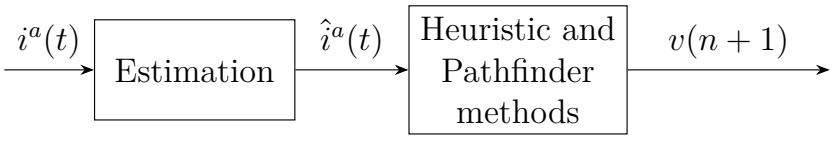

Fig. 6.1 Random Heuristic Pathfinder Idleness Estimator (HPIE) strategy, with $v(n)$ being the nth visited node.

On a side note, it is worth noting that CR can also be described by the equation **Eq. 6.2.1** using another function $\tilde{f}$. The difference between CR and any strategy using an estimation of true idleness is that the latter uses both static information about the structure of the topology and the average behaviour of other agents, whereas the former uses only local information, but dynamically, that is the individual idleness of its neighbour nodes. CR can thereby be considered as an idleness-estimator strategy where its estimation is simply its vector of individual idleness.

6.2.2 Drawback of the deterministic approach

An important drawback of the deterministic approach is that the relation between individual and true idleness is likely not a function, and therefore could not be learnt. This has led to state a theorem outlining the conditions under which such a relation is not a function:

Theorem 6.2.1. *Let $G = (V, E)$ be a graph, let A be a society of agents and consider two runs of a given strategy, arbitrarily named first and second run. If:*

- *in the initial state, a node $u \in V$ is occupied with an agent $a_1 \in A$ for the first run and with an agent $a_2 \in A$ for the second run, a_1 and a_2 can have the same agent identifier, and*
- *a next node $v \in V$ is selected by the strategy for the agent a_1 in the first run and for the agent a_2 in the second run and*
- *for the first run, it exists $w \in V, w \neq v$ that is occupied by an agent $a_3 \in A, a_3 \neq a_1, a_3 \neq a_2$ or that has already been reached by a_3 when, at time t, a_1 arrives at v and*
- *for the second run, no agents have reached w, when at time t, a_2 arrives at v,*

then the relation between the individual idleness $i^a(t)$ and the true idleness $i(t)$ is not a function, but a multivalued function.

Proof. At time t, the individual idlenesses for a_1 in the first run and for a_2 in the second run are equal: both have $i^a_j(t) = t, \forall j \neq v$ and $i^a_v(t) = 0$. For the first run $i_w(t)$, the true idleness of w, is equal to 0 if a_3 occupies it, or equal to $t - \tau < t$, where $\tau > 0$ is the travel time of agent a_3 from its initial position to w, otherwise. For the second run $i_w(t) = t$. Thus, to the same individual idleness corresponds two different values of the true idleness. Hence, the relation between the individual idleness $i^a(t)$ and the true idleness $i(t)$ is not a function. □

The conditions of **Theorem 6.2.1** are likely in the case of the graphs considered in this dissertation. Indeed, the analysis of a simple example of a 4-node graph with two islands and with 2 agents has shown that these conditions hold.

6.2.3 Stochastic approach

In order to make IE agents better distributed over the nodes of the graph to patrol, and by doing so increase the variability of the IE strategies, the entropy of estimated idlenesses must be increased to enhance the variability of estimated idleness. To that end, a supplementary step in the decision process is now added to IE as shown in **Figure 6.2**: for a given node $v \in V$ at time $t \in \mathbb{N}^*$, any IE agent $a \in A$, after making an estimate of the true idleness of v, noted $\hat{i}_v(t)$, considers this estimate as a mean of a random variable supported in the interval $[0, i_v^a(t)]$. A sample is then sampled according to this random variable to acquire a new datum, in fact a new idleness estimate used in Heuristic and Pathfinder methods. The resultant generic strategy type is named *random idleness estimator* (RIE).

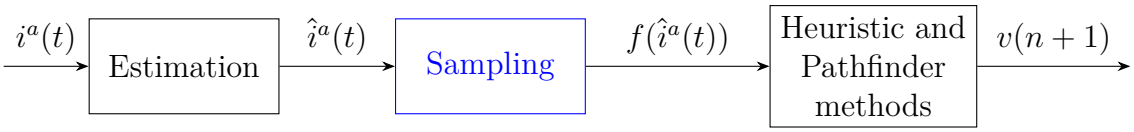

Fig. 6.2 Random Heuristic Pathfinder Idleness Estimator (RHPIE) strategy, with $v(n)$ being the nth visited node.

Such a probability distribution is now characterised.

6.2.3.1 Problem formulation

In order to sample the random variable, its probability distribution, and more precisely its mass function, shall be defined. The objective is to find the less specific mass function that presents the specified mean. A measure of *generality* of a mass function is its entropy as defined in **Appendix A**.

In all the remainder of this section, let:

- $m = \hat{i}_v^a(t)$, be $\forall a \in A$ the estimation of true idleness of v generated by the statistical model of a; it is here used as the expectation of the probability mass function of I,
- $i = i_v^a(t)$ be the individual idleness of $v \in V$, according to a, at time t.

According to **Eq. 3.1.3**, I is supported in $\mathcal{I} = \{0, 1, \ldots, i\}$ and its probability mass function is the family $\{P(I = 0), P(I = 1), \ldots, P(I = i)\}$. Hereinafter, $\forall k \in \mathcal{I}$, $p_k = P(I = k)$ and $p = (p_0, \ \ldots \ , p_i)$, such that $p \in [0, 1]^{i+1}$.

By definition of entropy, the highest entropy probability distribution must verify:

$$\max_{\{p_0, p_1, \ldots, p_i\}} \left(-\sum_{k=0}^{i} p_k \ \log(p_k) \right) \tag{6.2.2}$$

Eq. 6.2.2 can be rewritten as follows:

$$\min_{\{p_0, p_1, \ldots, p_i\}} \left(\sum_{k=0}^{i} p_k \ \log(p_k) \right) \tag{6.2.3}$$

The distribution maximising **Eq. 6.2.2** is the most entropic and the most uniform one. Indeed, the uniform distribution being that which maximises the entropy, making a phenomenon more entropic is equivalent to augment its *uniformness*, and homogenise the probabilities of its events. Moreover, what ensures that the entropy function has an optimum and can be maximised is that its Hessian is a negative definite matrix; the entropy is thus strictly concave, and by doing so, has a unique maximum. The Hessian has the following form:

$$H(\mathcal{H}) = \begin{bmatrix} -\frac{1}{p_0} & 0 & \ldots & 0 & 0 \\ 0 & -\frac{1}{p_1} & & & \\ \vdots & & \ddots & & \\ \vdots & & & \ddots & \\ 0 & 0 & \ldots & 0 & -\frac{1}{p_i} \end{bmatrix} \tag{6.2.4}$$

As previously stated, considering that p is a probability mass function, and I is a r.r.v. of expectation m, the two following constraints have to be satisfied:

$$\sum_{k=0}^{i} p_k = 1 \tag{6.2.5}$$

$$\sum_{k=0}^{i} k \ p_k = m \tag{6.2.6}$$

Therefore, **Eq. 6.2.3**, **Eq. 6.2.5** and **Eq. 6.2.6** make up the optimisation problem to solve for the purpose of obtaining the appropriate probability mass function, where **Eq.**

6.2.3 is the objective function, and **Eq. 6.2.5** and **Eq. 6.2.6** constitute the constraints of the problem, as follows:

$$\begin{aligned} &\min_{\{p_0,p_1,\dots,p_i\}} \left(\sum_{k=0}^{i} p_k \ \log(p_k) \right) \\ &\text{s.t.} \ \sum_{k=0}^{i} p_k = 1 \\ &\sum_{k=0}^{i} k \ p_k = m \\ &\forall k \in [|0,i|], p_k \geq 0 \end{aligned} \tag{6.2.7}$$

6.2.3.2 General solution

Now that the optimisation problem is formally stated, it will be solved using the method of Lagrange multipliers. To that end, its Lagrangian is first formulated. **Eq. 6.2.3** is the objective function, whereas **Eq. 6.2.5** and **Eq. 6.2.6** are the constraints for the method of Lagrange multipliers:

$$\begin{aligned} \forall p \in [0,1]^{i+1}, \forall \lambda = (\lambda_1, \lambda_2) \in \mathbb{R}, \ \mathcal{L}(p,\lambda) = \\ \sum_{k=0}^{i} p_k \ \log(p_k) + \\ \lambda_1 (\sum_{k=0}^{i} p_k - 1) + \\ \lambda_2 (\sum_{k=0}^{i} k \ p_k - m) \end{aligned} \tag{6.2.8}$$

First order conditions of non-linear programming state that the gradient of $\mathcal{L}$ w.r.t. p shall be equal to 0, which is given by:

$$\begin{aligned} &\forall p = (p_0, \ \dots \ , p_i) \in [0,1]^{i+1}, \ \forall k \in \mathcal{I}, \\ &\qquad \frac{\partial \mathcal{L}}{\partial p_k} = \log(p_k) + 1 + \lambda_1 + \lambda_2 \ k \\ &\forall p = (p_0, \ \dots \ , p_i) \in [0,1]^{i+1} \\ &\qquad \nabla_p \mathcal{L}(p,\lambda) = (\frac{\partial \mathcal{L}}{\partial p_0}, \ \dots \ , \frac{\partial \mathcal{L}}{\partial p_i}) \\ &\forall p = (p_0, \ \dots \ , p_i) \in [0,1]^{i+1} \\ &\qquad = (\log(p_0) + 1 + \lambda_1, \quad \dots \quad , \log(p_i) + 1 + \lambda_1 + \lambda_2 \ i) \end{aligned} \tag{6.2.9}$$

Thus:

$$\begin{aligned} & p = (p_0, \ \dots \ , p_i) \in [0,1]^{i+1}, \ \forall k \in \mathcal{I}, \\ & \qquad \frac{\partial \mathcal{L}}{\partial p_k} = 0 \\ \iff & p = (p_0, \ \dots \ , p_i) \in [0,1]^{i+1}, \ \forall k \in \mathcal{I}, \ \forall \lambda = (\lambda_1, \lambda_2) \in \mathbb{R} \\ & \qquad \log(p_k) + 1 + \lambda_1 + \lambda_2 \ k = 0 \\ \iff & p = (p_0, \ \dots \ , p_i) \in [0,1]^{i+1}, \ \forall k \in \mathcal{I}, \ \forall \lambda = (\lambda_1, \lambda_2) \in \mathbb{R} \\ & \qquad p_k = e^{-1-\lambda_1-k \ \lambda_2} \end{aligned} \tag{6.2.10}$$

Note that owing to the exponential, p_k is positive. Substituting p_k from **Eq. 6.2.10** into **Eq. 6.2.5** and **Eq. 6.2.6** provides a system of two equations with two unknowns: λ_1 and λ_2. This system can be solved and substitution of the solution in **Eq. 6.2.10** provides p_k for $k = 0, \dots, i$.

6.2.3.3 Formal relations and concrete expressions

These relations and expression are yielded by changing variables. A first change is given by **Eq. 6.2.11**:

$$\begin{aligned} & p = (p_0, \ \dots \ , p_i) \in [0,1]^{i+1}, \ \forall k \in \mathcal{I}, \ \forall \lambda = (\lambda_1, \lambda_2) \in \mathbb{R} \\ & \qquad p_k = \underbrace{e^{-1-\lambda_1}}_{=a} \ \underbrace{e^{-k\lambda_2}}_{=b^k: \ b=e^{-\lambda_2}} \end{aligned} \tag{6.2.11}$$

Finally:

$$\begin{aligned} & \forall k \in \mathcal{I}, \ \forall \lambda = (\lambda_1, \lambda_2) \in \mathbb{R} \\ & \qquad p_k = a \ b^k : \ a = e^{-1-\lambda_1}, \ b = e^{-\lambda_2} \end{aligned} \tag{6.2.12}$$

From **Eq. 6.2.12** and **Eq. 6.2.6**, it follows:

$$\begin{aligned} & \forall a, b \in \mathbb{R}, \ \sum_{k=0}^{i} k \ p_k = (\underbrace{\sum_{k=0}^{i} p_k}_{=1}) \times m \\ \iff & \forall a, b \in \mathbb{R}, \ \sum_{k=0}^{i} (k-m) \ p_k = 0 \\ \overset{6.2.12}{\iff} & \forall a, b \in \mathbb{R}, \ a \sum_{k=0}^{i} (k-m) \ b^k = 0 \end{aligned} \tag{6.2.13}$$

Then, according to **Eq. 6.2.5**:

$$\begin{aligned}
&\forall a, b \in \mathbb{R},\ \sum_{k=0}^{i} p_k = 1 \\
\overset{6.2.12}{\iff} &\forall a, b \in \mathbb{R},\ \sum_{k=0}^{i} a\ b^k = 1 \\
\iff &\forall a \in \mathbb{R}^*, b \in \mathbb{R},\ \sum_{k=0}^{i} b^k = \frac{1}{a} \\
\iff &\forall a \in \mathbb{R}^*, b \in \mathbb{R},\ (1 + b + \ \ldots\ + b^i) = \frac{1}{a}
\end{aligned} \tag{6.2.14}$$

Case 1: b = 1. $b = 1$ and **Eq. 6.2.14** imply[1]:

$$\begin{aligned}
\iff &\forall a \in \mathbb{R}^*,\ \sum_{k=0}^{i} 1 = \frac{1}{a} \\
\iff &\forall a \in \mathbb{R}^*,\ \frac{1}{a} = i + 1 \\
\iff &\forall a \in \mathbb{R}^*,\ a = \frac{1}{i+1}
\end{aligned} \tag{6.2.15}$$

which entails:

$$\forall k \in \mathcal{I},\ p_k = \frac{1}{i+1} \tag{6.2.16}$$

Therefore, $b = 1$ and **Eq. 6.2.13** imply:

$$\begin{aligned}
&\forall a \in \mathbb{R}^*,\ a \sum_{k=0}^{i} (k - m) = 0 \\
\iff &\forall a \in \mathbb{R}^*,\ \sum_{k=0}^{i} (k - m) = 0 \\
\iff &\forall a \in \mathbb{R}^*,\ \sum_{k=0}^{i} k = \sum_{k=0}^{i} m \\
\iff &\forall a \in \mathbb{R}^*,\ \frac{i+1}{2}\ i = (i+1)\ m \\
\iff &\forall a \in \mathbb{R}^*,\ m = \frac{i}{2}
\end{aligned} \tag{6.2.17}$$

Thus, when $b = 1$, or equivalently, $m = \frac{i}{2}$, the highest entropy probability distribution supported by $\mathcal{I}$, is the discrete uniform distribution, which is, in fact, undoubtedly the most entropic distribution, by definition of entropy.

[1] The demonstration is left to the reader.

Finally, in the case where $b = 1$:

Case 1: b = 1.

$$\begin{aligned} a &= \frac{1}{i+1} \\ b &= 1 \\ m &= \frac{i}{2} \end{aligned} \tag{6.2.18}$$

Case 2: $b \neq 1$**.** When $b \neq 1$, two possible cases must be distinguished: the case where $\mathbf{b < 1}$ and that where $\mathbf{b > 1}$.

Case 2.1: b < 1. First, because in this case the largest p_k is p_0, that is **Eq. 6.2.12** describes a decreasing exponential with respect to k, $b < 1$ and **Eq. 6.2.13** imply:

$$m < \frac{i}{2} \tag{6.2.19}$$

Thus, $b < 1$ corresponds to the case where $m < \frac{i}{2}$. Then, according to $b \neq 1$ and **Eq. 6.2.14**, it follows:

$$\begin{aligned} &\forall a \in \mathbb{R}^*, b \in \mathbb{R} : b \neq 1,\ (1 + b + \ \ldots\ + b^i) = \frac{1}{a} \\ &\forall a \in \mathbb{R}^*, b \in \mathbb{R} : b \neq 1,\ a = \frac{1-b}{1-b^{i+1}} \end{aligned} \tag{6.2.20}$$

Also, **Eq. 6.2.20** entails:

$$a \neq 0 \tag{6.2.21}$$

Then, $b < 1$ implies $a \neq 0$, and according to **Eq. 6.2.13**, it follows:

$$\forall b \in \mathbb{R},\ \sum_{k=0}^{i} (k - m)\ b^k = 0 \tag{6.2.22}$$

Finally, in the case where $b < 1$:

Case 2.1: b < 1, formal relations.

$$\begin{aligned} &\forall b \in \mathbb{R} : \ b < 1, \ a = \frac{1-b}{1-b^{i+1}} \\ &\forall b \in \mathbb{R}, \ b \text{ is such that: } \sum_{k=0}^{i} (k-m) \ b^k = 0 \\ &m < \frac{i}{2} \end{aligned} \tag{6.2.23}$$

Eq. 6.2.22 being a high degree polynomial, it can hardly be solved. The solution will then be approximated.

Case 2.2: b > 1. First, because in that case the largest p_k is p_i, that is **Eq. 6.2.12** describes an increasing exponential with respect to k, $b > 1$ and **Eq. 6.2.13** imply[2]:

$$m > \frac{i}{2} \tag{6.2.24}$$

Thus, $b > 1$ corresponds to the case where $m > \frac{i}{2}$.

According to **Eq. 6.2.22**, considering that determining a solution of b in $[0, 1]$ is more convenient than finding such a solution in $[1, +\infty]$, **Eq. 6.2.10** will now be rewritten so that determining a solution for b in $[1, +\infty]$, is equivalent to finding a b' in $[0, 1]$.

According to **Eq. 6.2.10**, $\forall k \in \mathcal{I}, \ p_k$ can be reformulated using the change of indices $(l = i - k)$, as follows:

$$\begin{aligned} &\iff p = (p_0, \ \dots \ , p_i) \in [0,1]^{i+1}, \ \forall k \in \mathcal{I}, \ \forall \lambda = (\lambda_1, \lambda_2) \in \mathbb{R} \\ &\qquad\qquad p_k = e^{-1-\lambda_1 - k \ \lambda_2} \\ &\overset{l=i-k}{\iff} p = (p_0, \ \dots \ , p_i) \in [0,1]^{i+1}, \ \forall l \in \mathcal{I}, \ \forall \lambda = (\lambda_1, \lambda_2) \in \mathbb{R} \\ &\qquad\qquad p_l = e^{-1-\lambda_1-(i-l)\lambda_2} \\ &\iff p = (p_0, \ \dots \ , p_i) \in [0,1]^{i+1}, \ \forall l \in \mathcal{I}, \ \forall \lambda = (\lambda_1, \lambda_2) \in \mathbb{R} \\ &\qquad\qquad p_l = \underbrace{e^{-1-\lambda_1}}_{=a} \ \underbrace{e^{-(i-l)\lambda_2}}_{=b'^{i-l}; \ b'=e^{-\lambda_2}} \end{aligned} \tag{6.2.25}$$

[2]The demonstration is left to the reader.

The change of indices ($l = i - k$) leaves the value of the sum identical. It also leaves the interval over which $(p_l)_{l \in \mathbb{N}^*}$ evolves, unchanged, and accordingly well indexed. Such a change of indices is then well defined. Finally:

$$\begin{aligned} &\forall k \in \mathcal{I},\ \forall \lambda = (\lambda_1, \lambda_2) \in \mathbb{R} \\ &\qquad p_k = a\ b'^{i-k} :\ a = e^{-1-\lambda_1},\ b' = e^{-\lambda_2} \end{aligned} \tag{6.2.26}$$

Thereupon, **Eq. 6.2.13** can be rewritten according to the new formulation of $p_l, \forall l \in \mathcal{I}$ (**Eq. 6.2.26**):

$$\begin{aligned} &\forall a, b' \in \mathbb{R},\ \sum_{k=0}^{i} k\ p_k = (\underbrace{\sum_{k=0}^{i} p_k}_{=1}) \times m \\ \iff &\forall a, b' \in \mathbb{R},\ \sum_{k=0}^{i} (k-m)\ p_k = 0 \\ \overset{6.2.26}{\iff} &\forall a, b \in \mathbb{R},\ a \sum_{k=0}^{i} (k-m)\ b'^{i-k} = 0 \end{aligned} \tag{6.2.27}$$

Likewise, **Eq. 6.2.14** and **Eq. 6.2.26** entail:

$$\begin{aligned} &\forall a, b' \in \mathbb{R},\ \sum_{k=0}^{i} p_k = 1 \\ \iff &\forall a, b' \in \mathbb{R},\ \sum_{k=0}^{i} a\ b'^{i-k} = 1 \\ \iff &\forall a \in \mathbb{R}^*, b' \in \mathbb{R},\ \sum_{k=0}^{i} b'^{i-k} = \frac{1}{a} \\ \iff &\forall a \in \mathbb{R}^*, b' \in \mathbb{R},\ (b'^{i} + b'^{i-1} + \ \ldots\ + 1) = \frac{1}{a} \\ \iff &\forall a \in \mathbb{R}^*, b' \in \mathbb{R} : b' \neq 1,\ a = \frac{1 - b'}{1 - b'^{i+1}} \end{aligned} \tag{6.2.28}$$

In both cases $\mathbf{b < 1}$ and $\mathbf{b > 1}$, the expression of a with respect to b and b' respectively, is identical. Therefore, $b > 1$ implies $a \neq 0$, and according to **Eq. 6.2.27**, it follows:

$$b' \in \mathbb{R},\ \sum_{k=0}^{i} (k-m)\ b'^{i-k} = 0 \tag{6.2.29}$$

Finally, in the case where $b > 1$:

Case 2.2: b > 1, formal relations.

$$\begin{aligned}
&\forall b' \in \mathbb{R} :\ b' < 1,\ a = \frac{1-b'}{1-b'^{i+1}} \\
&\forall b' \in \mathbb{R},\ b' \text{ is such that: } \sum_{k=0}^{i}(k-m)\ b'^{i-k} = 0 \\
&m > \frac{i}{2}
\end{aligned} \tag{6.2.30}$$

As for **b < 1**, **Eq. 6.2.29** being a high degree polynomial, it can hardly be solved. The solutions of **Eq. 6.2.22** and **Eq. 6.2.29** can be approximated.

6.2.3.4 Approximated solving

b will approximate i by $+\infty$ using series convergence.

Case 2.1: b < 1. **Eq. 6.2.22** can be approximated as follows:

$$\begin{aligned}
&\forall b \in \mathbb{R} :\ b < 1, && \sum_{k=0}^{i}(k-m)\ b^k = 0 \\
&\overset{i\to+\infty}{\sim} && \sum_{k=0}^{+\infty}(k-m)\ b^k = 0 \\
\iff\ &\forall b \in \mathbb{R} :\ b < 1, && \sum_{k=0}^{+\infty}(k-m)\ b^k = 0 \\
\iff\ &\forall b \in \mathbb{R} :\ b < 1, && m \sum_{k=0}^{+\infty} b^k = \sum_{k=0}^{+\infty} k\ b^k \\
\iff\ &\forall b \in \mathbb{R} :\ b < 1, && m\ \frac{1}{1-b} = \frac{b^2}{(1-b)^2} \\
&\overset{i\to+\infty}{\iff} && b = \frac{m}{1+m}
\end{aligned} \tag{6.2.31}$$

Finally, when $m < \frac{i}{2}$, i.e. $b < 1$, and i is large enough, $b \approx \frac{m}{1+m}$ is a good approximation.

Case 2.1: b < 1, concrete expressions.

$$\begin{aligned} b &= \frac{m}{1+m} \\ a &= \frac{1-b}{1-b^{i+1}} \\ m &< \frac{i}{2} \end{aligned} \tag{6.2.32}$$

Case 2.2: b > 1. **Eq. 6.2.29** can be approximated using series convergence, as previously, and the change of variables $l = i - k$, as follows:

$$\begin{aligned} & \forall b' \in \mathbb{R} :\ b' > 1, && \sum_{k=0}^{i} (k-m)\ b'^{i-k} = 0 \\ \overset{i \to +\infty}{\sim} & && \sum_{k=0}^{+\infty} (k-m)\ b'^{i-k} = 0 \\ \overset{l=i-k}{\iff} & \forall b' \in \mathbb{R} :\ b' < 1, && \sum_{l=0}^{+\infty} (i-m-l)\ b'^{l} = 0 \\ \iff & \forall b' \in \mathbb{R} :\ b' < 1, && (i-m) \sum_{l=0}^{+\infty} b'^{l} = \sum_{l=0}^{+\infty} l\ b'^{l} \\ \iff & \forall b' \in \mathbb{R} :\ b' < 1, && \frac{i-m}{1-b'} = \frac{b'}{(1-b')^2} \\ \iff & \forall b' \in \mathbb{R} :\ b' < 1, && (i-m)(1-b') = b' \\ \overset{i \to +\infty}{\iff} & && b' = \frac{i-m}{1+i-m} \end{aligned} \tag{6.2.33}$$

Finally, when $m > \frac{i}{2}$, i.e. $b > 1$ and $b' < 1$, and i is large enough, $b' \approx \frac{i-m}{1+i-m}$ is a good approximation.

Case 2.2: b > 1, concrete expressions.

$$\begin{aligned} b &= \frac{i-m}{1+i-m} \\ a &= \frac{1-b'}{1-b^{i+1}} \\ m &> \frac{i}{2} \end{aligned} \tag{6.2.34}$$

Based on the facts set out in all this section, at each decision step a patrolling idleness estimator agent $a \in A$, will draw randomly a new idleness $\hat{i}_v^{a,\sim}(t)$ for each node $v \in V$, according to the distribution $p = (p_0, \ \dots \ , p_i)$, such as $\forall k \in \mathcal{I}$:

- $p_k = \dfrac{1}{i+1}$, if $m = \dfrac{i}{2}$,
- $p_k = a\ b^k$, where a and b have as expressions those of **Eq. 6.2.32**, if $m < \dfrac{i}{2}$,
- $p_k = a\ b'^{i-k}$, where a and b' have as expressions those of **Eq. 6.2.34**, if $m > \dfrac{i}{2}$.

Although $\hat{i}_v^{a,\sim}(t)$ be estimated from $\hat{i}_v^a(t)$, if that does not lead to confusion, $\hat{i}_v^{a,\sim}(t)$ will siply be noted $\hat{i}_v^a(t)$ given that it constitutes the new estimate.

6.3 Some statistical models for idleness estimation

In this subsection, statistical models which give rise to three concrete idleness estimators are presented. In what follows, statistical models will be simply referred to as models.

As preliminary remark, **Eq. 6.1.1** indicates that for all of the models studied here, the input and output, both of dimension N, stand for the vector of individual idlenesses and the vector of estimated idlenesses, respectively.

The first model is simply the *mean*, which estimates, for each node the true idleness as being the average of all true idlenesses recorded on this node for a given MAP scenario, this for all logged MAP executions of this scenario, and for all time $t \in \mathbb{N}^*$. With such a model noted $Mean$, an agent $a \in A$ carries out the estimation of the true idlenesses at $t \in \mathbb{N}^*$ as follows:

$$\hat{i}^{-a}(t) = Mean(i^a(t), \boldsymbol{\theta}) = B, \ \boldsymbol{\theta} = \{B \in \mathcal{M}_{N\times 1}(\mathbb{R})\} \tag{6.3.1}$$

where $\hat{i}^{-a}(t)$ and $i^a(t)$ are vectors. Note that for this model the input i_t^a is not used.

Considering the Mean Squared Error (MSE) criterion, which is often used for learning as indicated in **Appendix A.3**, the mean is the best estimate of a constant model. In fact, when such a model is used as estimator of true idleness, the corresponding strategy is called *Heuristic Pathfinder Mean Estimator* (HPME) when estimated idleness is used by the strategy deterministically, and *Random Heuristic Pathfinder Mean Estimator* (RHPME) when it is used stochastically. $Mean$ accounts for a reference to be compared

with for strategies based on more complex estimators. Such a simple standard strategy enables assessing how effective are the latter. A learning strategy being worse than HPME or RHPME would have a learning ability, in the framework of MAP, poorer than the mean.

Then, when the estimator corresponds to a linear model noted *Lin* or *Linear*, the strategy is called *Heuristic Pathfinder Linear Estimator* (HPLE) when estimated idleness is used by the strategy deterministically, and *Random Heuristic Pathfinder Linear Estimator* (RHPLE) when it is used stochastically. For such a model, any agent $a \in A$ carries out the estimation of the true idlenesses at $t \in \mathbb{N}^*$ as follows:

$$\hat{i}^{-a}(t) = Lin(i^a(t), \boldsymbol{\theta}) = W \cdot i^a(t)^T + b^T,\ \boldsymbol{\theta} = \{W \in \mathcal{M}_{N\times N}(\mathbb{R}),\ b \in \mathbb{R}^N\} \tag{6.3.2}$$

with W and b being the model's matrix of weights and vector of biases, respectively. Training such a model corresponds to determine the W minimising a certain distance — distance which is the criterion to minimise — between $\hat{i}^{-a}(t)$ and $i(t)$.

A MLP composed with $H \in \mathbb{N}^*$ hidden ReLU layers, as described in **Appendix A**, is termed as *ReLU model* and abbreviated MLP_{ReLU}, while its corresponding strategy is called *Heuristic Pathfinder ReLU Estimator* (HPRE) when estimated idleness is used by the strategy deterministically, and *Random Heuristic Pathfinder ReLU Estimator* (RHPRE) when it is used stochastically. From such a model noted MLP^H_{ReLU}, any agent $a \in A$ carries out the estimation of the true idlenesses at $t \in \mathbb{N}^*$ as follows:

$$
\begin{aligned}
\hat{i}^{-a}(t) &= MLP^{H}_{ReLU}(i^a(t), \boldsymbol{\theta}) \\
&= W_{out} \cdot ReLU(\\
&\qquad W_H \cdot ReLU(\\
&\qquad\quad W_{H-1} \cdot ReLU(\\
&\qquad\qquad \dots \\
&\qquad\qquad\quad W^3 \cdot ReLU(\\
&\qquad\qquad\qquad W_2 \cdot ReLU(\ W_1 \cdot i^a(t)^T + b_1\ ) + b^2 \\
&\qquad\qquad\quad) + b^3 \\
&\qquad\qquad \dots \\
&\qquad\quad) + b^{H-1} \\
&\qquad) + b^H \\
&\quad) + b^{out}, \\
&\boldsymbol{\theta} = \{\forall h \in [|1, H|],\ \exists k_h \in \mathbb{N}^* \text{ with } k_1 = N : \forall h \in [|1, H-1|], \\
&\qquad W_h \in \mathcal{M}_{k_{h+1} \times k_h}(\mathbb{R}), W_{out} \in \mathcal{M}_{N \times k_H}(\mathbb{R}), b^h \in \mathbb{R}^{k_{h+1}}, b^{out} \in \mathbb{R}^N\}
\end{aligned}
\tag{6.3.3}
$$

with $ReLU$ being the element-wise ReLU activation, and $\forall h \in [|0, H|]$, W_h the weight matrix of layer h.

Finally, upon the training stage's completion, each idleness-estimator agent is endowed with the same parametrised model, i.e. $\boldsymbol{\theta}$ will be the same $\forall a \in A$. When the decision process is stochastic, the estimate of true idleness $\hat{i}^{-a}(t)$ is regarded as the mean of the probability distribution used to sample a new estimate of true idleness for each node $v \in V$.

A formal definition of the idleness-estimator strategy, as a generic strategy coming with many variants, having been laid down, as well as statistical models which have been introduced to be used as estimators, the training procedure of such models is detailed in the next section.

6.4 Training procedure

The training of the models is performed from the logged idleness sequences of any high-performance strategy f. Generally, high-performance strategies rely on communication

and centralised decision-making process. Also, from the foregoing, the true idlenesses used to train the model are generated by such a model strategy.

As mentioned in **Section 3.2**, for all configuration $\{G, N_a\}$, the model is trained over all of the couples of individual and true idleness sequences generated by a model strategy. Each execution gives rise to N_a couples composed of both an individual and true idleness sequences. By doing so, the model learns to output the most likely true idleness vector corresponding to the individual one provided as input to the model. Such a process can be thought of as *idleness reconstruction* and more generally *information reconstruction*.

These idleness sequences are then processed so as, for each agent, only the vectors of idleness corresponding to time steps when the agent stands upon a node are retained. In fact, idleness reconstruction aims at generating, throughout the mission, a sequence of vectors of on-vertex true idleness from the sequences of vectors of on-vertex individual idleness. This is performed by minimising a given objective function with respect to a model $m()$, given two sets of $S \in \mathbb{N}^*$ training sequences of length T_s: the set of on-vertex individual idleness sequences, noted I^{ind}, which stands for the input of the model in training, and that of on-vertex true idleness sequences, noted I^{tr}, standing for the output of the model. All sequences have the same length T_s, and each element of a sequence is of dimension N. I^{ind} and I^{tr} are then 3-dimensional tensors.

The MSE is then computed as follows:

$$MSE(I^{ind}, I^{tr}) = \frac{1}{S \times T_s \times N} \sum_{s=1}^{S} \sum_{n=1}^{T_s} || \ I^{tr}_{s\ n} - m(I^{tr}_{s\ n}, \boldsymbol{\theta}) \ ||_2^2 \tag{6.4.1}$$

where $\boldsymbol{\theta}$ stands for the model's parameters whose the dimension hinges upon the used model.

6.5 Experiments and results

In this section, the conduct of experiments is identical to that of the previous chapters apart from the used statistical models; the models trained and evaluated here are those described in **Section 6.1**. Therefore, this conduct is not recalled here. The same applies for the selection of the best HPCC variant: the evaluation criteria considered here are the same as in **Chapter 4**, that is the average idleness and the MI. Therefore, HCC 0.2 will then be the model strategy.

As previously the training settings and results are first detailed, then experiments and results are finally described and evaluated. A particular emphasis is placed on the training results which confirm that the conditions of **Theorem 6.2.1** concerning the relation mapping individual idleness to true idleness hold.

6.5.1 Training settings

As stated in **Subsection 4.3.1**, models were trained over data generated by 100 executions of the 12 selected MAP scenarios.

For each scenario 8 statistical models were trained: a mean model, a linear model, three MLPs with sigmoid units and three different ANNs with rectifier linear units (ReLU) as follows:

- a network with only 1 ReLU layer simply termed *ReLU*,
- a network with 1 hidden and an output ReLU layer, termed *ReLU Output* (ReLUO),
- an MLP with 1 ReLU layer, termed *ReLU MLP* or MLP_{ReLU}.

Three architectures of sigmoid MLPs are tested and evaluated:

- 1 layer with 50 units,
- 10 layers with 50 units,
- 2 layers with 1500 units.

To train all of these models, the database was divided into a training and validation database, leading to an apportionment of 80% for the training one and 20% for the validation one. The regression models, namely the linear and ANN models were trained using the full backward-propagation algorithm i.e. the whole training base was used to compute the gradient, while the mean model was trained by computing the mean of idlenesses over all of the idleness sequences for each node. The regression models were trained over between 10000 and 100000 epochs with respect to the model, with one model for each scenario, using the framework MAPTrainer introduced in **Chapter 3** and developped for this purpose. Regarding the initialisation, the identity matrix was used as the initial weight matrix for the linear model, whereas for the ANN models the weights and biases were initialised uniformly between -0.1 and 0.1. In each epoch, only one batch containing all the data of the training database is provided to the models. Also, it is worth noting that, from the statistical model's perspective, all of the sequences

used to train are independent, that is to say for a given scenario they are provided to the model regardless of the other sequences generated by the same execution: each sequence is independent from all the other sequences.

Finally, the MSE has been chosen as the objective function to optimise, and the standard gradient descent (GD) algorithm has been used to train the models, except for *Mean*. More precisely, GD is used to optimise the models according to the MSE, with different learning rates listed in **Table 6.1** which shows an overview of the training settings detailed here.

Model	**Sig. MLP**	**ReLU MLP**	**Lin.**	**ReLUO**	**ReLU**
Nb. of seqs.	$100 * N_a$	$100 * N_a$	$100 * N_a$	$100 * N_a$	$100 * N_a$
Train-val.	$80\% - 20\%$	$80\% - 20\%$	$80\% - 20\%$	$80\% - 20\%$	$80\% - 20\%$
Database	HCC 0.2	HCC 0.2	HCC 0.2	HCC 0.2	HCC 0.2
Batch size	Full	Full	Full	Full	Full
Learning rate	0.1	10^{-7}	10^{-7}	0.1	10^{-7}
Algorithm	GD	GD	GD	GD	GD
Nb. epochs	37500	37500	100000	10000	100000
Objective	MSE	MSE	MSE	MSE	MSE

Table 6.1 Overview of the training settings

6.5.2 Training results

For all the networks, except for the last tested sigmoid MLP, the number of units in each layer is equal to the number of nodes, that is 50. For each sigmoid MLP and for all the scenarios, the corresponding MSE values were by far the worst compared with the other statistical models. Besides, regardless of the size of the evaluated architecture, all of the trained sigmoid MLPs present the same cost. This leads to conclude that regardless of the size of the considered sigmoid MLP, the MSE is irreducible.

Fig. 6.3 shows, for each model, the MSE over all the database, that is to say over all of the training data and validation data, for each of the 12 scenarios here experimented, except the sigmoid MLPs which have not been depicted considering their poor performance previously described.

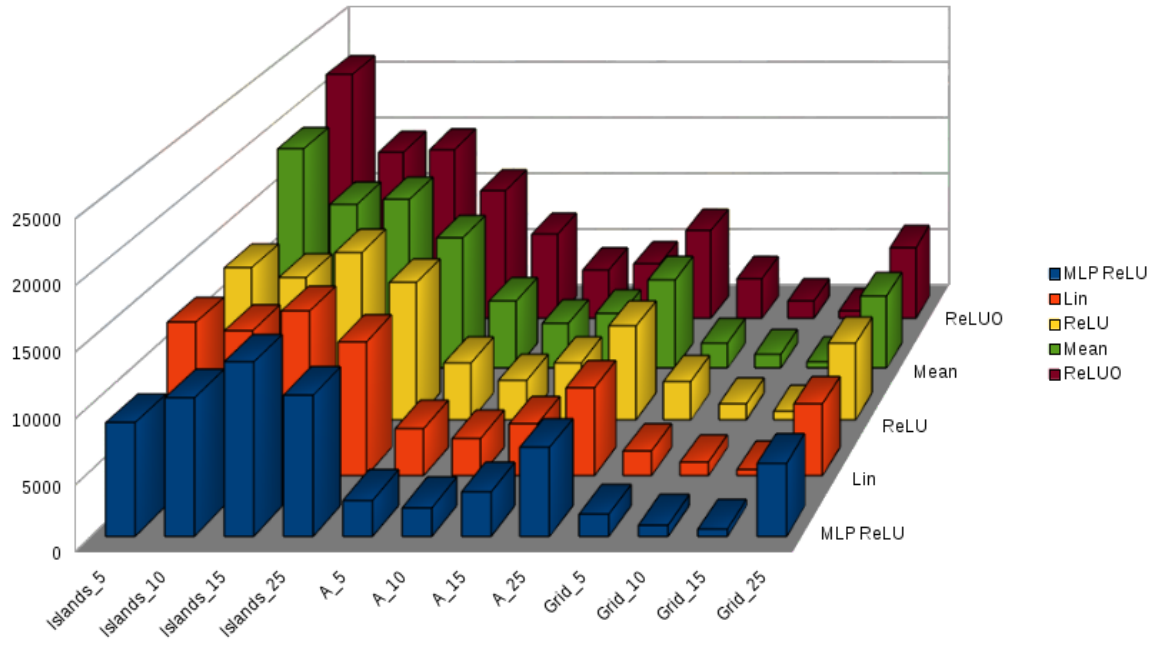

Fig. 6.3 Mean squared error (MSE) over all of the database for each scenario and model at the end of the training.

This figure shows that the best models are generally ReLU MLP and Linear. Globally, on the MSE, ReLU MLP is the best model, except for the scenarios $\{Islands, 15\}$, and $\{Islands, 25\}$, for which Linear is the best, and $\{Grid, 15\}$ for which Mean is the best. Regarding the model ReLU, i.e. one single ReLU layer, except for the map Islands where it is globally better than Mean, upon the other topologies it is worse than the latter. Finally, ReLUO, the model with an hidden layer and an output layer ReLU, is worse than Mean. Thus, amongst the three ReLU models, only ReLU MLP has been assessed in simulation.

Moreover, **Fig. 6.3** also highlights that for {Islands, 25}, MSE is quite high for all of the models. This is not full consistent with the approximation capability of neural networks: a neural network with at least one non-linear layer and one linear layer is able to approximate any function if the number of cells is large enough. These results seem to confirm that conditions of **Theorem 6.2.1** stated in **Subsection 6.2.2** are true and that the data do not represent a function.

Finally, only the 3 models Mean, Linear and ReLU MLP, have been experimented in simulation. These results will be analysed in the next section.

6.5.3 Simulation results

To evaluate their performance, HPME, HPLE and HPRE as well as their stochastic counterparts, namely RHPME, RHPLE and RHPRE, were trained on the HCC 0.2 database, then evaluated in simulation and compared with CR, the decentralised representative, and HCC 0.2, the centralised one from which they were therefore trained. The parameters r_H and r_P involving in the Heuristic and Pathfinder procedures have been set to 0.2, to be compared with HCC 0.2. Here the MI and QMI will be first considered studying the performance of the IE strategies.

IE strategies	MI	QMI
HPME 0.2	341	2990
HPLE 0.2	287	1065
HPRE 0.2	310	1109
RHPME 0.2	264	616
RHPLE 0.2	249	455
RHPRE 0.2	255	469
HCC 0.2	232	341
CR	362	584

Table 6.2 Normalised MI, QMI of the assessed IE strategies averaged over the A, Islands, and Grid topologies, and over 5, 10, 15 and 25 agents.

Table 6.2 that shows the performance of the IE strategies averaged over the 3 topologies and the 4 numbers of agents studied here, summarises the global trends on these evaluation criteria. The results highlight that the stochastic IE strategies are undoubtedly better than the deterministic ones, particularly on the QMI. Also, interestingly the worst RIE strategy is, at the very least, better than the best IE strategy on both criteria; this points out that the introduction of randomness improves undoubtedly the decision process of the IE generic strategy type. However, when compared with CR, the deterministic IEs are better than CR according to the MI, but by far worse than it on the QMI.

Fig. 6.5, **Fig. 6.4**, and **Fig. 6.6** show the best IE and RIE strategies, on the topologies *Islands*, *A* and *Grid*, respectively, according to the normalised MI.

Not surprisingly, HCC 0.2 always outperforms all of the other strategies, CR and the idleness estimators, on all the topologies, this for all of the numbers of agents.

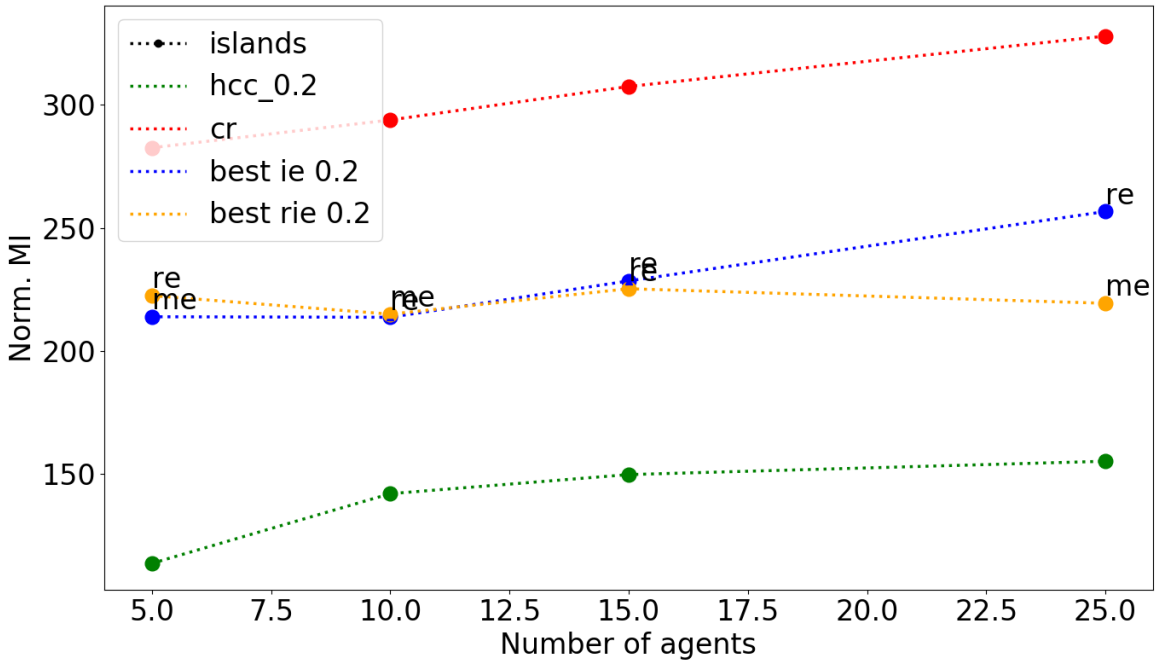

Fig. 6.4 Normalised MI, averaged over 100 runs, of the best IE and IRIE strategies on the Islands topology w.r.t. the number of agents, with *me* standing for *Mean*, *le* for *Lin* and *re* for *MLPReLU*.

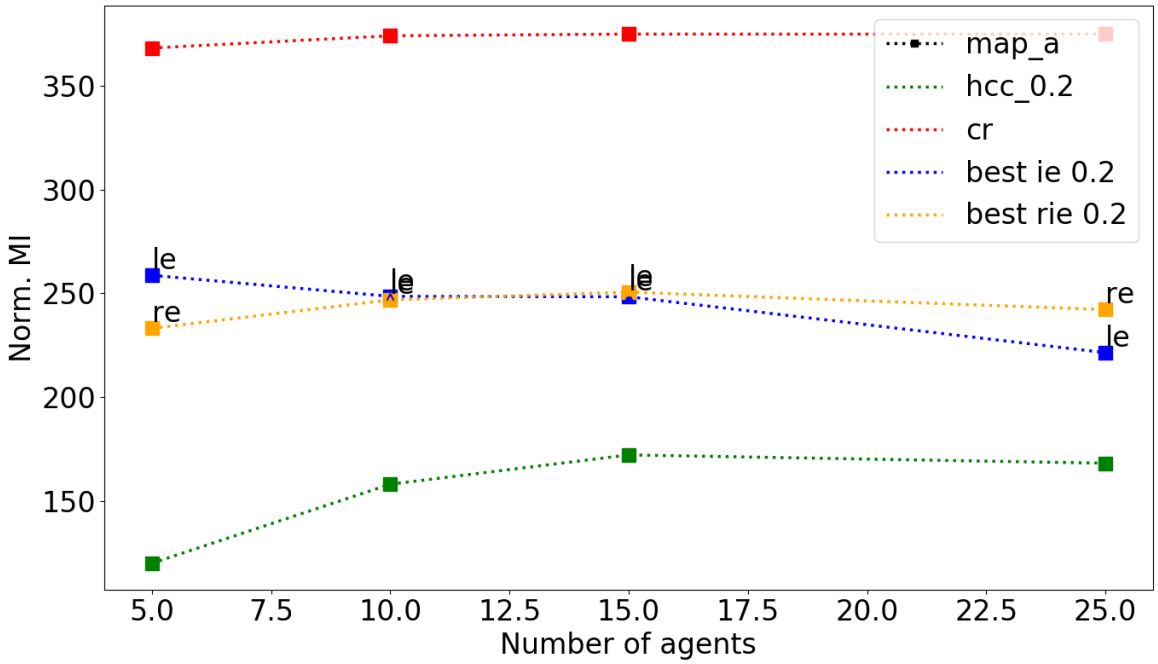

Fig. 6.5 Normalised MI, averaged over 100 runs, of the best IE and IRIE strategies on the A topology w.r.t. the number of agents, with *me* standing for *Mean*, *le* for *Lin* and *re* for *MLPReLU*.

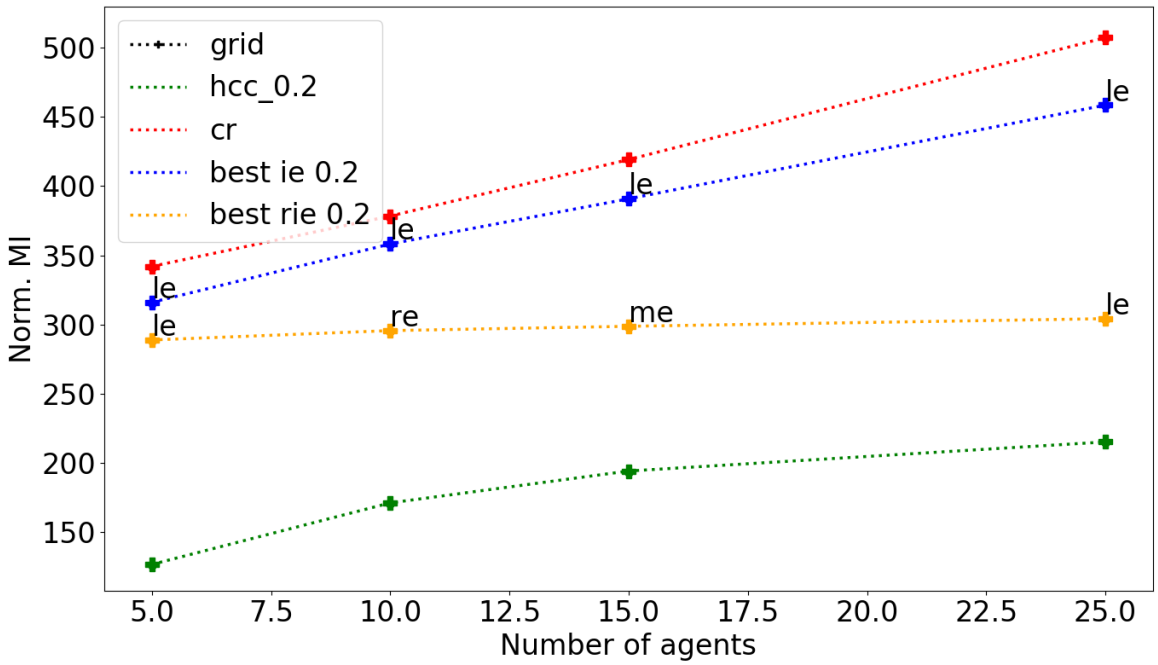

Fig. 6.6 Normalised MI, averaged over 100 runs, of the best IE and IRIE strategies on the Grid topology w.r.t. the number of agents, with *me* standing for *Mean*, *le* for *Lin* and *re* for $MLPReLU$.

According to the MI, all the IE strategies, deterministic or stochastic, outperform CR, the decentralised representative, on the three topologies.

On Islands and Grid, the best RIE strategy is globally better than the best IE and CR strategies. On A, the best IE strategy is worse than the best RIE strategy for 5 agents, it has approximately the same performance as the latter for 10 and 15 agents, and outperform the best RIE strategy for 25 agents. On Grid, the IE strategies show poor performance, particularly the best IE is barely better than CR.

Globally HPLE is the best strategy among the deterministic IE strategies, leading to consider *Lin* as the best estimator, whereas this trend is more balanced for the RIEs: the best statistical model depends on the considered scenario.

Finally, on this criterion, on the three topologies, and for the combined IE and RIE strategies, *Lin* is the best model in 42% of the cases, *Mean* in 33% of the cases, and $MLPReLU$ in 25% of the cases.

Fig. 6.8, **Fig. 6.7** and **Fig. 6.9** present the best IE and RIE strategies, on the topologies *Islands*, *A* and *Grid*, respectively, according to the normalised QMI.

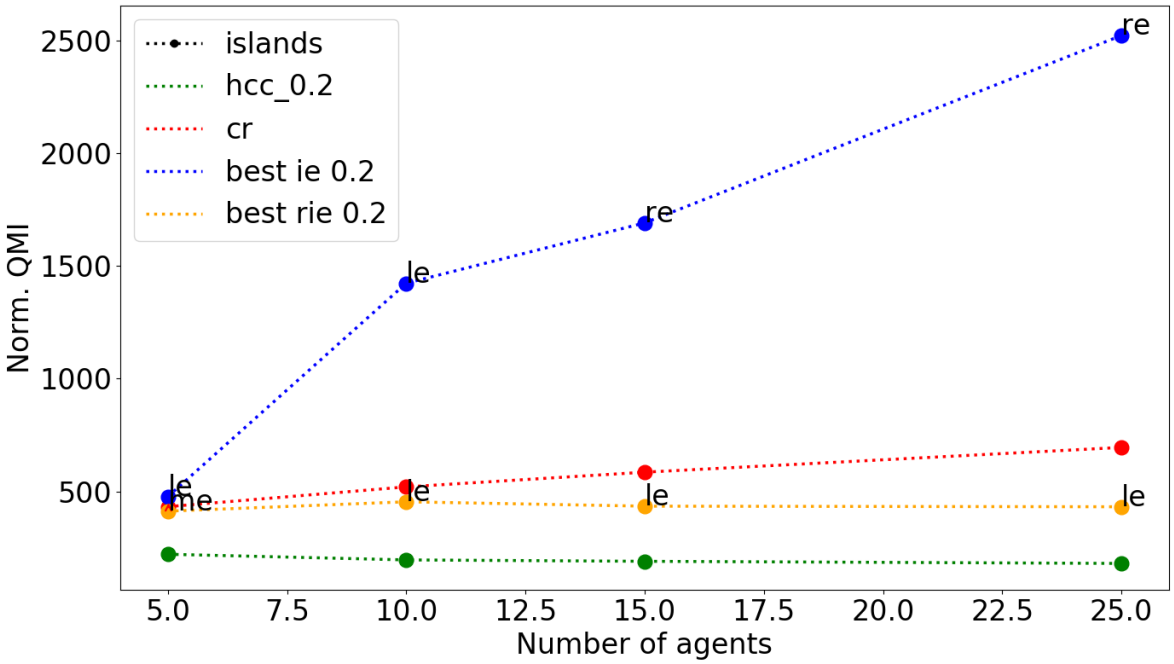

Fig. 6.7 Normalised QMI, averaged over 100 runs, of the best IE and IRIE strategies on the Islands topology w.r.t. the number of agents, with *me* standing for *Mean*, *le* for *Lin* and *re* for *MLPReLU*.

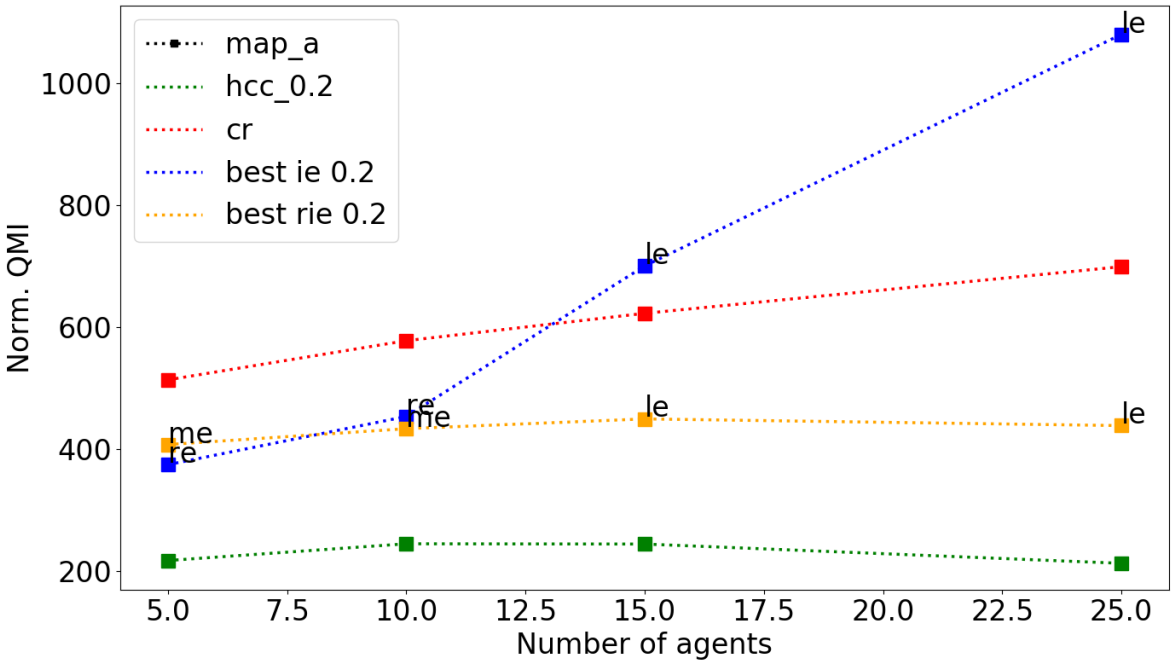

Fig. 6.8 Normalised QMI, averaged over 100 runs, of the best IE and IRIE strategies on the A topology w.r.t. the number of agents, with *me* standing for *Mean*, *le* for *Lin* and *re* for *MLPReLU*.

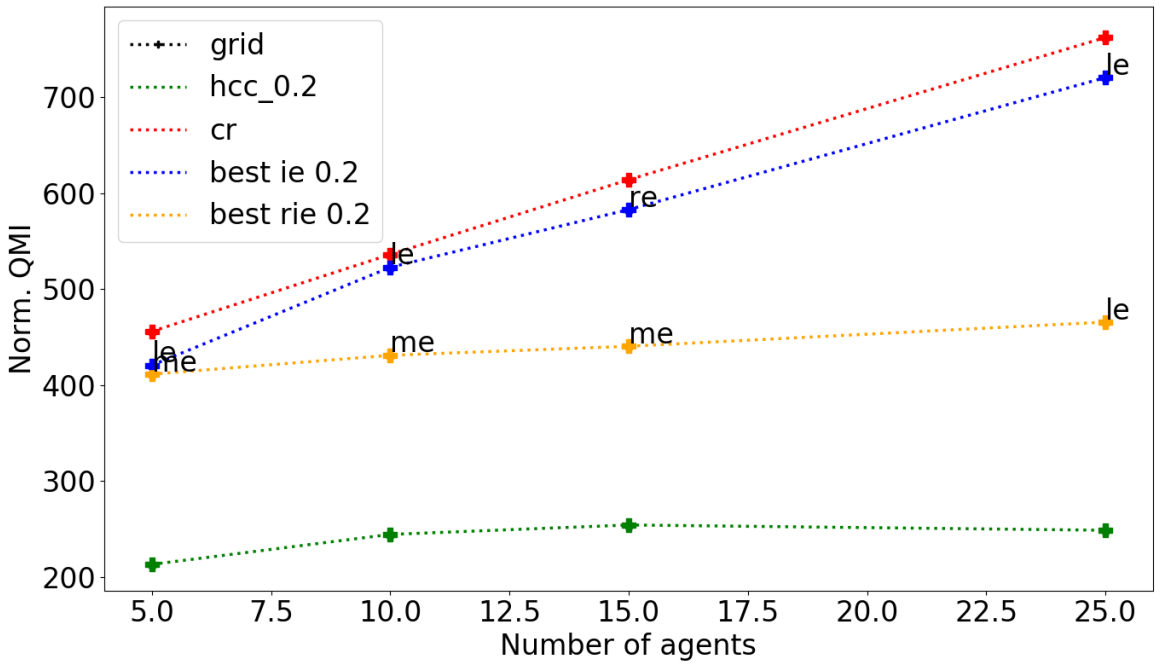

Fig. 6.9 Normalised QMI, averaged over 100 runs, of the best IE and IRIE strategies on the Grid topology w.r.t. the number of agents, with *me* standing for $Mean$, *le* for Lin and *re* for $MLPReLU$.

Globally, according to this criterion the deterministic IE strategies have poor performance, except for {A, 5} where HPRE is slightly better than RHPME, the best RIE strategy for this scenario. They are often worse than or equal to CR in terms of performance, especially on the Island topology on which they are overwhelmingly outperformed by CR.

With regard to the RIEs, they outperform all the time and to a large extent CR and the IE strategies on this criterion, and they constitute a realistic compromise as decentralised strategies — namely strategies that do not communicate — in terms of performance between CR and HCC 0.2.

Finally, on this criterion and on the three topologies, for the RIE strategies Lin is the best model in 66% of the cases and $Mean$ in 33% of the cases. Therefore, $MLPReLU$ is never the best model.

Thereupon, the results for the deterministic IE strategies show reasonable performance according to the criterion based upon the arithmetic mean, i.e. the MI, but they are offset by results that capture the equity of the strategy, hence the results on the QMI that justify the extension of the deterministic decision-making to the stochastic one; such

an extension gave rise to the RIE generic strategy type. In fact, these results showed the tendency of the deterministic IE agents to visit perpetually a particular set of nodes, set of nodes inferred by the statistical model, at the expense of the other ones, being thereby left unvisited. Such a tendency has been counteracted by the use of randomness in the decision process, and was justified by the performance of the RIE strategies. RIE is then an improvement of the IE generic strategy type.

6.6 Conclusion

A new MAP strategy based on machine learning and ANNs has been proposed in this chapter. This new type of strategy, qualified as IE, is based upon idleness estimation: agents are endowed with statistical models used as idleness estimator, which is a specific case of state estimator set out in **Section 3.2**; such statistical models are trained for the purpose of learning the relation mapping individual to true idlenesses. Two decision-making approaches were then laid down with regard to the manner idleness estimation is treated: a deterministic and a stochastic approaches. In the deterministic approach, idleness estimation is used as such, that is to say the idleness estimate is directly considered as the value of the node idleness to take into account to make the decision, whereas in the stochastic approach, for each estimated idleness this estimate is regarded as the average of a high-entropy probability distribution, and the final estimated idleness for a given node is drawn randomly according to this distribution. Such a stochastic approach gives rise to the RIE strategy type, which is also a generic type of MAP strategy.

The introduction of random drawing, and more generally of randomness, in the decision process enables, in fact, making the strategy more variable and more robust, in order to address the theoretical result established by **Theorem 6.2.1**, which states the relation to capture is not a function but rather a multivalued function.

The results presented in this chapter confirmed the theoretical result, which leads the statistical models to high errors. Indeed, the assessment of the deterministic IE strategies have highlighted that, globally, they have satisfactory performance on the MI, but poor performance on the QMI, which results from a poor distribution of visits over the nodes to monitor. Conversely, the RIE strategies have shown good performance, especially on the QMI, and have turned out to be high-performance decentralised strategies. Random variations, and more specifically random drawing of idleness, have thus offset the flaws highlighted by the IE strategies: the system has been made more robust by an inflow of randomness.

With regard to the problems raised by estimation of true idleness, different ANN architectures could be considered. For example, the low performance obtained by sigmoid functions might come from the numerical difficulty to approximate an identity relation for some high idleness values. An alternative way to use sigmoid functions could be to use element-wise multiplicative cells between the output of a sigmoid layer and the individual idlenesses. In that case, the sigmoid part of the network would learn the ratio between individual and true idlenesses instead of true idlenesses directly.

In the next chapter, interaction is introduced to turn the IE strategies, which are by definition decentralised strategies, into distributed strategies.

Chapter 7

Interacting Idleness Estimator: a strategy based on interaction

The introduction of randomness in the previous chapters has been a manner of taking into account an irreductible lack of information for an agent isolated from its society, isolation that has led to a lack of variability with regard to the distribution of the patrolling agents over the nodes. A stochastic model has then been established to cope with the lack of variability in a system composed of IE agents: the variability of visits over the nodes to monitor was increased by augmenting the complexity of the IE decision process. In such a stochastic approach, the estimation of true idleness is the outcome of a sample relying upon a high-entropy probability distribution. Another way to increase the complexity of a system is also to augment the interactions between its components. By doing so, in this chapter interaction is introduced as part of the IE strategy to take advantage of the ability to share information between agents throughout the patrolling mission, turning thereby the IE strategy into a distributed strategy, according to the definition of a distributed strategy stated in **Subsection 3.1.2**. Particularly, the RIEs being the best IE strategies, i.e. better than the deterministic IE strategies, the *interaction scheme* exposed in this chapter is integrated to and experimented with RIE agents.

Therefore, in **Section 7.1**, an interaction scheme is introduced to allow for agents to share information while patrolling, then in **Section 7.2** the resultant strategies are trained on both databases presented in **Section 3.5**, experimented in simulation, and finally the results are discussed.

7.1 Interacting Idleness Estimator

Interaction is the cornerstone of complex systems, and more specifically of MAS. Therefore, in MAP, which is, by definition, a MAS, an interaction scheme has been formalised and integrated in the IE strategies. This scheme can be set up both for the random IEs (RIE) and the deterministic ones (IEs). When it is activated for an IE strategy, the resultant strategy is regarded as an *interacting idleness estimator* (IIE), whereas with an RIE strategy it is considered as an *interacting random idleness estimator* (IRIE). Moreover, in our case, communication is the support of interaction. It is assumed that the communication mechanism is reliable, i.e. messages sent are received without modification and no messages are lost. It is now formally described.

7.1.1 Peer-to-peer interaction

Let a_1 and a_2 be two IE agents in A, able to communicate within a communication range noted r. Let, $\forall t \in \mathbb{N}^*$, $i^{+a_1}(t) = (i_1^{+a_1}(t), \ldots, i_N^{+a_1}(t))$ and $i^{+a_2}(t) = (i_1^{+a_2}(t), \ldots, i_N^{+a_2}(t))$, be their *shared idlenesses*, respectively. Here, they maintain two estimations of idleness:

- their individual idleness vector $i^a(t)$, $\forall a \in A$,
- a shared idleness vector $i^{+a}(t)$, resulting from the interaction with the agents they communicated with.

At the begining of the run, the vector of shared idleness is initialised so that it is equal to the vector of individual idleness: $i^{+a}(t) = i^a(t)$. At each time step it is incremented in the same way as the vector of individual idleness, namely by 1. Then, when a_1 and a_2 are at a distance lower than r, i.e. $d(a_1, a_2) < r$, $\forall d$ a distance on the graph G, they come into an *interaction context*. In such an interaction context, they are able to communicate and exchange information, leading to sending to each other their respective vector of shared idlenesses. Finally, each one of a_1 and a_2 agents applies the following rule to update their respective vector of shared idlenesses $i^{+a_1}(t)$ and $i^{+a_2}(t)$:

$$\begin{aligned} &\forall a \in \{a_1, a_2\}, \forall t \in \mathbb{N}^*, \ \forall v \in V, \\ &\qquad i_v^{+a}(t) \leftarrow \min(i_v^{+a_1}(t), i_v^{+a_2}(t)) \end{aligned} \tag{7.1.1}$$

When two agents apply the previous rule exchanging only they their respective vector of individual idleness, such agents are said to be in a *peer-to-peer interaction*.

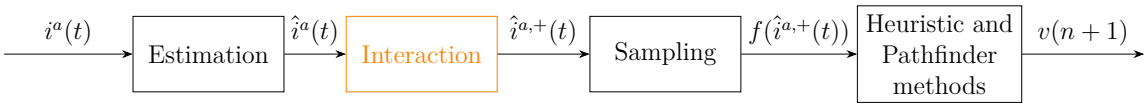

Fig. 7.1 Interacting Random Heuristic Pathfinder Idleness Estimator (IRHPIE) strategy, with $v(n)$ being the nth visited node.

7.1.2 Transitive interaction

Now, let R_I be the *interaction relation*, where R_I is a binary relation which denotes whether two agents can interact, i.e. whether $\forall a_1, a_2 \in A$, $d(a_1, a_2) < r \in \mathbb{R}^+$, a given parameter. Now, consider pairs of agents which are not in communication range, but which can communicate and interact in a peer-to-peer manner owing to the fact that other agents act as communication relays; in other terms there exists a path between them in the underlying graph of the binary relation R_I. Electronic communication being instantaenous and real-time, in a real patrolling mission with vehicles embedding a publish-subscribe connectivity framework on-board, such as *Data Distribution Service* (DDS), this type of communication is practicable. These agents are then able to communicate as if they were in the same communication range. Formally, this has been implemented in simulation using the transitive closure of R_I. Also, R_I is symmetric, that is to say $\forall a_1, a_2 \in A$, $a_1 R_I a_2 \implies a_2 R_I a_1$, or in other words, if a_1 can interact with a_2 then a_2 can interact with a_1.

This enhanced interaction, called *transitive interaction*, and noted R_I^{trans}, is defined as follows:

$$\begin{aligned} & \forall a, b \in A,\ a\ R_I^{trans}\ b \\ \iff & \forall a, b \in A,\ \exists n \in [|1, N_a - 1|] : \exists (a_1,\ \dots\ , a_n) \in A^n, a_n = b,\ \text{and} \\ & a R_I a_1 R_I\ \dots\ R_I a_n = b \end{aligned} \tag{7.1.2}$$

Thus, during a patrolling mission, at each time step $t \in \mathbb{N}^*$, for all agent $a, b \in A$ such as $a\ R_I^{trans}\ b$, a and b are able to interact. Then, each agent $a \in A$ applie the following rule to update its vector of shared idlenesses $i^{+a}(t)$:

$$\begin{aligned} \forall a \in A,\ \forall t \in \mathbb{N}^*, \forall v \in V,\ \exists n \in [|1, N_a - 1|] : a R_I^{trans} a_1,\ \dots\ , a R_I^{trans} a_n,\ \text{and} \\ i_v^{+a}(t) \leftarrow \min(i_v^a(t), i_v^{+a_1}(t), i_v^{+a_2}(t),\ \dots\ , i_v^{+a_n}(t)) \end{aligned} \tag{7.1.3}$$

Finally, when an IE strategy sets up this interaction scheme, the resultant strategy type is named *interacting idleness estimator* (IIE) when the decision process is deterministic, and *interacting random idleness estimator* (IRIE) when it is stochastic; the decision procedure of the latter is depicted in **Figure 7.1**. In these new interacting strategies, the enhanced estimation defined in **Eq. 6.1.3** corrects the initial estimation using the shared idleness instead of the individual one as follows:

$$\forall t \in \mathbb{N}^*, \forall v \in V, \forall a \in A, \hat{i}_v^a(t) = \min(\ \max(\hat{i}_v^{-a}(t), 0),\ i_v^{+a}(t)\ ) \tag{7.1.4}$$

When the statistical model used by an IIE agent to estimate idlenesses is:

- $Mean$, the resultant strategy is called *Interacting Heuristic Pathfinder Mean Estimator* (IHPME)
- Lin, the resultant strategy is called *Interacting Heuristic Pathfinder Linear Estimator* (IHPLE)
- MLP_{ReLU}, the resultant strategy is called *Interacting Heuristic Pathfinder ReLU Estimator* (IHPRE)

Likewise, when the statistical model used by an IRIE agent to estimate idlenesses is:

- $Mean$, the resultant strategy is called *Interacting Heuristic Pathfinder Mean Estimator* (IRHPME)
- Lin, the resultant strategy is called *Interacting Heuristic Pathfinder Linear Estimator* (IRHPLE)
- MLP_{ReLU}, the resultant strategy is called *Interacting Heuristic Pathfinder ReLU Estimator* (IRHPRE)

As stated previously, it is worth noting that the individual idleness vector $i^a(t)$, and not the shared one, is used in the IE algorithms by the statistical model to estimate the true idleness vector, given that statistical models are trained to reconstruct the true idleness vector from the individual one. As shown in **Eq. 7.1.4** the shared idleness $i_v^{+a}(t)$ is then used to correct $\hat{i}_v^{-a}(t)$, the estimate of true idleness.

7.2 Experiments and results

In this section, as in the previous chapter, the training settings are first detailed, then simulation experiments are analysed and evaluated.

The conduct of experiments is identical to that outlined in the previous chapter, except that the models were also trained on the HPCC 0.5 database. This conduct is therefore not reminded. The two model strategies, HCC 0.2 and HPCC 0.5, are then here used. Moreover, machine learning models trained from HCC 0.2 being the same as those used in the previous chapter, only HPCC 0.5 training settings and results are exposed in this section. Note that each time an arbitrage must be done with respect to a setting, e.g. an ANN architecture, a parameter, or a method, experiments are performed in the regular configuration {A, 15} to select the appropriate setting.

Regarding the training settings, the same statistical models as in the previous chapter, namely MLP_{ReLU}, Lin and $Mean$, are trained over 10000 sequences generated by HPCC 0.5 for each of the 24 scenarios. They were trained using the Adagrad algorithm with a learning rate of 0.1, over between 400 and 20000 epochs with respect to the model and scenario experimented.

7.2.1 Training results on HPCC 0.5 data

Fig. 7.2 and **Fig. 7.3** show, for each model, the MSE cost in validation of the trained models on Islands, A and Grid, and on B, Circle and Corridor, respectively, for each scenario experimented here.

Fig. 7.2 shows that on Islands, A and Grid, the three statistical models have approximately the same validation cost, except for {Islands, 5} where $Mean$ is the best model by almost 800, {Islands, 25} where it is better by approximately 1000, and {A, 15} and {Grid, 5} where its MSE is approximately twice as less than the two other models. In average $Mean$ is the best model on Islands (8876), whereas Lin and MLP_{ReLU} are the best ones on A (6569) and Grid (5708), respectively. In average over all of the three topologies MLP_{ReLU} and Lin are the best models with the same performance of 7120 against 7901 for $Mean$. Finally, MLP_{ReLU} and Lin are the best statistical models, even though for some scenarios $Mean$ is by far the best one.

Training performance on B, Circle and Corridor depicted in **Fig. 7.3** show that Lin is the best model on B, with an average value of 6853, MLP_{ReLU} is the best model on Circle with an average of 1415, whereas $Mean$ is the best one on Corridor with an

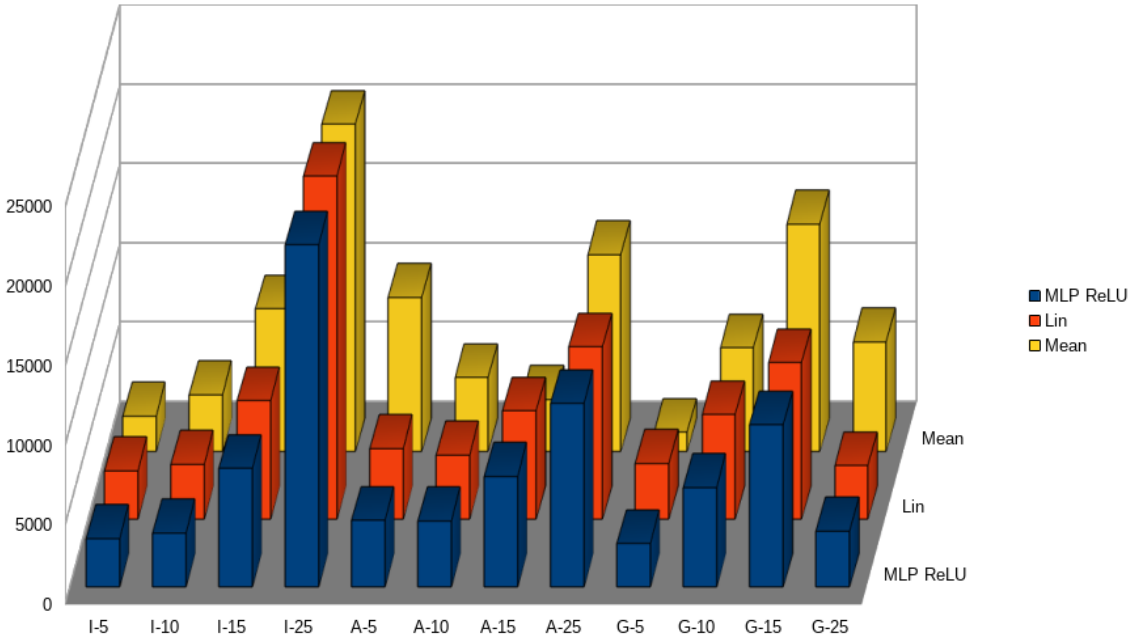

Fig. 7.2 MSE over all of the database for each model and scenario on Islands (I), A and Grid(G) at the end of the training.

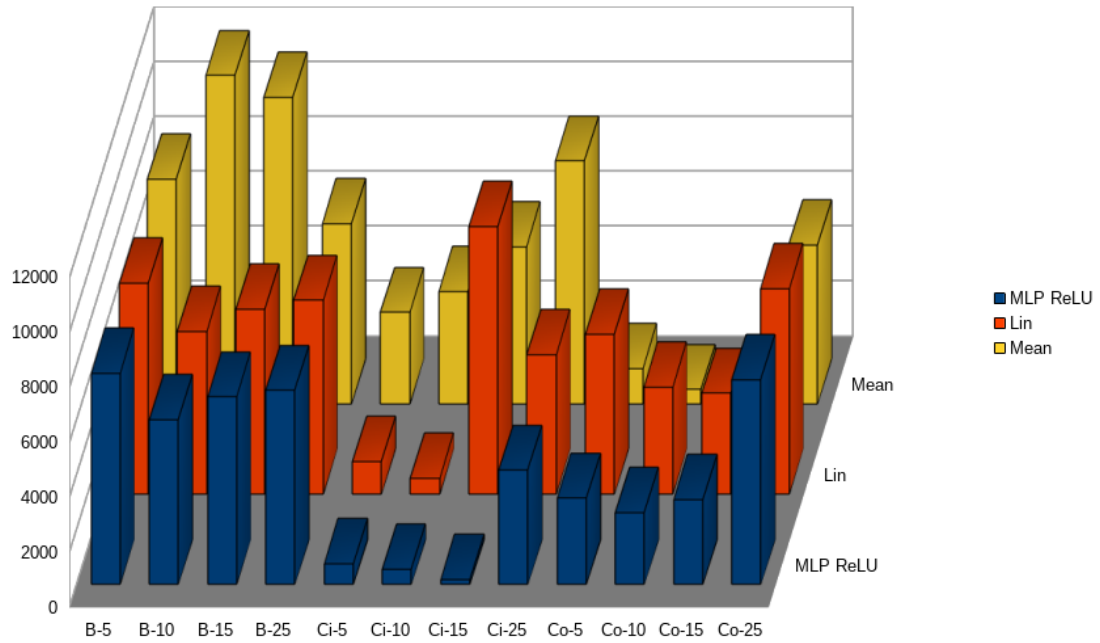

Fig. 7.3 MSE over the validation database for each model and scenario on B, Circle (Ci) and Corridor (Co) at the end of the training.

average MSE of 1905. Globally, in average over these three topologies MLP_{ReLU} is still the best model.

Finally, in average over the six topologies MLP_{ReLU} is the best model with an average MSE of 5626, although the difference with the other two models is small: 6254 for Lin and 6759 for $Mean$. The predominance of MLP_{ReLU} is coherent given the fact that being an ANN, its ability to learn complexe data structures is higher. However, it has to be pointed out that according to the training performance the difference with $Mean$ is quite low, $\sqrt{(6759)} - \sqrt{(5626)} = 7$ periods, although MLP_{ReLU} is by far more complex that $Mean$. The contribution of an ANN such as MLP_{ReLU} is therefore small.

7.2.2 Simulation results

The results of experiments on the RIE and IRIE strategies are now discussed.

As in **Chapter 5**, the six usual topologies and the four usual numbers of agents for each topology have been experimented. The RIE 0.2 and IRIE 0.2 strategies, have only been experimented on the three topologies Islands, A, and Grid, whereas the RIE 0.5 and IRIE 0.5 strategies, that is RIE and IRIE strategies whose the r_H and r_P parameters involved in the Heuristic and Pathfinder procedures are set to 0.5, have been experimented on the six topologies. 24 configurations and 72 scenarios with a RIE strategy have therefore been experimented, for which one scenario gave rise to 100 runs, and hence a total of 7200 missions run with a RIE strategy. For the IRIE strategies, the communication range r is set to 14. This choice has been made considering the graphs to patrol being situated in a frame of side 100, 14 corresponds to the tenth of the diagonal's lenght of this frame, that is approximately 141; a communication range corresponding to the tenth of the diagonal's lenght of the area to patrol seems reasonable in a critical patrolling mission. On the first three topologies, Islands, A and Grid, the RIE and IRIE, 0.2 and 0.5, strategies are compared with CR, the decentralised representative, HPCC 0.5 and RAMPAGER 2-50. Then, on B, Circle and Corridor, only RIE and IRIE 0.5 are compared with CR, HPCC 0.5 and RAMPAGER 2-50. The IE strategies, namely the deterministic ones, are not retained here due to their low performance; the RIE strategies always outperform them to a large extent, especially on the QMI and more generally in terms of robustness and variability. Finally, the focus will be placed on the WI and Iav; the WI to evaluate the variability of the strategy, and in doing so the quality of the distribution of agents over the nodes, and the Iav to evaluate the MI and QMI in an aggregated way, as shown in **Subsection 2.1.3**.

Figs. 7.4 to **7.15** show the WI and Iav on Islands, A, Grid, B, Circle and Corridor, respectively, of the RAMPAGER, the best RIE 0.2, IRIE 0.2, RIE 0.5 and IRIE 0.5, averaged over 100 runs for each scenario.

Globally, except for some scenarios on Circle and Corridor, HPCC 0.5 remains the best distributed strategy on both criteria and has the same performance as in **Subsection 5.2.5**. The best IRIE 0.2 and 0.5 are often the best strategies for all of the studied scenarios: the best IRIE 0.2 is almost always the best one on Islands, A and Grid, and the best IRIE 0.5 outperforms often the other strategies on B, Circle and Corridor. Most of the time RAMPAGER is outperformed by the best IRIE or RIE strategy. All of the best IRIE strategies are generally better than CR, the decentralised representative, except for the particular topologies Circle and Corridor, where CR is particularly suited to this topologies and is better than HPCC 0.5 for many scenarios on these graphs. Lastly, for all of the idleness estimator strategies *Mean* is almost always the best model for any number of agents different from 25.

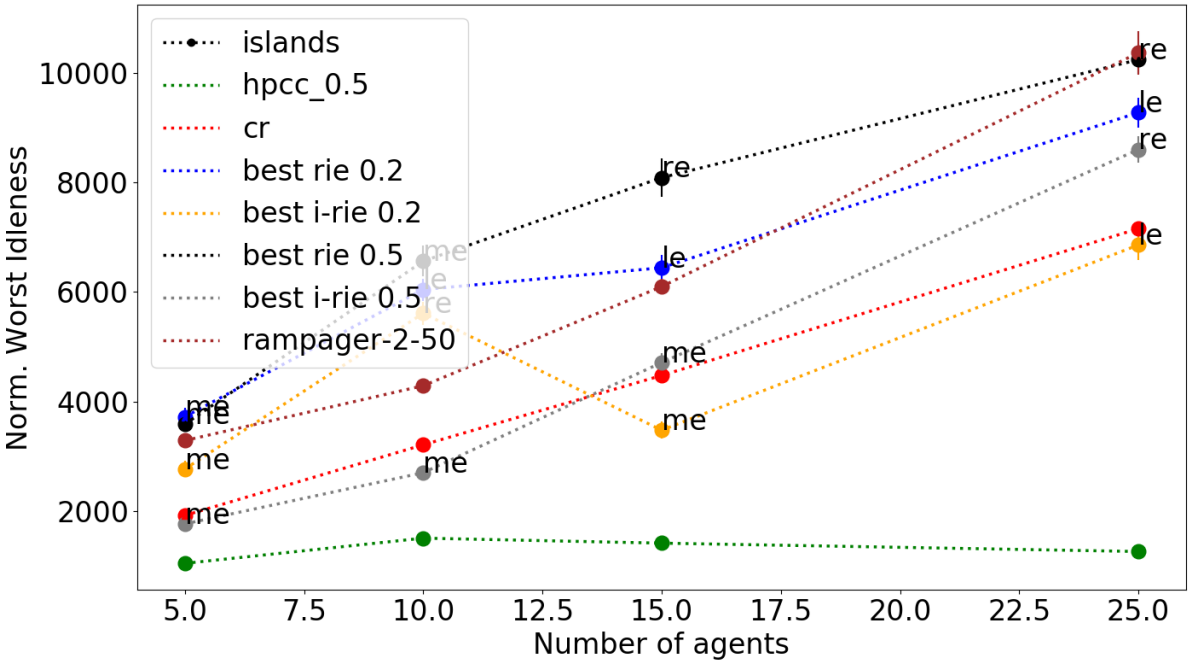

Fig. 7.4 Normalised WI, averaged over 100 runs, of the best variant of RAMPAGER, and the best RIE and IRIE w.r.t. the number of agents on Islands, with *me* standing for *Mean*, *le* for *Lin* and *re* for *MLPReLU*.

On Islands, **Fig. 7.4** and **Fig. 7.5** show that on the WI, IRHPME 0.5 is the best RIE strategy for 5 and 10 agents, whereas IRHPME 0.2 is the best one for 15, and IRHPLE 0.2 the best for 25 agents. For these scenarios, they outperform all CR on

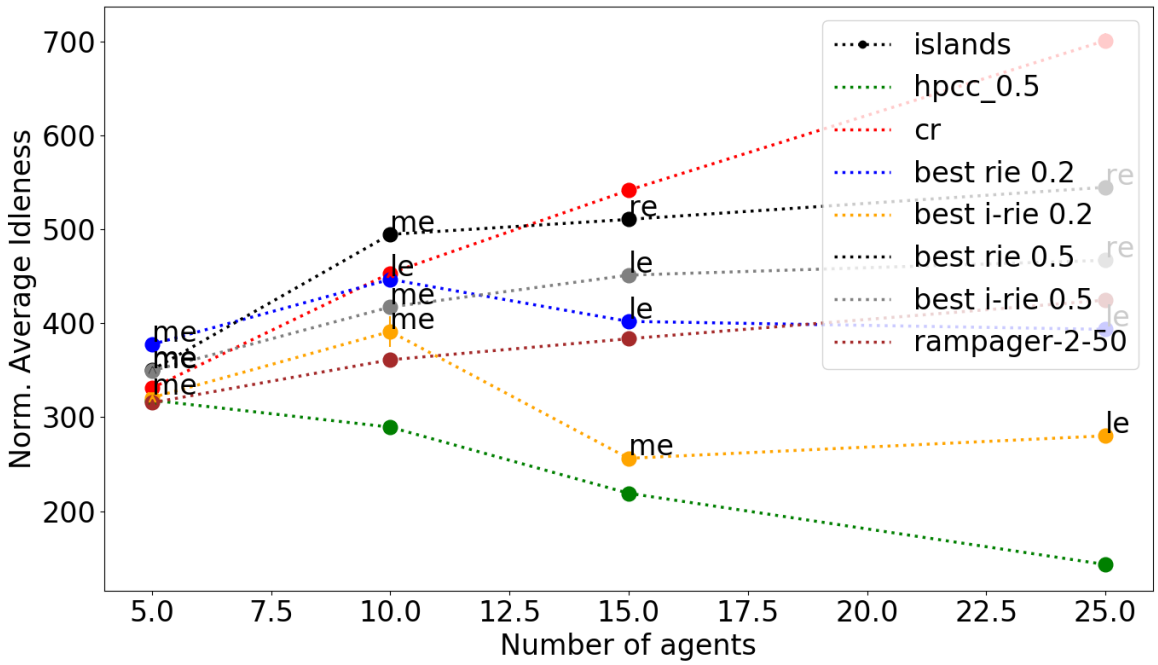

Fig. 7.5 Normalised Iav, averaged over 100 runs, of the best variant of RAMPAGER, and the best RIE and IRIE w.r.t. the number of agents on Islands, with *me* standing for *Mean*, *le* for *Lin* and *re* for *MLPReLU*.

the WI. For the Iav, almost all of the machine learning strategies are better than CR. RAMPAGER is the best machine learning strategy for 10 agents, and IRHPME 0.2 and IRHPLE 0.2 are the best for both 10, 15 agents, and 25 agents, respectively. Except for 25 agents, on the WI and the Iav RAMPAGER outperforms the RIE strategies, i.e. the non-interacting RIE strategies.

Fig. 7.6 and **Fig. 7.7** show the performance of the studied strategies on A. On both the WI and the Iav, IRHPME 0.2 is the best IE strategy, and is better than CR for all of the number of agents, except for 25 agents where IRHPLE 0.2 is the best one. It is worth noting that on the Iav, IRHPME 0.2 is even better than HPCC 0.5 for 5 agents, and very close to it for 10 agents. With regard to RAMPAGER and the best IRIE 0.5, they are globally outperformed by the best RIE 0.2 on the Iav, as well as on the WI for RAMPAGER.

On Grid, the results depicted on **Fig. 7.8** and **Fig. 7.9** show that, as previously, IRHPME 0.2 is the best IE strategy on both the WI and Iav, except for 25 agents where IRHPLE 0.2 remains the best one. RAMPAGER and the best IRIE 0.5 are generally worse than the best RIE 0.2, except for the best IRIE 0.5 which outperforms the best RIE 0.2 on the WI for 5 and 10 agents.

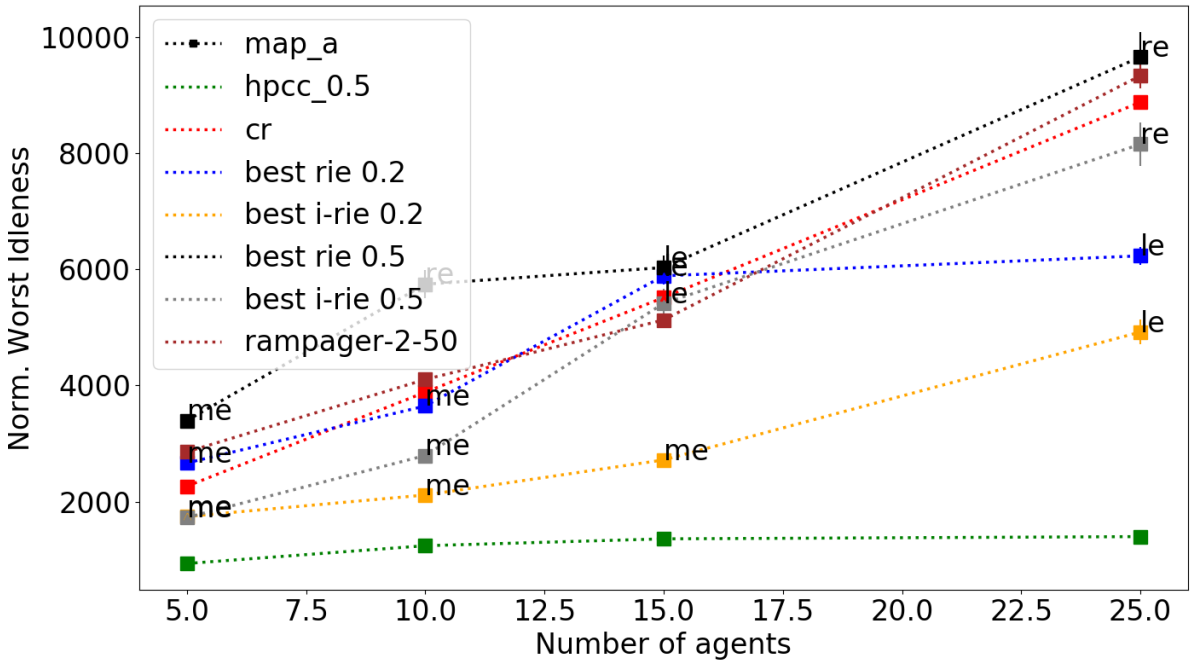

Fig. 7.6 Normalised WI, averaged over 100 runs, of the best variant of RAMPAGER, and the best RIE and IRIE w.r.t. the number of agents on A, with *me* standing for *Mean*, *le* for *Lin* and *re* for *MLPReLU*.

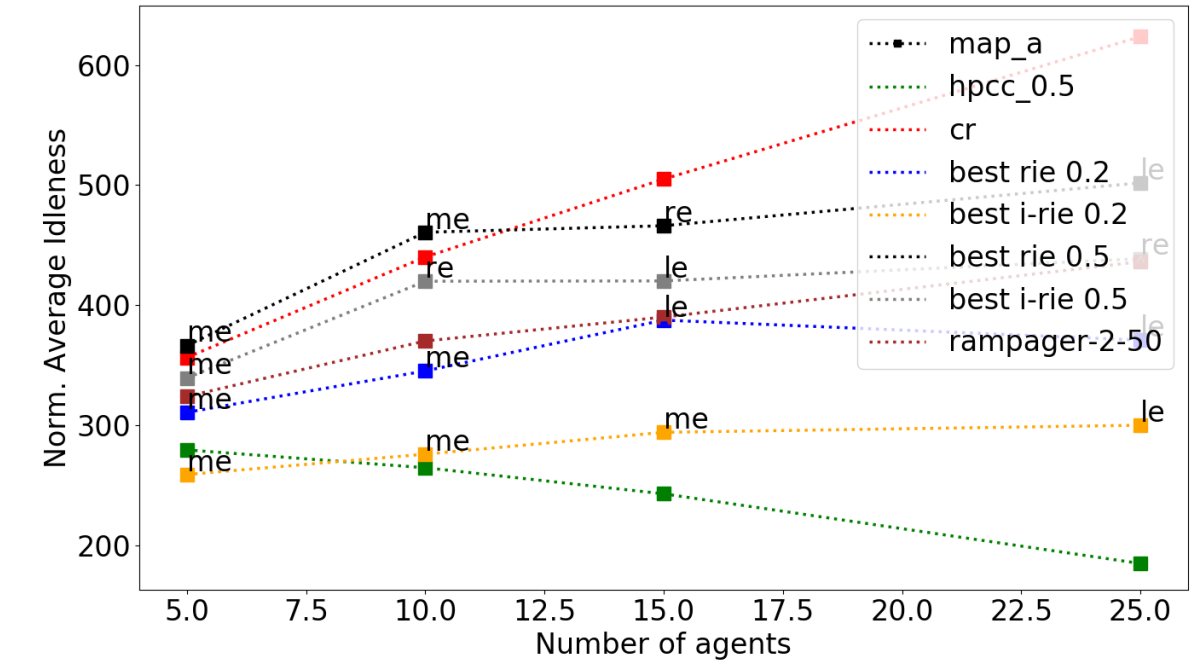

Fig. 7.7 Normalised Iav, averaged over 100 runs, of the best variant of RAMPAGER, and the best RIE and IRIE w.r.t. the number of agents on A, with *me* standing for *Mean*, *le* for *Lin* and *re* for *MLPReLU*.

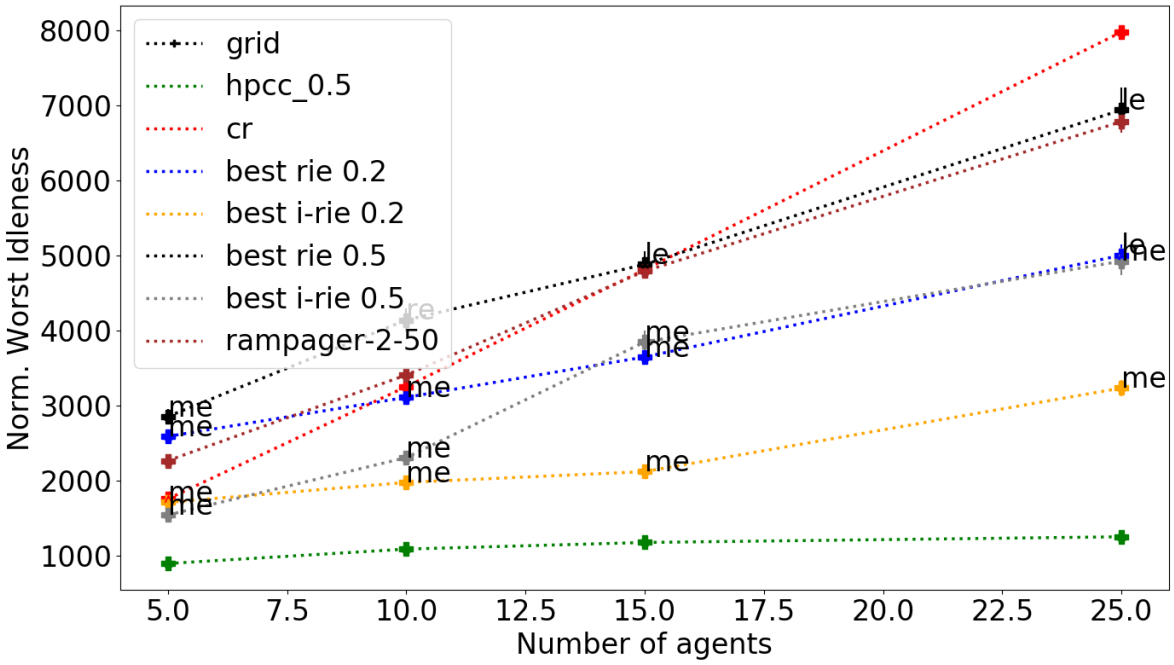

Fig. 7.8 Normalised WI, averaged over 100 runs, of the best variant of RAMPAGER, and the best RIE and IRIE w.r.t. the number of agents on Grid, with *me* standing for *Mean*, *le* for *Lin* and *re* for *MLPReLU*.

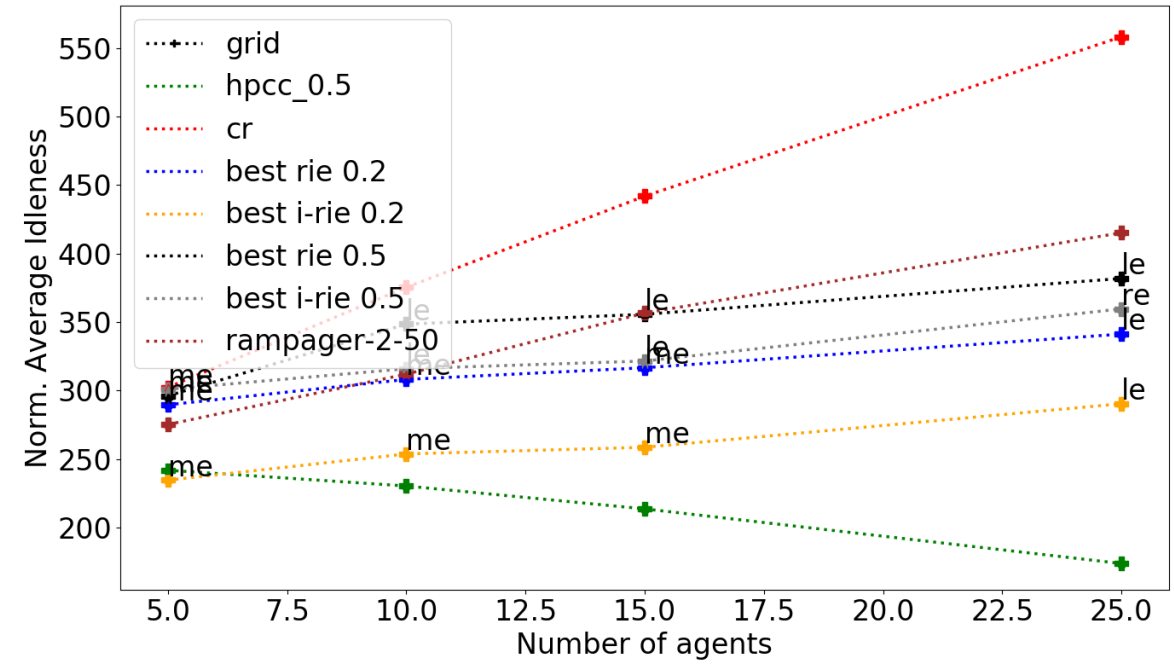

Fig. 7.9 Normalised Iav, averaged over 100 runs, of the best variant of RAMPAGER, and the best RIE and IRIE w.r.t. the number of agents on Grid, with *me* standing for *Mean*, *le* for *Lin* and *re* for *MLPReLU*.

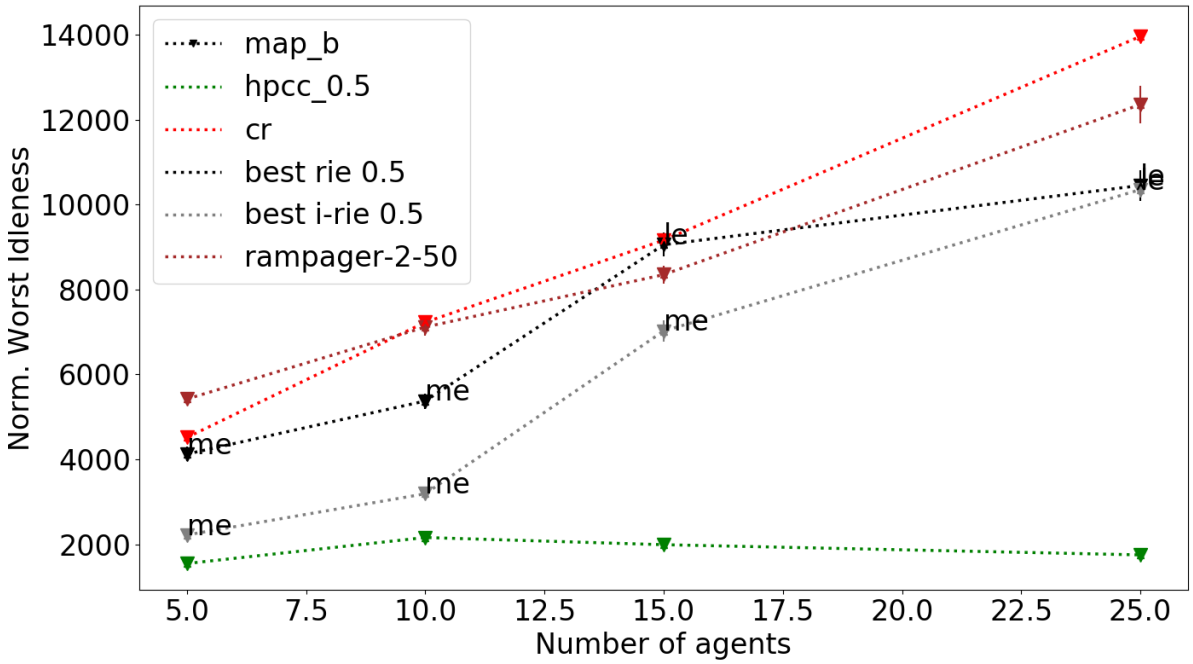

Fig. 7.10 Normalised WI, averaged over 100 runs, of the best variant of RAMPAGER, and the best RIE and IRIE w.r.t. the number of agents on B, with *me* standing for *Mean*, *le* for *Lin* and *re* for *MLPReLU*.

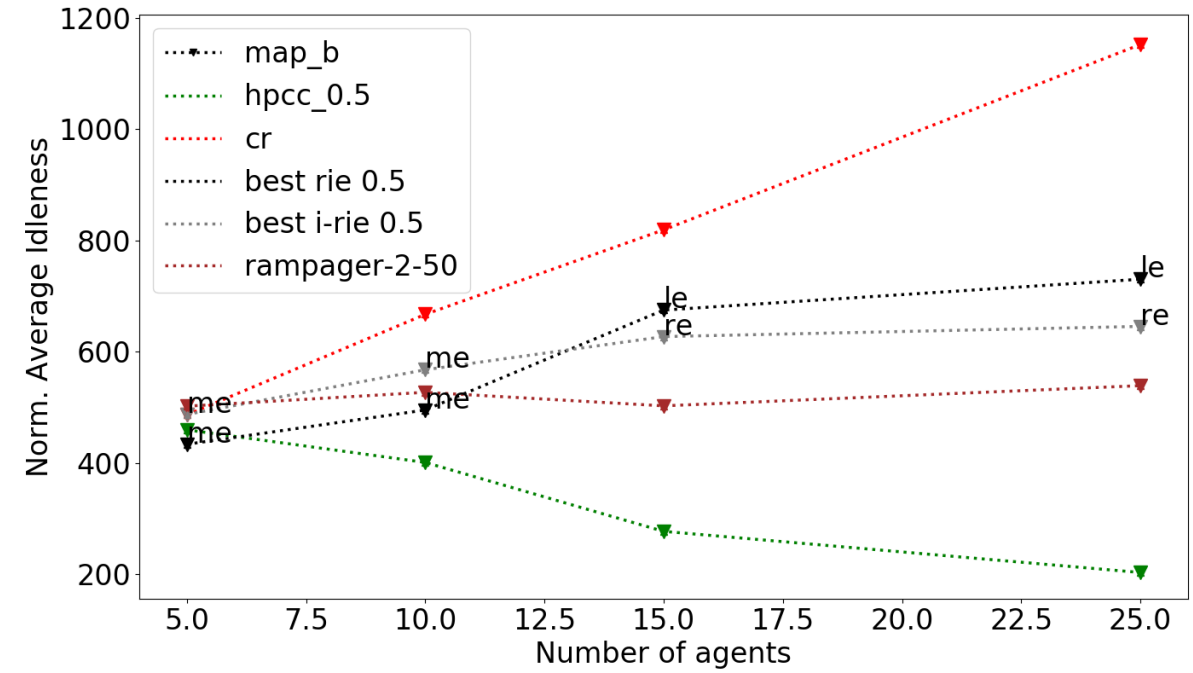

Fig. 7.11 Normalised Iav, averaged over 100 runs, of the best variant of RAMPAGER, and the best RIE and IRIE w.r.t. the number of agents on B, with *me* standing for *Mean*, *le* for *Lin* and *re* for *MLPReLU*.

On B, **Fig. 7.10** and **Fig. 7.11** show that IRHPME 0.5 is the best strategy after HPCC 0.5 on the WI, except for 25 agents where IRHPLE is the best strategy. On this criterion, the best RIE 0.5 is generally better than RAMPAGER. Then, on the Iav RHPME 0.5 is the best IE strategy for 5 and 10 agents, where it is even better than HPCC 0.5, whereas RAMPAGER is the best one for 15 and 25 agents. On this criterion, RAMPAGER outperforms globally the best RIE 0.5. In addition, CR is always outperformed by the RIE strategies and RAMPAGER on both criteria.

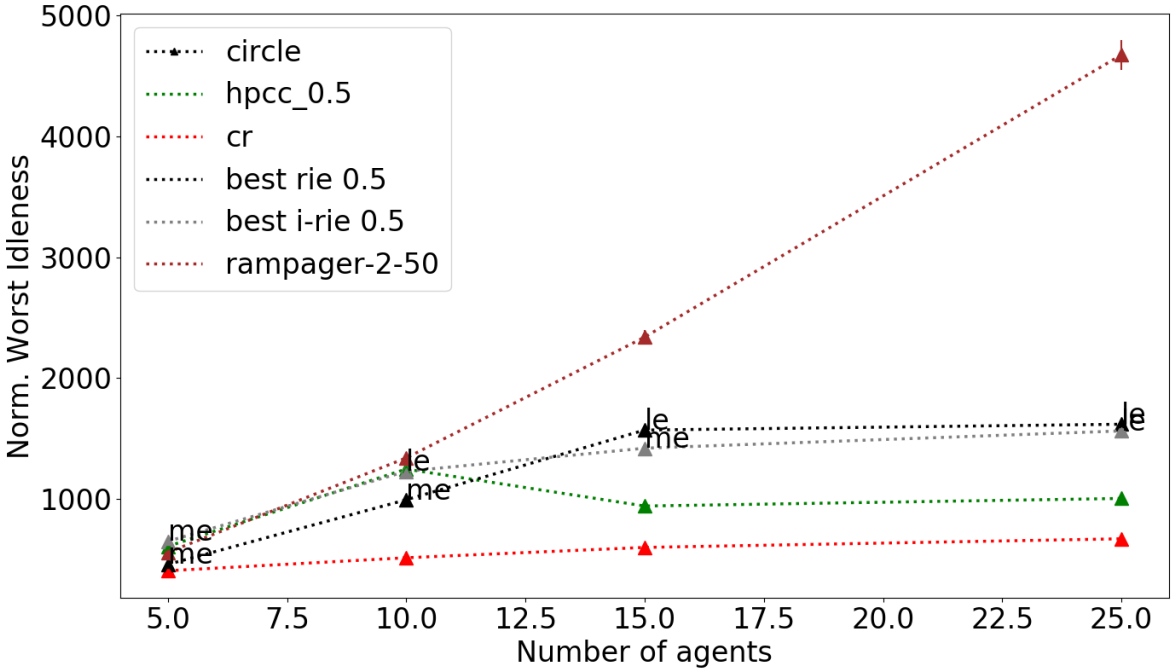

Fig. 7.12 Normalised WI, averaged over 100 runs, of the best variant of RAMPAGER, and the best RIE and IRIE w.r.t. the number of agents on Circle, with *me* standing for *Mean*, *le* for *Lin* and *re* for $MLPReLU$.

On Circle, **Fig. 7.12** and **Fig. 7.13** show that RHPME 0.5 and IRHPLE 0.5 are globally the best machine learning strategy. RHPME 0.5 is the best one, and is even better than HPCC 0.5, for 5 and 10 agents for both criteria, and IRHPLE 0.5 is the best one for 15 and 25 agents. RAMPAGER is generally outperformed by the best IRIE 0.5 and RIE 0.5. Also, as exposed in **Subsection 5.2.5**, CR is globally the best strategy.

Finally, on Corridor, **Fig. 7.14** and **Fig. 7.15** show that IRHPME 0.5 is the best strategy on the WI, whereas it is IRHPLE 0.5 on the Iav, apart from 5 agents where RHPME 0.5 outperforms all of the other strategies. On the Iav, RAMPAGER, the best RIE 0.5 and IRIE 0.5 outperform HPCC 0.5, although it corresponds to their model strategy. On the WI, RAMPAGER is strongly outperformed by RHPME 0.5 for 5 and

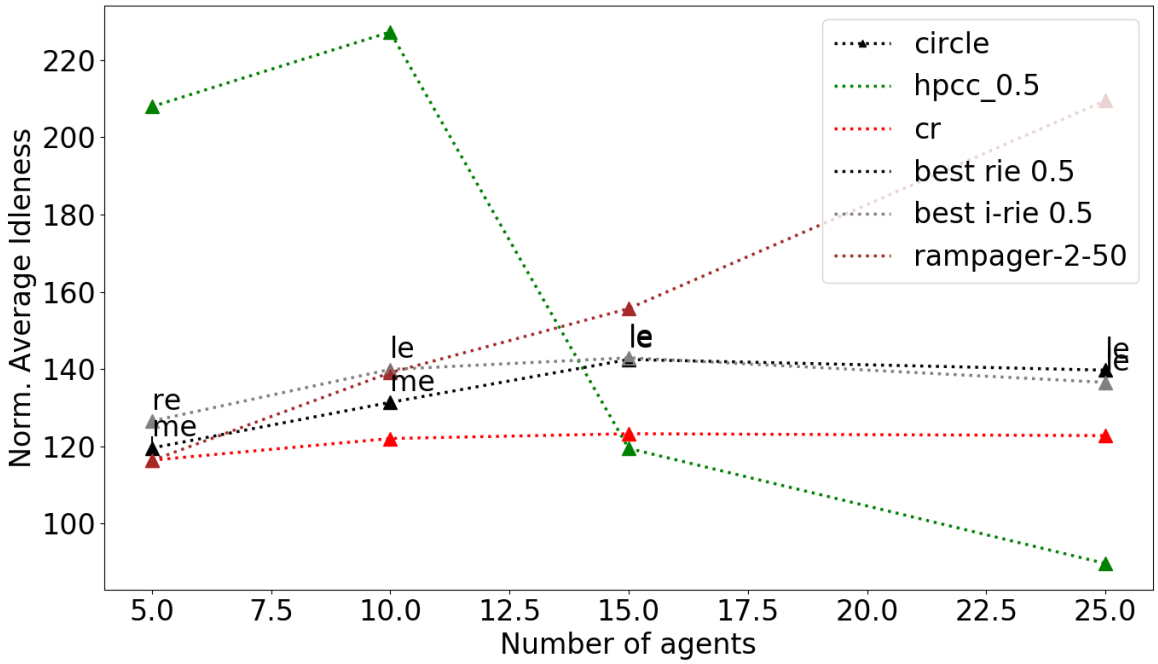

Fig. 7.13 Normalised Iav, averaged over 100 runs, of the best variant of RAMPAGER, and the best RIE and IRIE w.r.t. the number of agents on Circle, with *me* standing for *Mean*, *le* for *Lin* and *re* for *MLPReLU*.

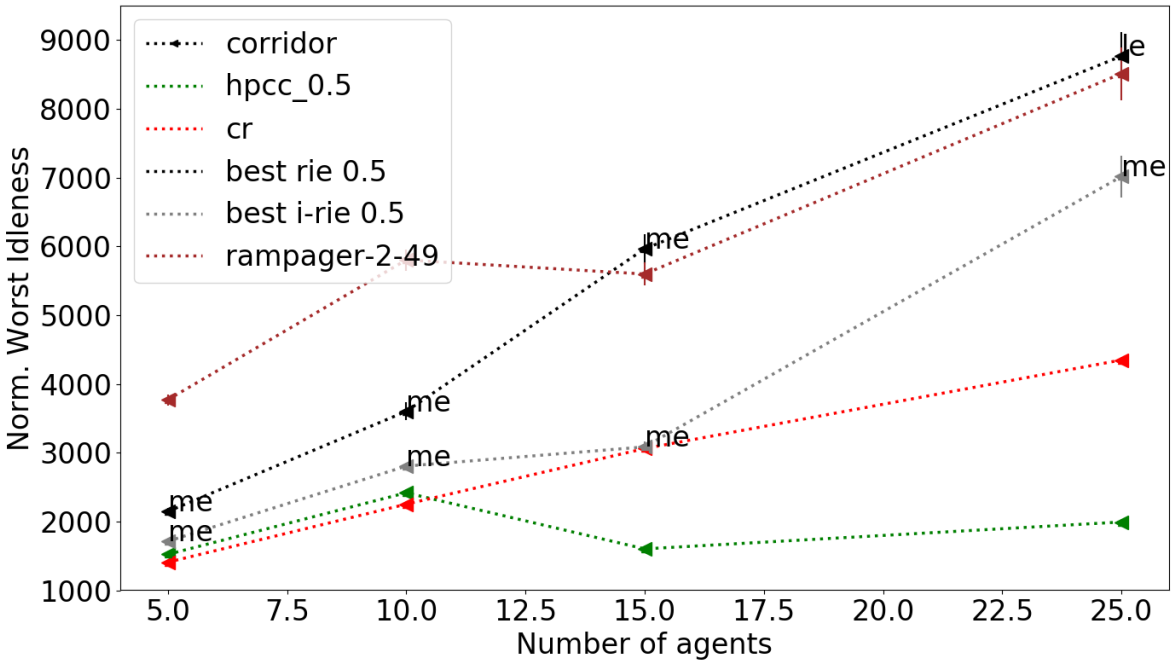

Fig. 7.14 Normalised WI, averaged over 100 runs, of the best variant of RAMPAGER, and the best RIE and IRIE w.r.t. the number of agents on Corridor, with *me* standing for *Mean*, *le* for *Lin* and *re* for *MLPReLU*.

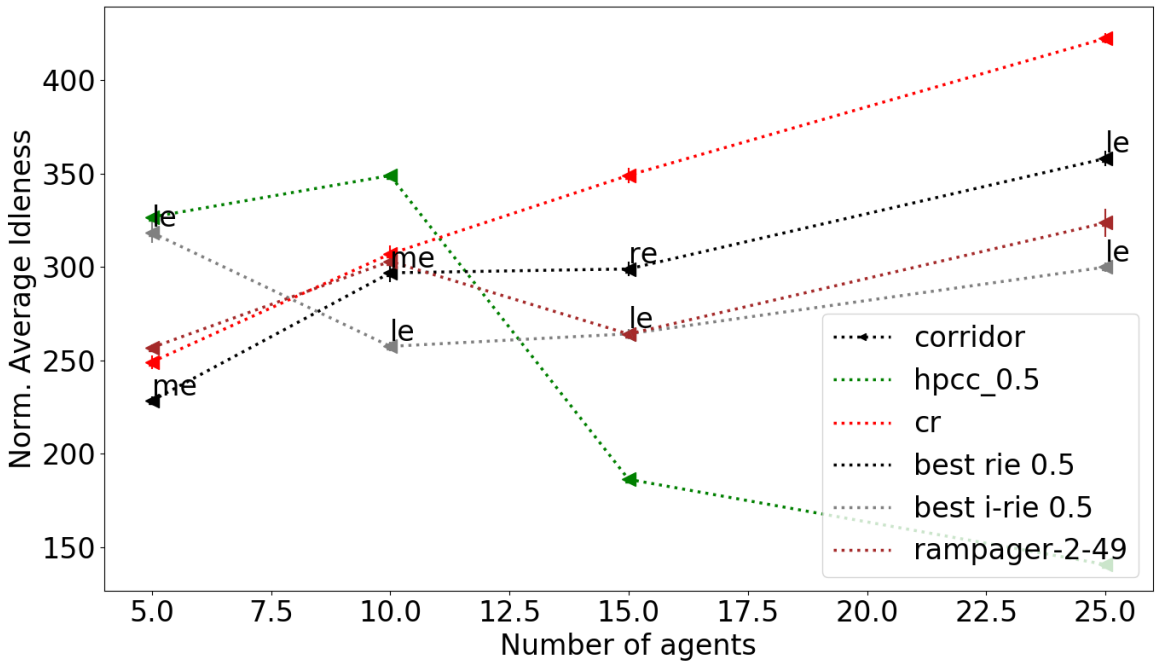

Fig. 7.15 Normalised Iav, averaged over 100 runs, of the best variant of RAMPAGER, and the best RIE and IRIE w.r.t. the number of agents on Corridor, with *me* standing for *Mean*, *le* for *Lin* and *re* for *MLPReLU*.

10 agents, but although it is close to them, it outperforms RHPME 0.5 and RHPLE 0.5 for 15 and 25 agents respectively. On the Iav, RAMPAGER is still outperformed by RHPME 0.5 for 5 and 10 agents, but it is better than RHPRE 0.5 and RHPLE 0.5 for 15 and 25 agent respectively, and by doing so, the latter being the best non-interacting RIE 0.5 strategies, it also outperfoms RHPME 0.5.

As in **Subsection 5.2.5**, learning from HPCC 0.5 for scenarios on Circle and Corridor with 5 and 10 agents is worthless.

Globally, a IRIE strategy is almost always the best strategy. Also, over all of the RIE strategies, whether interacting or not, *Mean* is the best statistical model when compared with the other RIE strategies on the same scenario, i.e. for each scenario involving RIE agents *Mean* has been the best model, such that it is the best:

- on the WI, in 60% of the cases, that is for 43 out of 72 scenarios,
- on the Iav, in 43% of the cases, namely for 31 out of 72 scenarios.

Likewise, either RHPME or IRHPME, which result from the *Mean* model, have turned out to be the best strategies over all of the configurations when compared with

the other strategies, i.e. for each MAP configuration the RHPME or IRHPME strategy outperform the other ones, such as they constitute the best strategy:

- on the WI, in 83% of the cases, i.e. for 20 out of 24 configurations,
- on the Iav, in 54% of the cases, that is to say for 13 out of 24 configurations.

Thereafter, RAMPAGER not being interacting, its performance are only compared with RHPME 0.2 and 0.5. Additional results, as well as those here depicted showed that:

- on the WI, either RHPME 0.2 or 0.5 outperform RAMPAGER in 50% of the cases, i.e. for 12 out of 24 configurations,
- on the Iav, either RHPME 0.2 or 0.5 outperform RAMPAGER in 58% of the cases, that is for 10 configurations out of 24.

Lastly:

- on the WI, RAMPAGER is outperformed by a non-interacting RIE strategy in 67% of the cases, namely for 16 out of 24 configurations
- on the Iav, RAMPAGER is outperformed by a non-interacting RIE strategy in 58% of the cases, that is to say for 14 out of 24 configurations.

Globally, RAMPAGER present therefore lower performance than the RIE strategies.

Mean, despite its simplicity, is thereupon the best statistical model as idleness estimator; it gives rise to the best machine learning strategy studied in this disseration. It is specifically a high-performance strategy in term of robustness: on the WI its non-interacting variants outperform RAMPAGER in 50% of the cases, and in taking into account the interacting variants in 60% of the cases. However, interestingly, it is rarely the best model for configurations with 25 agents.

Finally, comparing the simulation results established here with the training ones highlights there are no relations between training and simulation performance. Most of the time, the best statistical model in the training step is not the best model in simulation, whether it be for the RIE or IRIE strategies. This result originates certainly from the structure of the data used here and can be viewed as a consequence of **Theorem 6.2.1**. Also, the cost function used to optimise the statistical models, namely the MSE, might not be adapted to what is aimed at learning here. Considering estimated true idlenesses are to be used in the Heuristic and Pathfinder algorithms, what is the most relevant here

is not to approximate these values in average, but to reconstruct the order made up by the values of true idleness on the nodes: if $\forall t \in \mathbb{N}^*$, a node $v \in V$ has a true idleness greater than another node $w \in V$, the estimated true idleness of v ought to be greater than w's one, i.e $\hat{i}_v(t) > \hat{i}_w(t)$, whathever their values.

7.3 Conclusion

In this chapter it has been attempted to improve the IE strategies by adding information in their decision procedure by means of interaction to increase the complexity of the system, and therefore its robustness and adaptability. In the context of the information theory, interaction can be thought of as the counterpart of randomness introduced in the last chapter. To that end, a new type of strategy was introduced and defined: IRIE, which hinges on communication and interaction between agents — and which is in fact an improvement of the RIE strategy —, has given rise to three new strategies: Interacting Random Heuristic Pathfinder Mean Estimator (IRHPME), Interacting Random Heuristic Pathfinder Linear Estimator (IRHPLE) and Interacting Random Heuristic Pathfinder ReLU Estimator (IRHPRE).

In terms of results, the RIE strategies have globally shown good performance. The WI and Iav being evaluation criteria of robustness — the Iav as a quadratic function of QMI and the WI as a worst-case performance indicator —, on the WI the best RIE strategy outperforms RAMPAGER in 67% of the cases, and on the Iav in 58% of the cases. Regarding the IRIE strategies, they have turned out to be the best strategies, both in this chapter and in all this disseration. For all of the scenarios studied herein, the best IRIE strategy is practically always that which has the best performance.

Also, it is worth noting that a particular emphasis has to be placed on RHPME, the random strategy corresponding to model $Mean$, which is, interestingly enough, better than RAMPAGER in 50% of the cases on the WI. When considering IRHPME, it is the best statistical model in 60% of the cases on the WI, and in 43% of the cases on the Iav. Such a result highlights that the contribution of ANNs is quite limited, and considering the complexe procedure necessary to train ANNs compared to a simple calculation of an average, as well as the results set out above, it follows that $Mean$ seems to be the more appropriate statistical model.

Chapter 8

Conclusion

Recent years have seen an expansion of distributed artificial intelligence, and more precisely of multiagent applications, as well as many successes with certain of machine learning approaches. In this dissertation, these major fields of artificial intelligence have been combined and applied to the case study of the MAP problem, which provides a generic benchmark to study multiagent architectures. We proposed some methods based on supervised learning and studied their relevance in the context of MAP. Another aim of this work was to seize the opportunity of assessing machine learning algorithms in such a context, specifically by using these algorithms to create new decentralised or distributed multiagent strategies from a high-performance centralised strategy. Indeed, throughout this dissertation various machine learning models were studied.

In **Chapter 3**, some key concepts and tools were described, particularly the model of the temporal MAP problem and the characteristics of MAP data. A new methodology was proposed and implemented to distribute a centralised strategy. This methodology has turned out to be generic, and can be used beyond the frame of MAP to decentralise or distribute any centralised decision process into independent agents, to establish an equivalent decentralised strategy. To apply this methodology to our problematic, we developed a software ecosystem which is a prominent contribution of this work:

- PyTrol, an easy to use simulator coded in Python to study MAP strategies, but also to generate MAP data quickly,
- MAPTrainer, a framework based on PyTorch to train any machine learning model in the context of MAP,
- MAPTor, an annex tool to:

- prepare MAP configurations, scenarios and runs,
- process data generated by Pytrol in order to prepare them for the purpose of being trained in MAPTrainer,
- make statistics on simulation traces generated by PyTrol,
- plot statistics for each experimented MAP scenario.

Regarding PyTrol, in a not too distant future a concrete class will be implemented to allow agents and other communicating objects to communicate on TCP from remote machines, making the simulator completely distributed.

In **Chapter 4**, the first type of machine learning approach studied and assessed in the context of MAP was the learning of structure and parameters of a predictor used by an agent to predict of the next node to visit taking into account only its past decisions. Practically, we used LSTM networks that are well suited to learn temporal sequences, and in doing so, to MAP insofar as it constitutes a temporal decision problem. The new methodology laid down in **Chapter 3**, was implemented to give rise to a new MAP strategy based on LSTM networks used as node predictors. This strategy, called Random LSTM Path-Maker (RLPM), selects the next node to visit as being the outcome of a random drawing according to the distribution probability over the nodes generated by the LSTM network. The RLPM agents embed each an LSTM network used as node predictor. An important issue that had to be answered was whether the LSTM networks tightly integrate the structure of the topology.

In **Chapter 5**, we therefore made the hypothesis that LSTM networks could better learn to navigate over the area to patrol if the structure of the graph was integrated into the network. By doing so, an analytical procedure of an LSTM network initialisation with respect to the topology to patrol has been laid down, allowing the LSTM network to be wired so as it would not be able to predict a node not being a neighbour of that provided as input. Applying this structure-guided initialisation of LSTM network to RLPM gave rise to the RAMPAGER strategy. First experiments showed that the LSTM architectures with 1 or 2 layers and a number of neurons per layer equal to the number of nodes in the graph were globally the best among the LSTM architectures tested in this chapter, although results has been found to be mitigated, especially in term of robustness. However, it turned out that they were often better than CR, the decentralised representative strategy. Other experiments showed that RAMPAGER has turned out to be superior to the best RLPM variant. RAMPAGER also outperformed CR, for almost all of the scenarios and evaluation criteria, except for the particular Circle and Corridor topologies. The preliminary hypothesis has then been confirmed. This

strategy is particularly well adapted to missions where communication are forbidden or impossible.

In **Chapter 6**, we considered a second type of machine learning approach, that is learning of structure and parameters of an estimator which estimates the global state of the system from the partial knowledge of the agent and using the resulting estimate being then used in classical algorithms to derive the next decision of the agent. Practically, for MAP this approach lead to train estimators of true idleness. The aim was to train them to estimate true idleness with respect to individual idleness, namely to learn the relation mapping individual to true idlenesses. This new application gave rise to a new type of MAP strategy, named IE, which is based on idleness estimation: agents are endowed with the trained machine learning model that they use in mission as idleness estimator. However, although 8 machine learning models were trained and evaluated, only 3 out of the 8 were retained, according to their training performance.

First experiments highlighted that the IE strategies are not robust, and present generally poor performance. Also, the trained models led to high errors owing to the nature of the relation to capture: we demonstrated that this relation is likely not a function with **Theorem 6.2.1**, which is a notable contribution of this chapter. This theorem indicates that there exists at least a simple and canonical topology for which the relation between the vectors of individual idleness and the vector of true idleness is not a function, but a multivalued function. Further studies could be led to address this problem by appending features to data vectors to turn the multivalued function into a classical function, i.e. a single-valued function.

Thus, the RIE type of strategy was created from the IE type, by appending a step of random drawing of idlenesses to the decision procedure, random drawing performed according to a highly entropic probability distribution whose the expectation is provided by the estimator. This evolution of the IE strategies gave rise to three new strategies: RHPME, RHPLE and RHPRE, and has turned out to be, by far, better than the deterministic IE strategies.

In **Chapter 7**, the IE strategies were improved to make them more robust, using, this time, opportunistic information. To that end, information was added in the decision-making process in the form of communication and interaction, to better distribute the visits of agents over the nodes. Then, integrating the interaction scheme into the RIE strategies led to the new IRIE type of strategy, and gave rise concretely to the IRHPME, IRHPLE and IRHPRE strategies.

Finally, the RIE strategies showed good performance in experiments, they outperformed RAMPAGER in 60% of the cases in average according to the Iav and WI. The system made up of RIE agents has become more robust by means of random variations. Regarding the IRIE strategies, they turned out to be the best strategies of this dissertation, showing often the highest performance.

Interestingly, the mean, as idleness estimator, has been found to be the best machine learning model of this work: it outperformed all of the other models studied in this dissertation. Even in a non-interacting context, it is better than RAMPAGER in 50% of the cases on the WI, whereas considering IRHPME, its corresponding interacting strategy, it gave rise to the best strategy in 60% of the cases on the WI, and 43% of the cases on the Iav. This result has highlighted that the contribution of ANNs is small for the two approaches proposed in the context of MAP.

Thus, in this dissertation a ground work is provided and several new types of strategies based on machine learning have been established, although many issues remain unresolved, suggesting some directions for future work in the intersection of MAP and machine learning.

Preliminary avenues to explore would be to use a new cost function to train machine learning models differently. This could improve robustness evaluation criteria, such as the WI or QMI. A training system which would optimise directly the evaluation criterion to optimise, e.g. the WMI or QMI, based on reinforcement learning could be considered.

In the case of LSTM networks, that is to say the RLPM and RAMPAGER strategies which are strategies based on node predictors, to exploit the potential of LSTM networks for the generation of paths as part of MAP more sophisticated and complex architectures could be implemented and evaluated in the future. Also, an interaction scheme for these strategies could be investigated, to make RLPM agents interact and exchange information.

For the IE strategies, regarding the problem of estimation of true idleness, other ANN architectures could be considered, although the values of the MSE criterion on their own do not seem to be germane compared with the order of idlenesses. Indeed, with respect to the Heuristic and Pathfinder algorithms, what must be reconstructed when trying to estimate a vector of true idleness with respect to a vector of individual idleness, is the order on the nodes made up by the true idlenesses. Another approach could be to take into account in the learning process the fact that the output of the estimator is a parameter of a distribution and to consider the cross-entropy as criterion to optimise instead of the MSE.

The problem of generalisation, pertaining to any machine learning model studied in this dissertation, could also be investigated. In fact, although certain of models or architectures be efficient for some configurations, none is able to outperform the other ones in all configurations. The sole model that outperforms the other ones to a large extent is the mean in its random versions, namely RHPME or IRHPME.

References

[1] Acevedo, J. J., Arrue, B. C., Maza, I., and Ollero, A. (2013). Cooperative large area surveillance with a team of aerial mobile robots for long endurance missions. *Journal of Intelligent & Robotic Systems*, 70(1-4):329–345.

[2] Almeida, A., Castro, P., Menezes, T., and Ramalho, G. (2003). Combining idleness and distance to design heuristic agents for the patrolling task. In *II Brazilian Workshop in Games and Digital Entertainment*, pages 33–40.

[3] Almeida, A., Ramalho, G., Santana, H., Tedesco, P., Menezes, T., Corruble, V., and Chevaleyre, Y. (2004). Recent advances on multi-agent patrolling. In *Brazilian Symposium on Artificial Intelligence*, pages 474–483. Springer.

[4] Bar-Yam, Y. (2004). Multiscale variety in complex systems. *Complexity*, 9(4):37–45.

[5] Basilico, N., Gatti, N., Rossi, T., Ceppi, S., and Amigoni, F. (2009). Extending algorithms for mobile robot patrolling in the presence of adversaries to more realistic settings. In *2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, volume 2, pages 557–564. IEEE.

[6] Bennett, M., Schatz, M. F., Rockwood, H., and Wiesenfeld, K. (2002). Huygens's clocks. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 458(2019):563–579.

[7] Beynier, A. (2016). Cooperative multiagent patrolling for detecting multiple illegal actions under uncertainty. In *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 9–16. IEEE.

[8] Chevaleyre, Y. (2004). Theoretical analysis of the multi-agent patrolling problem. In *Intelligent Agent Technology, 2004.(IAT 2004). Proceedings. IEEE/WIC/ACM International Conference on*, pages 302–308. IEEE.

[9] Choset, H. (2001). Coverage for robotics–a survey of recent results. *Annals of mathematics and artificial intelligence*, 31(1-4):113–126.

[10] Chu, H. N., Glad, A., Simonin, O., Sempe, F., Drogoul, A., and Charpillet, F. (2007). Swarm approaches for the patrolling problem, information propagation vs. pheromone evaporation. In *19th IEEE international conference on tools with artificial intelligence (ICTAI 2007)*, volume 1, pages 442–449. IEEE.

[11] Davoodi, A., Fazli, P., and Mackworth, A. K. (2009). On multi-robot area coverage. In *Proceedings of the 7th Japan Conference on Computational Geometry and Graphs, JCCGG 2009*, pages 75–76.

[12] Drogoul, A. and Collinot, A. (1998). Applying an agent-oriented methodology to the design of artificial organizations: A case study in robotic soccer. *Autonomous agents and multi-agent systems*, 1(1):113–129.

[13] Drogoul, A., Landau, S., and Muñoz, A. (2007). Robocup: un benchmark pour l'iad?

[14] D'Ambrosio, D. B., Goodell, S., Lehman, J., Risi, S., and Stanley, K. O. (2012). Multirobot behavior synchronization through direct neural network communication. In *International Conference on Intelligent Robotics and Applications*, pages 603–614. Springer.

[15] Elmaliach, Y., Shiloni, A., and Kaminka, G. A. (2008). Frequency-based multi-robot fence patrolling. *Bar Ilan Univ., Comput. Sci. Dept., Maverick Group, Tech. Rep*, 1:2008.

[16] Fazli, P., Davoodi, A., Pasquier, P., and Mackworth, A. K. (2010). Complete and robust cooperative robot area coverage with limited range. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5577–5582. IEEE.

[17] Glad, A., Buffet, O., Simonin, O., and Charpillet, F. (2009). Self-organization of patrolling-ant algorithms. In *2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 61–70. IEEE.

[18] Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional lstm networks. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 4, pages 2047–2052. IEEE.

[19] Guo, Y., Parker, L. E., and Madhavan, R. (2007). Collaborative robots for infrastructure security applications. In *Mobile robots: the evolutionary approach*, pages 185–200. Springer.

[20] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[21] Karypis, G. and Kumar, V. (1998). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392.

[22] King, R. E. (1999). *Computational intelligence in control engineering*. CRC Press.

[23] Konar, A. (2006). *Computational intelligence: principles, techniques and applications*. Springer Science & Business Media.

[24] Lauri, F., Charpillet, F., et al. (2006). Ant colony optimization applied to the multi-agent patrolling problem. In *IEEE Swarm Intelligence Symposium*.

[25] Lauri, F. and Koukam, A. (2008). A two-step evolutionary and aco approach for solving the multi-agent patrolling problem. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 861–868. IEEE.

[26] Lauri, F. and Koukam, A. (2014). Robust multi-agent patrolling strategies using reinforcement learning. In *International Conference on Swarm Intelligence Based Optimization*, pages 157–165. Springer.

[27] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436.

[28] Machado, A., Almeida, A., Ramaldo, G., Zucker, J.-D., and Drogoul, A. (2002a). Multi-agent movement coordination in patrolling. In *Proceedings of the 3rd International Conference on Computer and Game*, pages 155–170.

[29] Machado, A., Ramalho, G., Zucker, J.-D., and Drogoul, A. (2002b). Multi-agent patrolling: An empirical analysis of alternative architectures. In *International Workshop on Multi-Agent Systems and Agent-Based Simulation*, pages 155–170. Springer.

[30] Marier, J.-S., Besse, C., and Chaib-Draa, B. (2010). Solving the continuous time multiagent patrol problem. In *2010 IEEE International Conference on Robotics and Automation*, pages 941–946. IEEE.

[31] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

[32] Menezes, T., Tedesco, P., and Ramalho, G. (2006). Negotiator agents for the patrolling task. In *Advances in Artificial Intelligence-IBERAMIA-SBIA 2006*, pages 48–57. Springer.

[33] Mo, J. P. and Sinha, A. (2014). *Engineering systems acquisition and support*. Elsevier.

[34] Moreira, D. M., Ramalho, G., and Tedesco, P. A. (2009). Simpatrol-towards the establishment of multi-agent patrolling as a benchmark for multi-agent systems. In *ICAART*, pages 570–575.

[35] Oliehoek, F. A., Spaan, M. T., Terwijn, B., Robbel, P., and Messias, J. V. (2017). The madp toolbox: an open source library for planning and learning in (multi-) agent systems. *The Journal of Machine Learning Research*, 18(1):3112–3116.

[36] Pasqualetti, F., Franchi, A., and Bullo, F. (2012). On cooperative patrolling: Optimal trajectories, complexity analysis, and approximation algorithms. *IEEE Transactions on Robotics*, 28(3):592–606.

[37] Portugal, D., Couceiro, M. S., and Rocha, R. P. (2013). Applying bayesian learning to multi-robot patrol. In *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 1–6. IEEE.

[38] Portugal, D., Pippin, C., Rocha, R. P., and Christensen, H. (2014). Finding optimal routes for multi-robot patrolling in generic graphs. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 363–369. IEEE.

[39] Portugal, D. and Rocha, R. (2010). Msp algorithm: multi-robot patrolling based on territory allocation using balanced graph partitioning. In *Proceedings of the 2010 ACM symposium on applied computing*, pages 1271–1276. ACM.

[40] Portugal, D. and Rocha, R. P. (2013). Distributed multi-robot patrol: A scalable and fault-tolerant framework. *Robotics and Autonomous Systems*, 61(12):1572–1587.

[41] Portugal, D. and Rocha, R. P. (2016). Cooperative multi-robot patrol with bayesian learning. *Autonomous Robots*, 40(5):929–953.

[42] Poulet, C. (2013). *Coordination dans les systèmes multi-agents: Le problème de la patrouille en système ouvert.* PhD thesis, Paris 6.

[43] Prim, R. C. (1957). Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401.

[44] Rekleitis, I., New, A. P., Rankin, E. S., and Choset, H. (2008). Efficient boustrophedon multi-robot coverage: an algorithmic approach. *Annals of Mathematics and Artificial Intelligence*, 52(2-4):109–142.

[45] Ren, W. and Cao, Y. (2011). Overview of recent research in distributed multi-agent coordination. In *Distributed Coordination of Multi-agent Networks*, pages 23–41. Springer.

[46] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

[47] Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning internal representations by error propagation. *Parallel distributed processing*, 1:318–362.

[48] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.

[49] Russell, S. J. and Norvig, P. (1995). *Prentice Hall Series in Artificial Intelligence.* Prentice Hall Englewood Cliffs, NJ:.

[50] Sak, T., Wainer, J., and Goldenstein, S. K. (2008). Probabilistic multiagent patrolling. In *Brazilian Symposium on Artificial Intelligence*, pages 124–133. Springer.

[51] Sales, D. O., Feitosa, D., Osório, F. S., and Wolf, D. F. (2012). Multi-agent autonomous patrolling system using ann and fsm control. In *2012 Second Brazilian Conference on Critical Embedded Systems*, pages 48–53. IEEE.

[52] Sampaio, P. A., Ramalho, G., and Tedesco, P. (2010). The gravitational strategy for the timed patrolling. In *2010 22nd IEEE International Conference on Tools with Artificial Intelligence*, volume 1, pages 113–120. IEEE.

[53] Santana, H., Ramalho, G., Corruble, V., and Ratitch, B. (2004). Multi-agent patrolling with reinforcement learning. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1122–1129. IEEE Computer Society.

[54] Siddique, N. and Adeli, H. (2013). *Computational intelligence: synergies of fuzzy logic, neural networks and evolutionary computing.* John Wiley & Sons.

[55] Stout, B. (1996). Smart moves: Intelligent pathfinding. *Game developer magazine*, 10:28–35.

[56] Tan, J., Xi, N., Sheng, W., and Xiao, J. (2004). Modeling multiple robot systems for area coverage and cooperation. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 3, pages 2568–2573. IEEE.

[57] Werbos, P. (1974). Beyond regression:" new tools for prediction and analysis in the behavioral sciences. *Ph. D. dissertation, Harvard University*.

[58] Wiandt, B. and Simon, V. (2018). Autonomous graph partitioning for multi-agent patrolling problems. In *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 261–268. IEEE.

[59] Williams, R. J. and Peng, J. (1990). An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural computation*, 2(4):490–501.

Appendix A

Artificial Neural Networks

A.1 Some key concepts of information theory

A.1.1 Entropy

In 1948 Shannon defined *information*, a new notion which allows quantifying the degree of incertainty of a system. In the same time, he also defined a new measure, called *Shannon's entropy*, to quantify information. Intuitively, the more predictable a system, the more certain it is, and the less information it will carry. It should be added that, as far as entropy measures the quantity of information, and actually of uncertainty, the complexity of a system being the uncertainty emerging from interactions taking place between its components, entropy measures also the complexity of this system.

A.1.2 Cross-entropy

Let P and Q be two probability measures with the same support $\mathcal{X}$. Then, the cross-entropy between P and Q can be interpreted as the average quantity of information — of base corresponding to that of the used log — carried by Q, with respect to a reference distribution P. Mathematically, the cross-entropy between P and Q corresponds to:

$$\mathcal{H}(P, Q) = \sum_{x \in \mathcal{X}} P(x) \log(Q(x)) \tag{A.1.1}$$

It follows that the cross-entropy is a relevant error function to learn a probability distribution insofar as it is minimum when $P = Q$.

A.2 Artificial neuron model

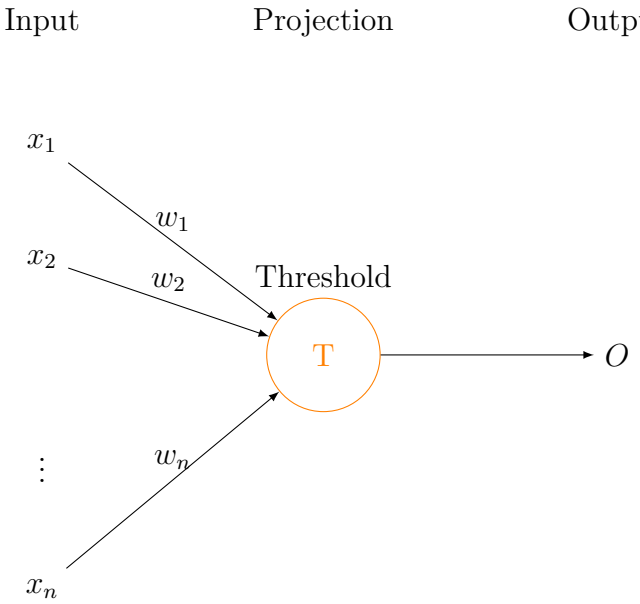

Fig. A.1 McCulloch and Pitts's neuron model.

The first model of an *artificial neuron*, shown in **Fig. A.1**, was outlined in 1943 by Warren McCulloch and Walter Pitts [31]. In this model, an artificial neuron is a *computational unit* which was modelled with an eletrical circuit. In this model, the firing rule is defined as:

$$O = \begin{cases} 1 \text{ if } \sum_{i=1}^{n} w_i \; x_i \geq T \\ 0 \text{ if } \sum_{i=1}^{n} w_i \; x_i < T \end{cases} \tag{A.2.1}$$

where $\forall i \in [|1, n|]$, x_i is the ith input, w_i is the weight of x_i — weights represent the proportion of information flowing from the input to the neuron —, and $T \in \mathbb{R}$ is a threshold value.

Thereafter, Frank Rosenblatt invented the *perceptron* [46], also known as *linear perceptron*, shown in **Fig. A.2**, adding a bias to McCulloch and Pitt's model, turning the firing rule into:

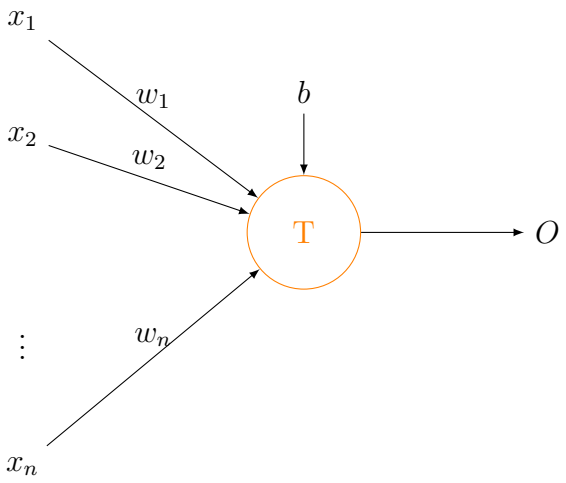

Fig. A.2 Rosenblatt's perceptron model.

$$O = \begin{cases} 1 \text{ if } \sum_{i=1}^{n} w_i \; x_i - b \geq T \\ 0 \text{ if } \sum_{i=1}^{n} w_i \; x_i - b < T \end{cases} \tag{A.2.2}$$

where $b \in \mathbb{R}$ is the bias.

Then, this first artificial neuron model was generalised so that it can output any real value, and not any more only 0 or 1. In the new model, the neuron outputs:

$$neur = \sum_{i=1}^{n} w_i \; x_i + b \tag{A.2.3}$$

where b is now regarded as the threshold value.

Afterwards, to distort the input space with the aim of making the outputs linearly separable, a nonlinear activation function has been included in the neuron arrangement. The output of the neuron in now expressed in the form:

$$y = f(neur) = f(\sum_{i=1}^{n} w_i \; x_i + b) \tag{A.2.4}$$

Adding an activation function in the neuron arrangement enables using adequate levels of amplification where necessary for small input signals, which avoids the risk of driving the output to unacceptable limits [54].

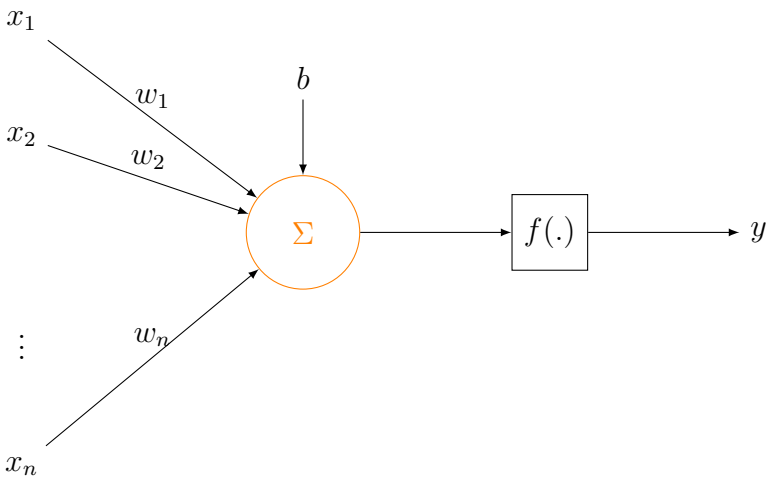

Fig. A.3 Perceptron model.

Finally, such a generalisation of the artificial neuron model has led to the perceptron model shown in **Fig. A.3**, which is, in fact, a misnomer for a neuron model more complicated than the linear perceptron.

When several artificial neurons are connected so that information can flow from neuron to neuron, the resultant network of neurons is called *artificial neural network* (ANN). As a consequence, an ANN is merely a nonlinear regression model meant to approximate any function not being linearly separable.

A.3 Network architectures and algorithms

Considering the type of connectivity in an ANN, two basic *achitectures* of networks are distinguished:

- feedforward neural networks,
- recurrent neural networks (RNNs).

A.3.1 Feed-forward neural networks

Feed-forward networks are characterised by forward connectivity: the signal traversing the network flows only from the input to the output layer. Among the large variety

of feed-forward ANNs, *multi-layer perceptron* (MLP) networks are composed of several stacked layers of neurons. Except for the input layer, each neuron uses a nonlinear activation function, and maps the output of the previous layer to the input of the current layer. As set out above, each one computes a weighted and biased sum of the previous layer's inputs, which is finally passed through the nonlinear activation function, making up thereby the neuron's output. In addition, as part of function approximation, on each layer the nonlinear activation functions provide a truncated basis, in its linear algebra acceptation, to perform the approximation.

Using multilayer ANNs has been made possible thanks to the works of both Paul Werbos [57] and Rummelhart, Hinton and Williams [48] [47]. Werbos discovered the mathematical framework for the new training scheme of layered networks, that is to say the *backpropagation* procedure, and used it to estimate a dynamic model to predict nationalism and social communications as part of his PhD thesis [57]. With regard to Rumelhart, Hinton, and Williams, in the seminal book *Parallel Distributed Processing, Vol. 1*, they introduced for the first time the use of the backpropagation procedure to adjust the weights of the connections in an ANN so as to minimise a measure of the difference between the actual output vector of the ANN and the desired output vector [48] [47].

Feed-forward network architectures are composed of three types of layers:

- the *input layer*, which is the entry point of the network, forwards the vector presented to it; such a vector stands for the data to process
- the *hidden layers*, which correspond to successive nonlinear transformations distorting the input,
- the *output layer*, which is the last layer wherefrom the network's response is delivered; this layer can also be linear.

When the hidden layers have the same number of neurons, such a number is called *hidden size* or *hidden dimension*.

Neuron output will now be characterised mathematically for each layer of an ANN. Let:

- $\mathbf{H}$ be the set of hidden layers of the network,
- $\forall l > 0, H_l \in \mathbf{H}$ be the set of the neurons' identifiers on the layer l, and H_0 be the set of the inputs' identifiers on the input layer,

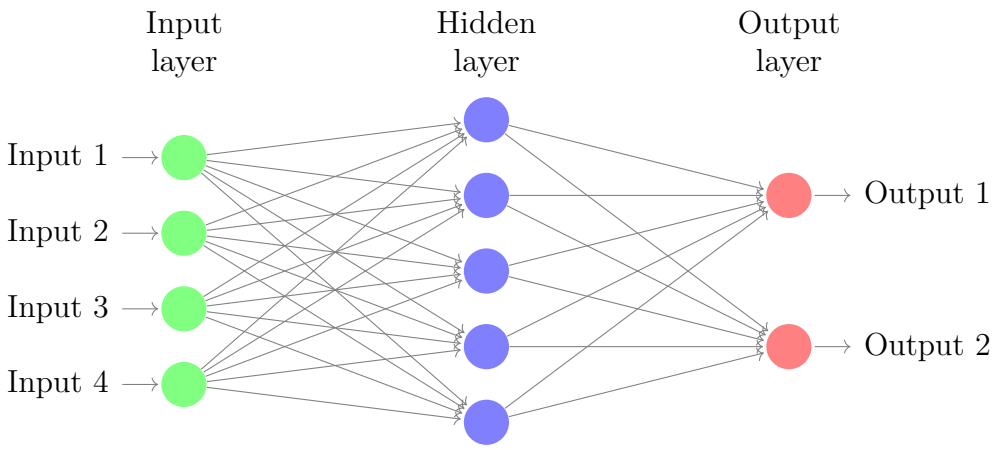

Fig. A.4 Feed-forward ANN with one hidden layer.

- $N_{in} \in \mathbb{N}$ be the dimension of the ANN's input layer,
- $y^0 \in \mathbb{R}^{N_{in}}$ be an input vector.

Then, to each neuron $i \in H_l$, standing for the ith neuron of the layer H_l, is associated:

- $(w_{ij}^l)_{i \in H_{l-1}}$, the weight vector of j,
- $b_i^l \in \mathbb{R}$, the bias of i,
- f_l, the activation function associated with the layer l,

The output of i noted y_i^l corresponds then to:

$$\forall l \in \mathbb{N}^* : l \leq \ card(\mathbf{H}),\ z_i^l = \sum_{j \in H_{l-1}} w_{ij}^l y_j^{l-1} + b_i^l \tag{A.3.1}$$

$$\forall l \in \mathbb{N}^* : l \leq \ card(\mathbf{H}),\ y_i^l = f(z_i^l) \tag{A.3.2}$$

From the foregoing, to each layer l can be associated a matrix of weights, noted W_l, of shape $card(H_l) \times card(H_{l-1})$, and a vector of biases, noted B_l, of shape $card(H_l)$. Each line i of W_l and B_l correspond to the vector of weights $(w_{ij}^l)_{j \in H_{l-1}}$, and the bias b_i^l, respectively. Let F^l be the element-wise activation function, $\forall l \in [|1, card(\mathbf{H})|]$, the output vector of the layer l, noted y^l, can then be defined such as:

$$y^l = F^l(W_l \cdot y^{l-1} + B_l) \tag{A.3.3}$$

Several kinds of activation functions can be applied to z, for example the *logistic sigmoid*, $f(z) = 1/(1+exp(-z))$, the hyperbolic tangent $f(z) = (exp(z)-exp(-z))/(exp(z)+exp(-z))$, and the *linear rectifier* (ReLU), $f(z) = max(0, z)$. It is worth noting that the framework can be extended so that the use of the identity function $f(z) = z$ instead of a nonlinear as activation function leads to an *affine transformation* layer. Moreover, allowing optional weights and bias leads to a constant or *linear mapping* layer.

A.3.2 Learning

In supervised machine learning such networks are generally optimised using *gradient-descent-based methods*, such as the *backpropagation algorithm*, by minimising an *objective function*, also called *criterion*, representing the difference between the output of the network and its desired value [57] [48] [47]. When the objective function is minimised, it may be called *cost function*, *loss function*, or *error function*, though a distinction may be made between loss function and cost function: the former can be regarded as the error for an input-output pair, while the latter as the error over the whole batch or database. Quite common objective functions are the *Mean Squared Error* (MSE), or the *cross-entropy loss* (CE) when data are probabilities. Lastly, algorithms used to optimise such error functions are called *optimisation algorithm* or *learning algorithm*, and thereupon, the process consisting in optimising such a function is termed *learning* or *training*, with respect to the context.

A.3.3 Recurrent neural networks

Recurrent Neural Networks (RNNs) are neural networks which process an input sequence one element at a time, while maintaining in their hidden units — neurons in the *hidden layers* — a *state vector* called *hidden state*, containing information about the history of the sequence's past elements. Each output of the hidden units h_t depends upon the hidden state h_{t-1}. This hidden state can be viewed as a *memory*. Indeed, adding memory to a neural network allows to process information of the sequence itself: the sequential information is preserved in the hidden state that enable finding correlations between events separated by several time steps. This memory is contained in the *hidden layers* which have a *feedback loop*; such layers are called *recurrent layers*.

A.3.3.1 Long Short-Term Memory

Long Short-Term Memory networks are a special kind of RNN introduced and designed to take into account long-term dependencies. They have the same general chain structure

as the RNNs except that the repeating module has a different structure as shown in **Fig. A.5**. In the first place, in their seminal article Hochreiter et al. have defined an LSTM network as being a RNN with one input layer, one fully self-connected hidden layer containing purpose-built *memory cells* and *gate units*, and an output layer [20]. This memory unit corresponds to a neuron with a recurrent self-connection. As a consequence, a cell referred originally to an object with a single scalar output. The activations of those neurons within the memory units constitute the *state* noted c_t, sometimes called *cell state*, of the LSTM network.

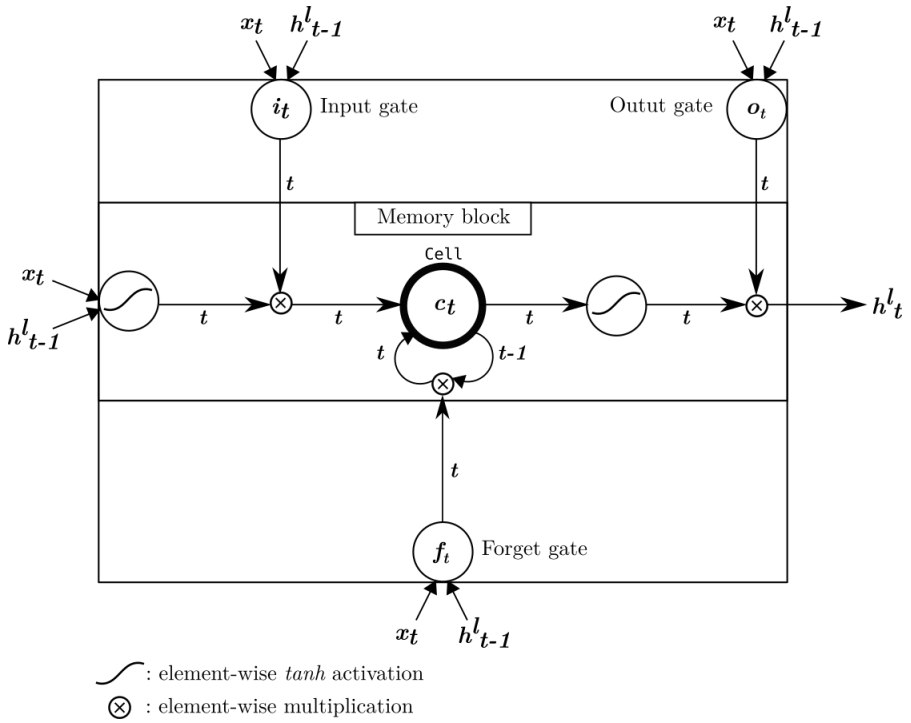

Fig. A.5 Layered LSTM unit: the core composant of the LSTM architecture.

An *LSTM layer* consists of a set of recurrently connected blocks, known as *memory blocks*, which in turn consists of cells [18]. Each cell, being a neuron, outputs a scalar. In the original model, as shown in **Fig. A.6**, each memory block contains one or more layered recurrently connected neurons, called memory cells, and shares the same two multiplicative units: i_t the *input gate*, and o_t the *output gate*; all the cells of a memory block are connected to the same gate units [20]. The gate units provide continuous analogies of the write and read operations for the cells. A memory block of size 1 is then a simple memory cell in the original model of Hochreiter et al. connected to *tanh* activations [20]. These blocks can be thought of as a differentiable version of the memory chips in a digital computer. In doing so, it follows the network can only interact with

the cells via the gates. Thereafter, f_t, the *forget gates*, has been introduced; such an additional gate can be regarded as reset operation for the cell.

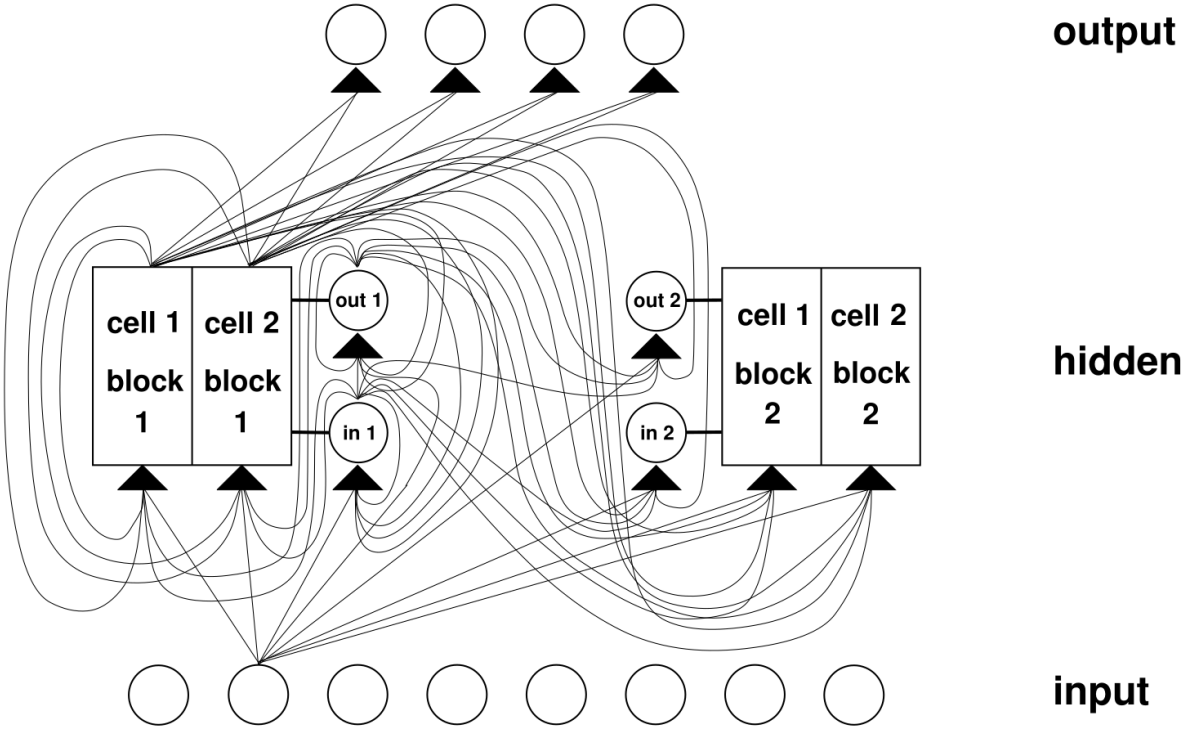

Fig. A.6 Example of an original LSTM network with 8 inputs, 4 outputs, and 2 memory blocks of size 2, such as defined by Hochreiter et al. [20]

Finally, the memory block and the gates form the *LSTM unit*, which corresponds to a repeating module. The state is thereupon the memory accumulated by the LSTM through time by using its forget, input and output gates. However, unlike the basic RNN model in which it covers the same concept, the cell state c_t must not be confused with the hidden state h_t, the former being the cell output while the latter the output of the hidden layers.

Also, $\forall l \in [|1, L|] : L \in \mathbb{N}^*$ is the number of LSTM layers, it should be emphasised that the hidden state, respectively the cell state, noted h_t, respectively c_t, of an LSTM network, must be distinguished from the hidden state, respectively the cell state, of the layer l, noted h_t^l, respectively c_t^l. In fact, one has $h_t = (h_t^1, \ldots, h_t^L)$ and $c_t = (c_t^1, \ldots, c_t^L)$

For some years and hitherto, most implemented LSTM architectures are devised as if they contain only one cell in their LSTM units. The LSTM units of a same layer can thereupon be "layered" into only one LSTM unit, as shown in **Fig. A.5**, where for all t a time step, i_t, f_t, o_t and c_t, the input gate, forget gate, output gate and cell activation,

respectively, turn consequently into vectors with the same size as the hidden vector h_t; hence the element-wise multiplication $*$. In that context, an LSTM layer can be viewed as a vectorial LSTM unit and therefore both the vectorial cell and gates compose a layer. It follows that defining the size of a layer's cell defines that of its memory cell block and that of its hidden state in cascade.

The hidden state output from an LSTM layer l is computed according to the following composite function:

$$i_t^l = \sigma(W^{l,ix}\ x_t^l + W^{l,ih}\ h_{t-1}^l + b^{l,i}) \tag{A.3.4}$$
$$f_t^l = \sigma(W^{l,fx}\ x_t^l + W^{l,fh}\ h_{t-1}^l + b^{l,f}) \tag{A.3.5}$$
$$o_t^l = \sigma(W^{l,ox}\ x_t^l + W^{l,oh}\ h_{t-1}^l + b^{l,o}) \tag{A.3.6}$$
$$c_t^l = f_t^l * c_{t-1}^l + i_t^l * \tanh(W^{l,cx}\ x_t^l + W^{l,ch}\ h_{t-1}^l + b^{l,c}) \tag{A.3.7}$$
$$h_t^l = o_t^l * tanh(c_t^l) \tag{A.3.8}$$

The parameters of an LSTM layer to learn for a layer l are consequently:

- $W^{l,ix}, W^{l,ih}, W^{l,fx}, W^{l,fh}, W^{l,ox}, W^{l,oh}, W^{l,cx}$ and $W^{l,ch}$
- $b^{l,i}, b^{l,f}, b^{l,o}$ and $b^{l,c}$

It has also to be pointed out that the structure corresponding to several memory blocks in a layer l, as established in the work of Hochreiter et al. [20], can be derived from its more general architecture by setting to 0 the elements of $W^{l,ih}$, $W^{l,fh}$, $W^{l,oh}$ which are not block-diagonal.

Finally, Deep LSTMs combine the multiple levels of representation that have proven to be effective in deep networks with the flexible use of long-range context that empowers RNNs. The architecture of the deep LSTMs is the same as that outlined previously apart from the fact that there are several LSTM layers.

Stateful and stateless LSTM. A LSTM network is said *stateful* if during training it is necessary to reset its cell and hidden states between each batch, in addition to being reset after each number of time steps of backpropagation through time, according to the TBPTT algorithm. A LSTM network is then said *stateless* if it is not stateful, that is $\forall n \in \mathbb{N}^*$ such as s_n is the n^{th} sequence of each batch, the LSTM network is trained as if s_n ranged from the first batch to the last one, in other terms, the s_n's in all of the

batches make up one sequence.

A.3.4 Learning specific to RNNs

A.3.4.1 Backpropagation through time.

BackPropagation Through Time (BPTT) is the application of the backpropagation algorithm to RNNs. The BPTT algorithm consists in backpropagating the gradient error as if the RNN was unrolled in time; the gradient error is therefore backpropagated in space and time. More precisely, the network is first unrolled for all the time steps. Then all of the elements of the sequence are provided to the network, while for each time step the gradients are calculated and accumulated. Finally, the network is rolled back up while keeping the accumulation of gradients, and weights are updated. It is worth noting that to use this algorithm, the entire history of network input and network state since the first time step must be saved.

However, BPTT can be computationally expensive as the number of time steps increases. It can also lead to the vanishing or exploding gradient problem. To address these problems, William and Peng have proposed a modification of the BPTT algorithm in which only a fixed number of time steps h is considered for the backpropagation [59]. This modification is equivalent to use a bounded-history approximation to the BPTT algorithm consisting in retaining only relevant information for the fixed number of time steps h, and forgetting any information older than h time steps. This can be thought of as a heuristic technique for simplifying the computation. The resultant algorithm is called *truncated backpropagation through time* (TBPTT) [59].

Appendix B

Additional results

B.1 Performance of some HPCC variants

B.1.1 Average idleness (Iav)

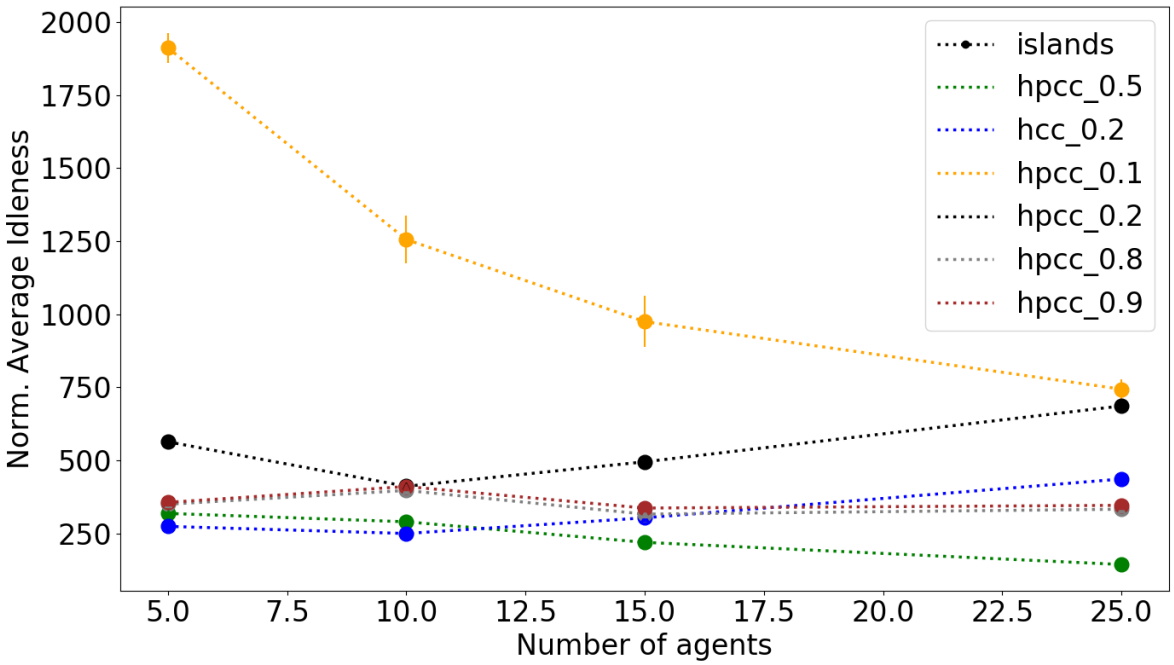

Fig. B.1 Normalised Iav, averaged over 100 runs, of different variants of HPCC for 5, 10, 15 and 25 agents on the Islands topology.

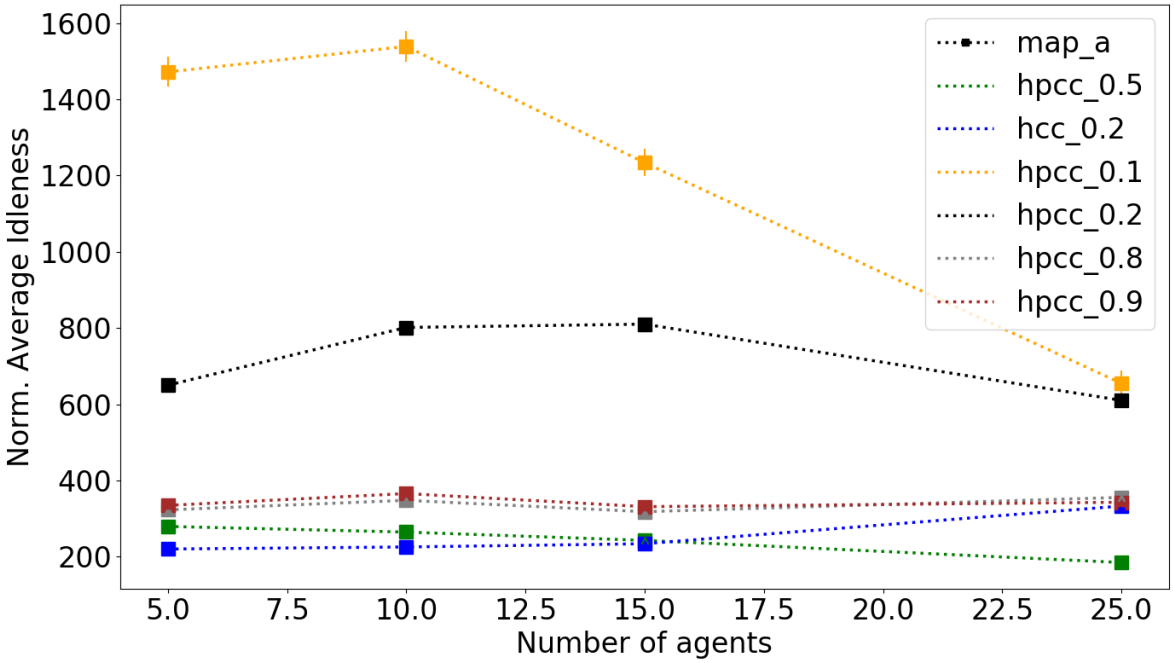

Fig. B.2 Normalised Iav, averaged over 100 runs, of different variants of HPCC for 5, 10, 15 and 25 agents on the A topology.

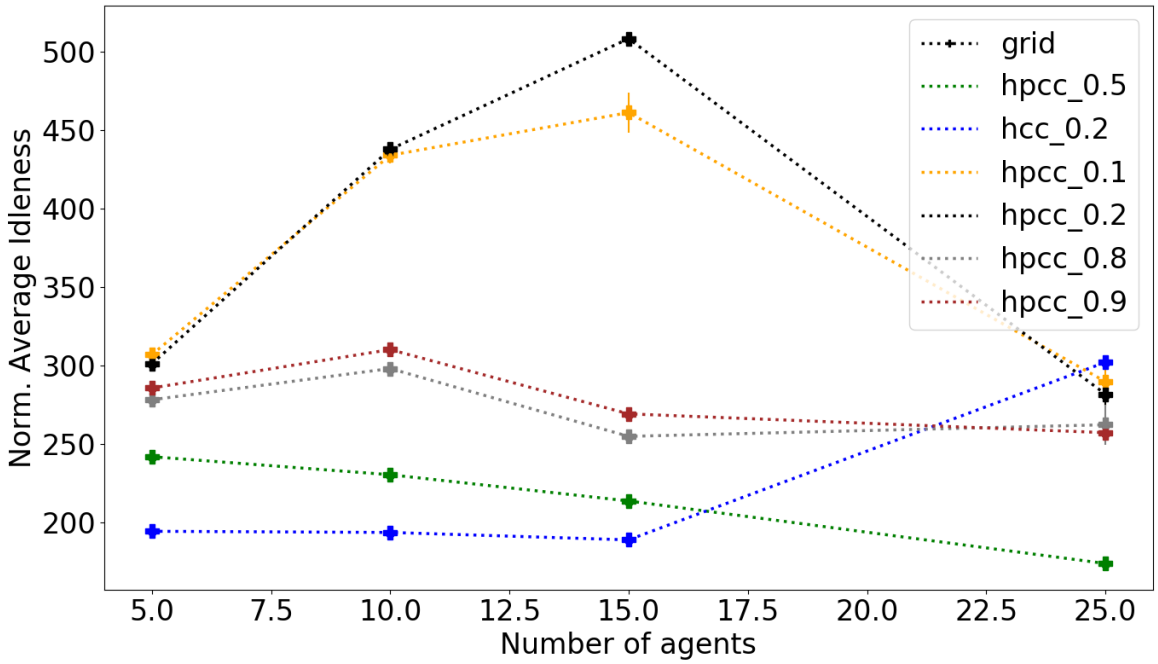

Fig. B.3 Normalised Iav, averaged over 100 runs, of different variants of HPCC for 5, 10, 15 and 25 agents on the Grid topology.

B.1.2 Mean Interval (MI)

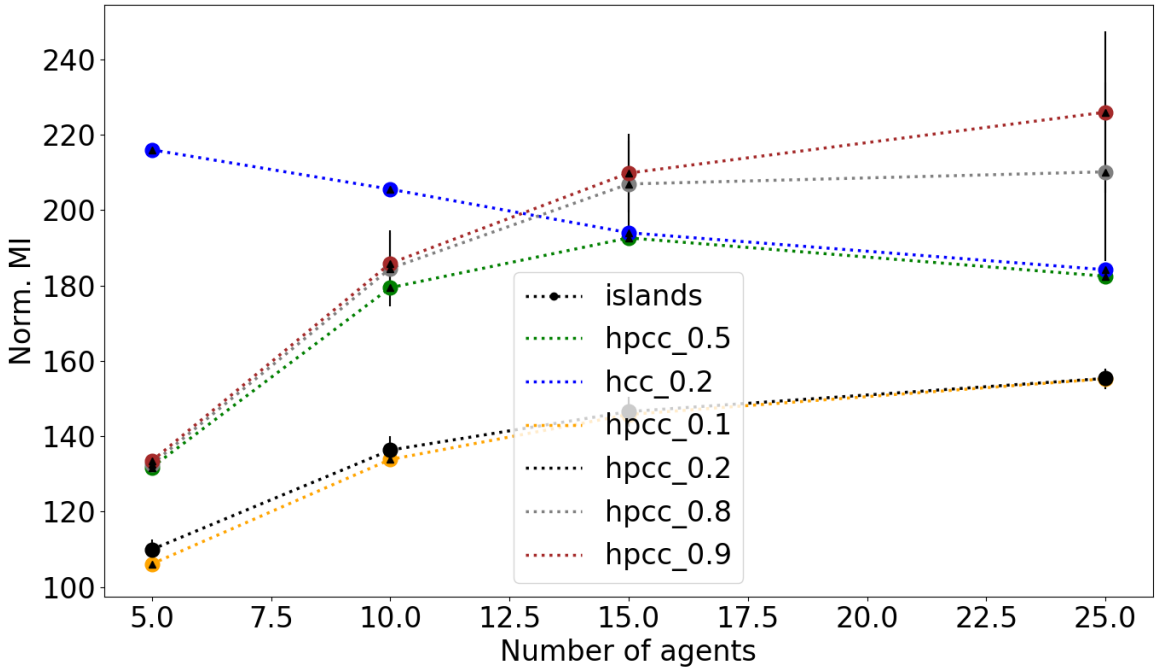

Fig. B.4 Normalised MI, averaged over 100 executions, of different variants of HPCC for 5, 10, 15 and 25 agents on the Islands topology.

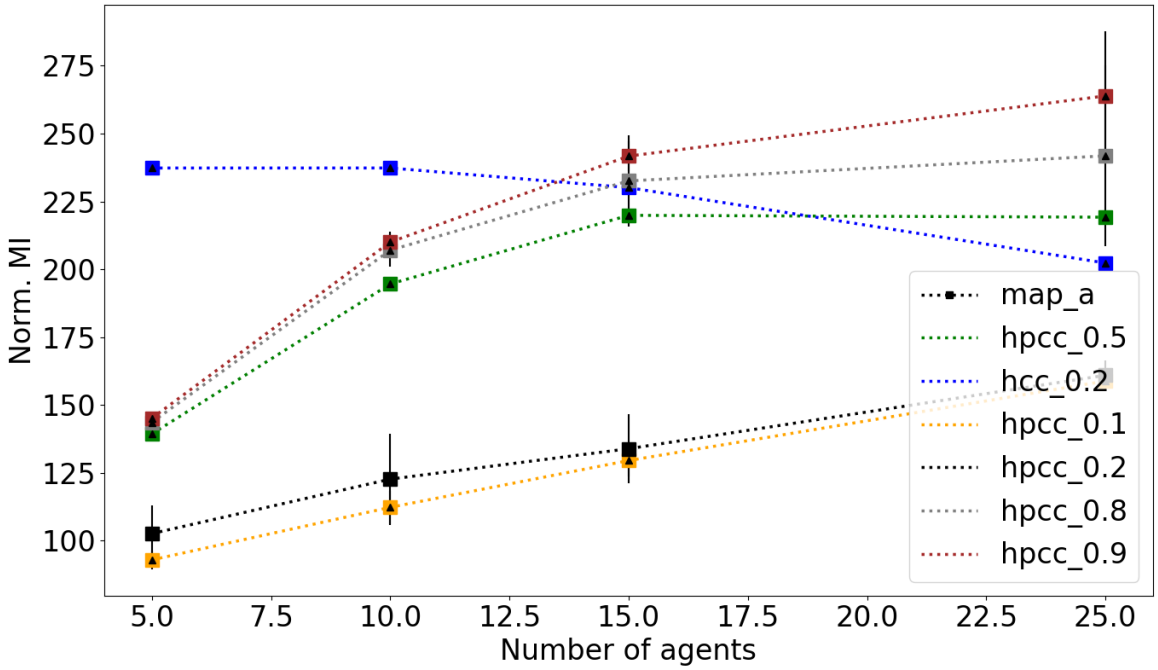

Fig. B.5 Normalised MI, averaged over 100 executions, of different variants of HPCC on the A topology.

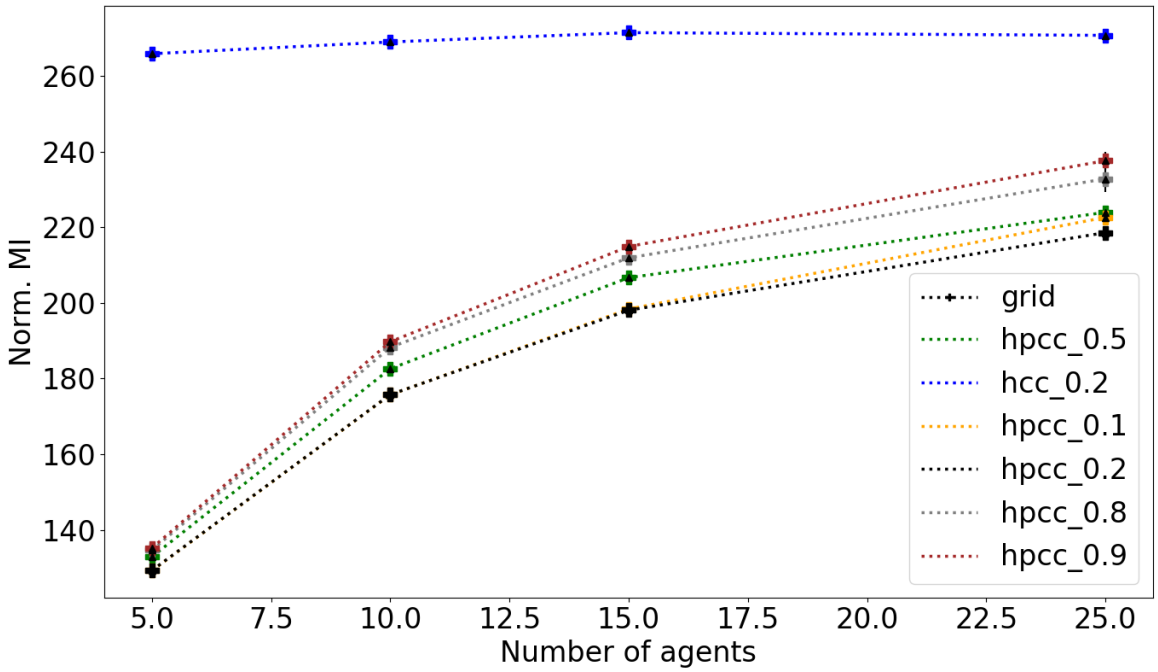

Fig. B.6 Normalised MI, averaged over 100 executions, of different variants of HPCC for 5, 10, 15 and 25 agents on the Grid topology.

B.1.3 Quadratic Mean Interval (QMI)

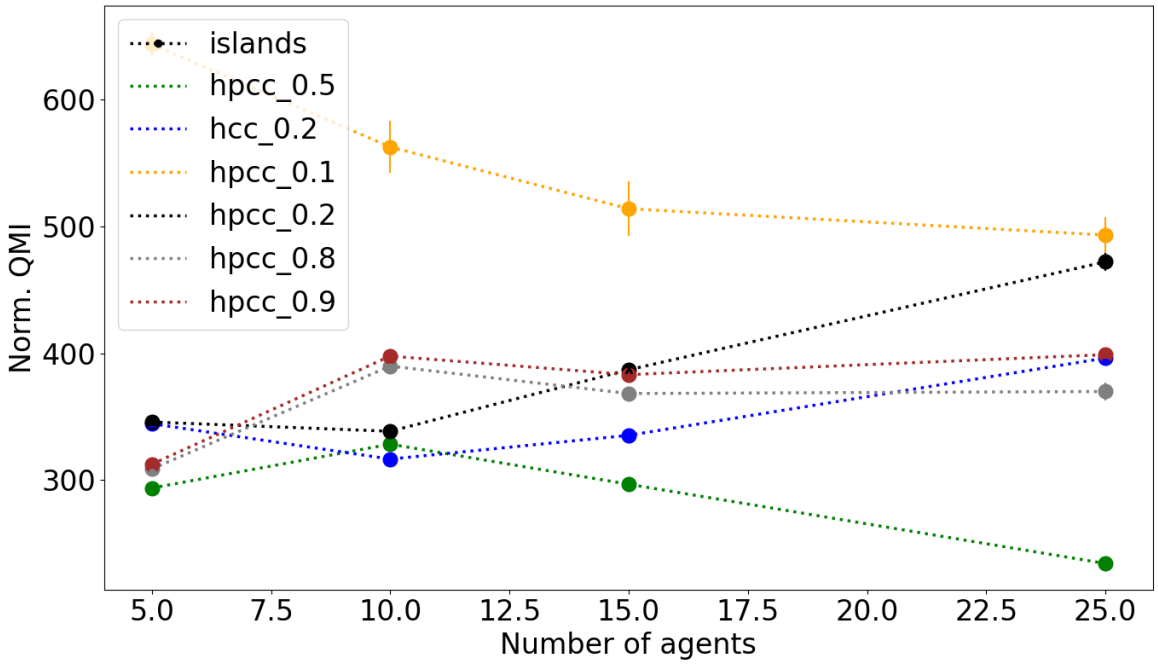

Fig. B.7 Normalised QMI, averaged over 100 runs, of different HPCC variants for 5, 10, 15 and 25 agents on the Islands topology.

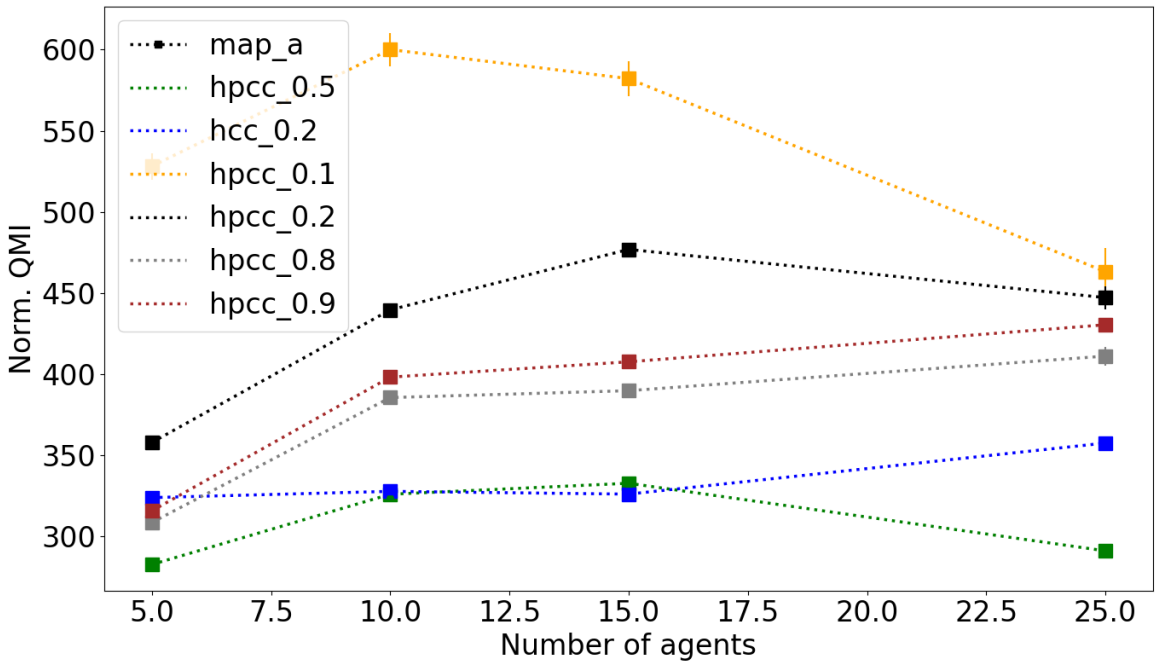

Fig. B.8 Normalised QMI, averaged over 100 runs, of CR and different HPCC variants for 5, 10, 15 and 25 agents on the A topology.

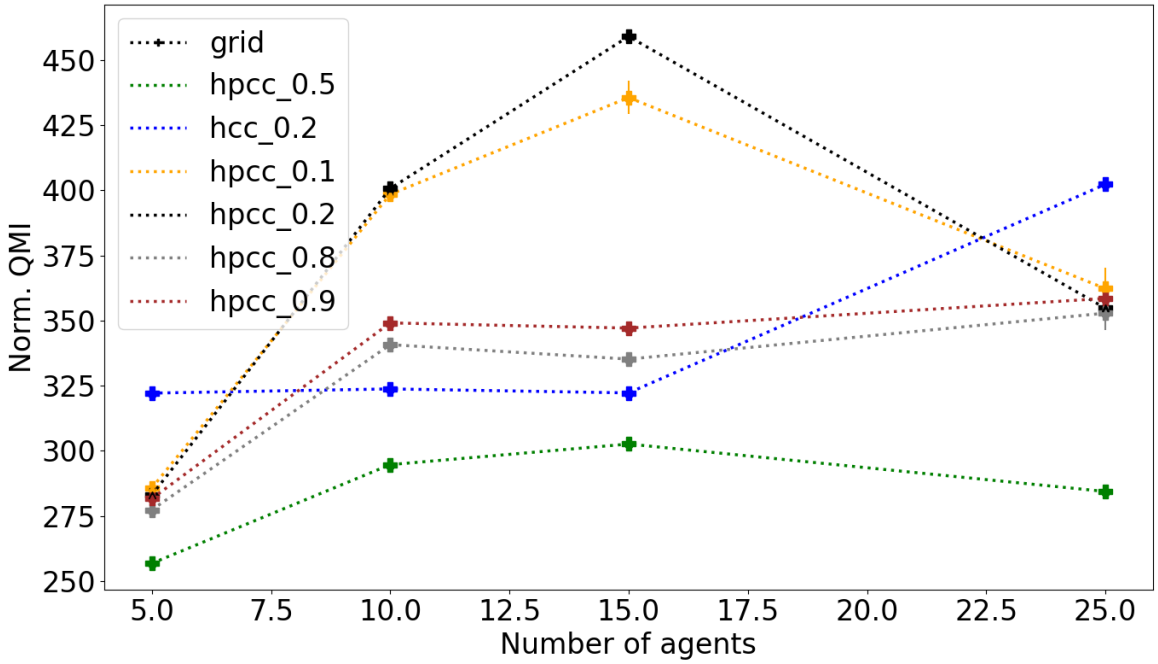

Fig. B.9 Normalised QMI averaged over 100 runs, of different HPCC variants for 5, 10, 15 and 25 agents on the Grid topology.

B.1.4 Worst Idleness (WI)

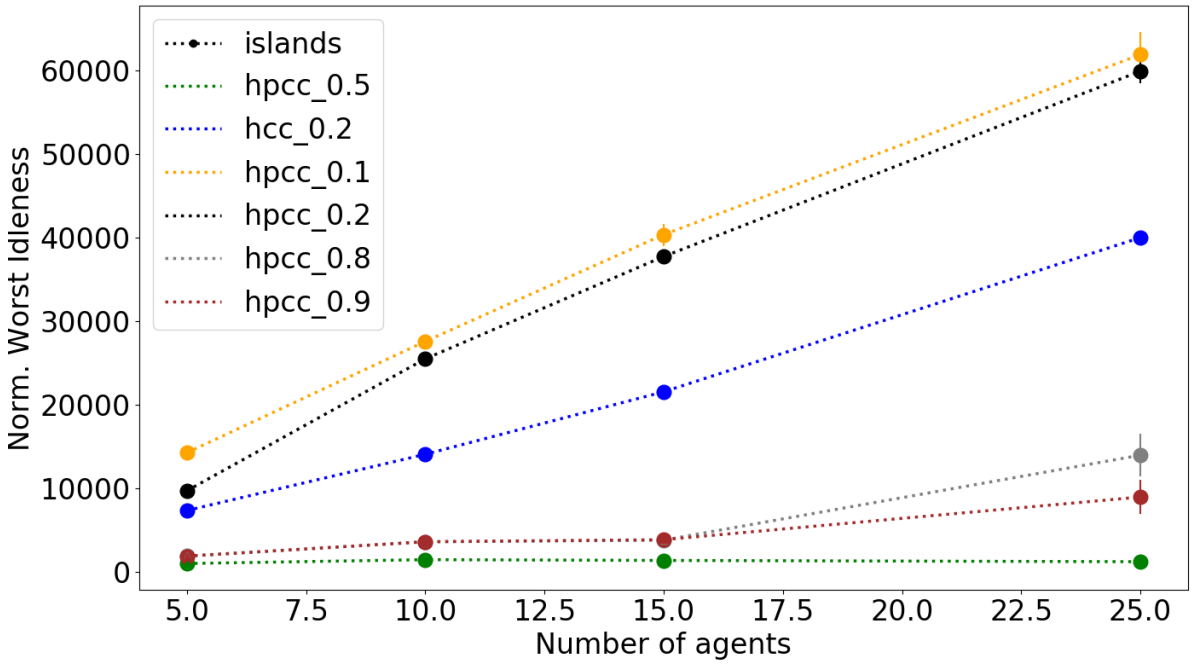

Fig. B.10 Normalised WI, averaged over 100 runs, of different HPCC variants for 5, 10, 15 and 25 agents on the Islands topology.

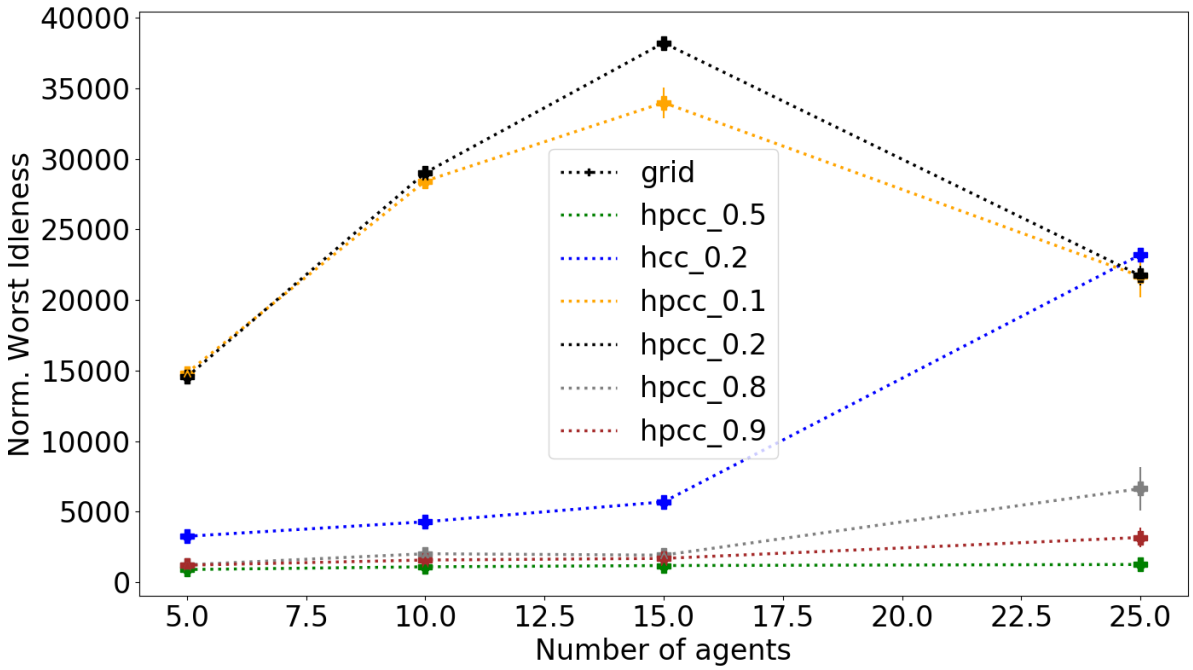

Fig. B.11 Normalised WI, averaged over 100 runs, of different HPCC variants for 5, 10, 15 and 25 agents on the A topology.

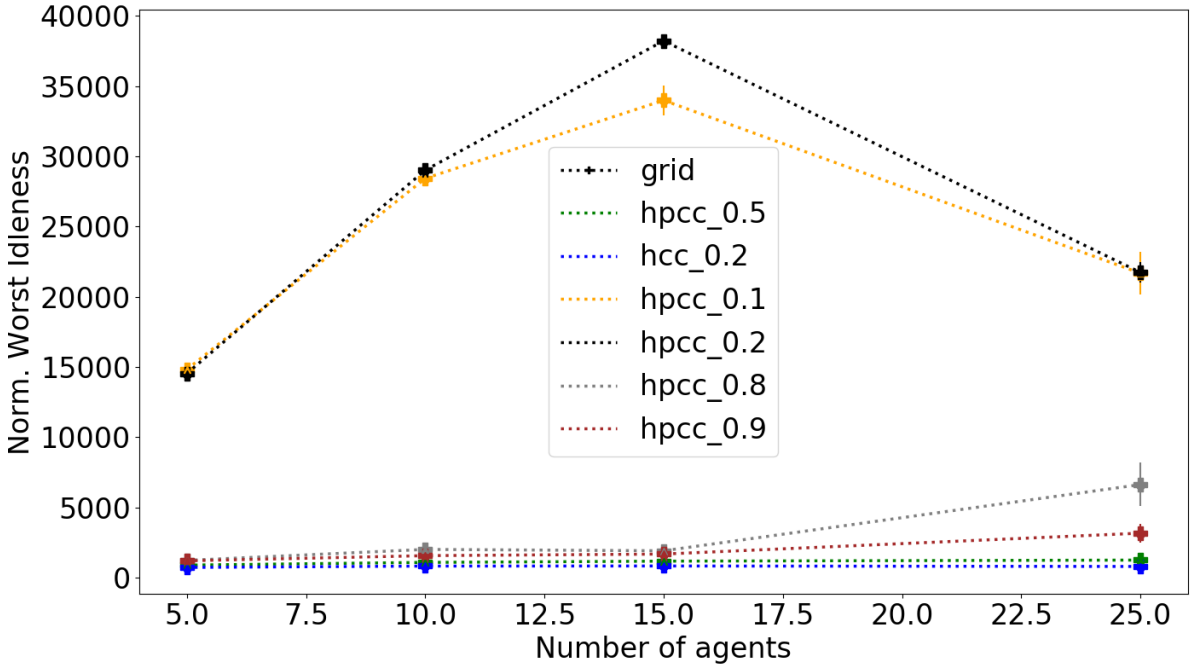

Fig. B.12 Normalised WI, averaged over 100 runs, of different HPCC variants for 5, 10, 15 and 25 agents on the Grid topology.

B.2 Training performances

B.2.0.1 Topology-guided-initialised (2-50)-LSTM

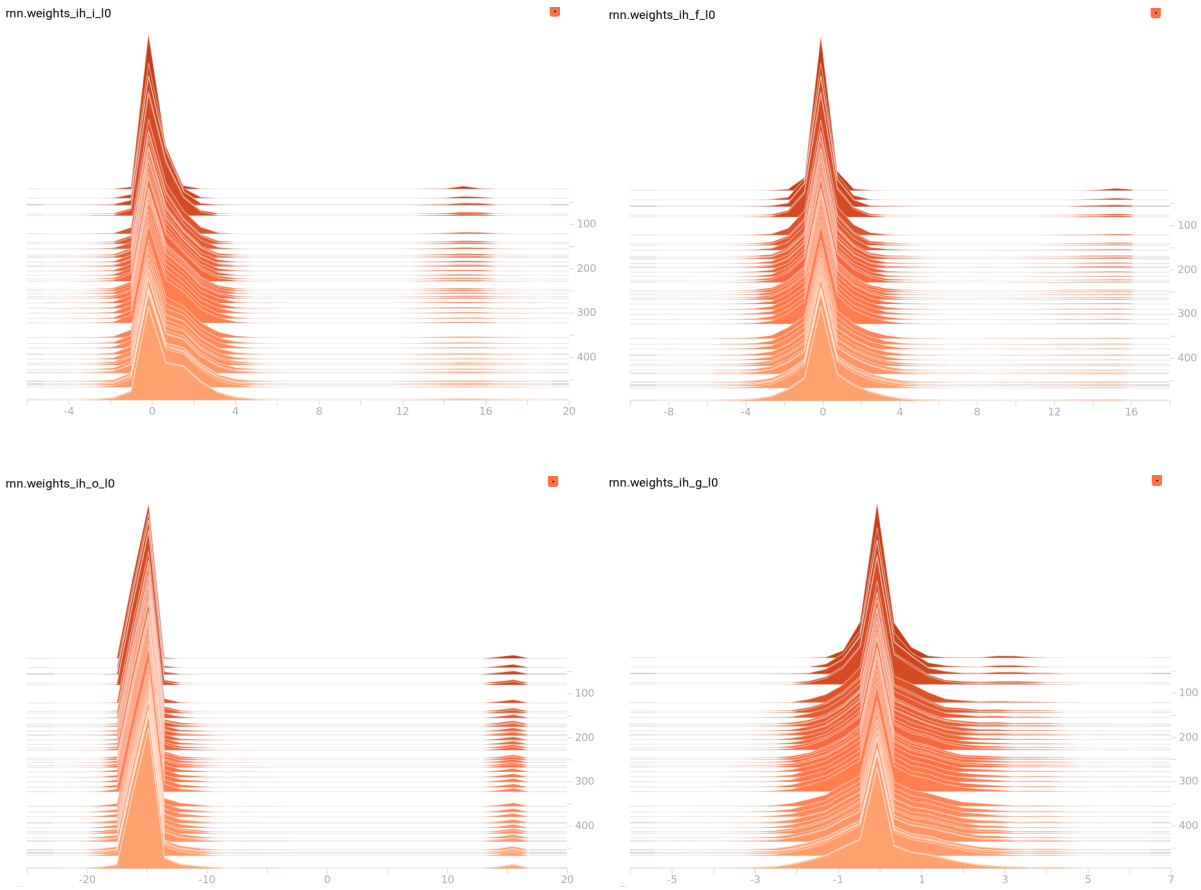

Fig. B.13 Distribution during the training of the weights of the input of x_t, upon the input (i_t), forget (f_t), output (o_t) gates and the cell state (g_t), respectively, for the $(2, 50)$-LSTM network trained on {HPCC 0.5, A, 15}

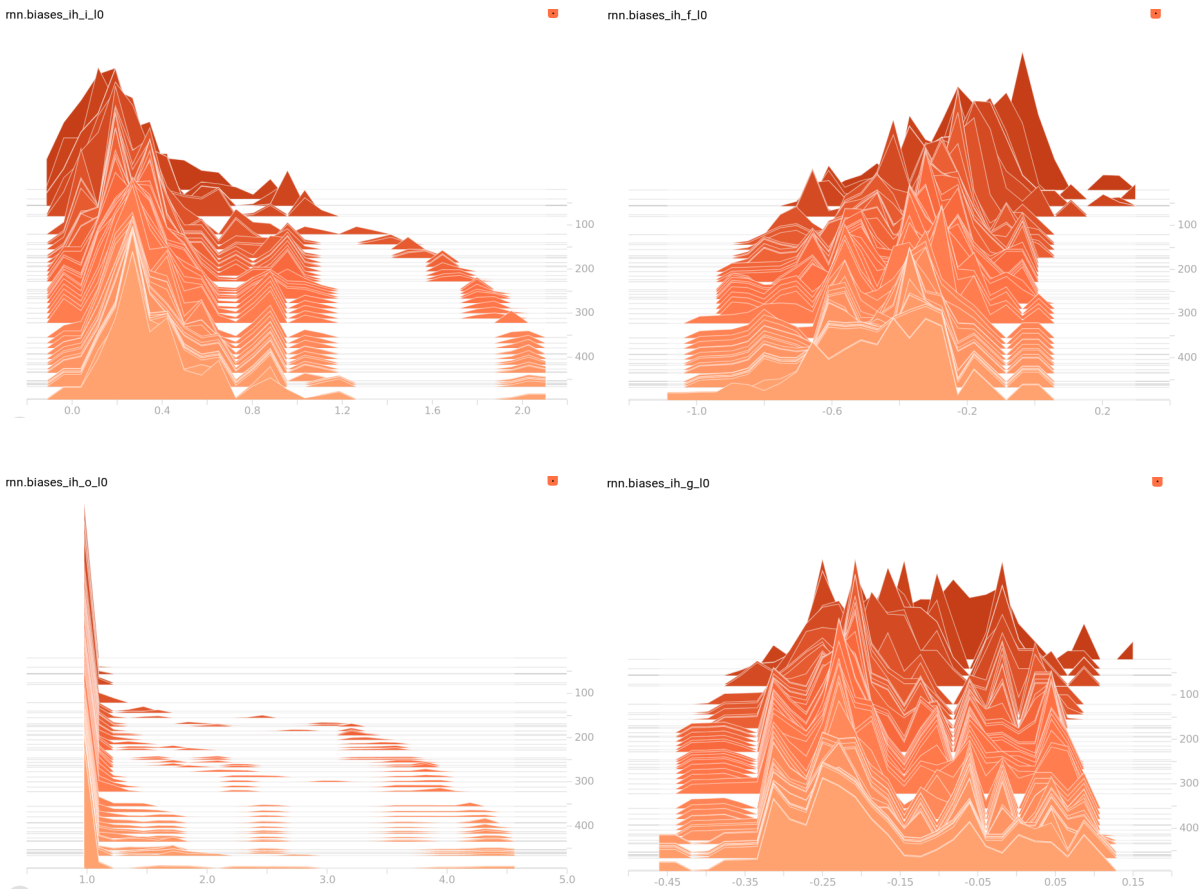

Fig. B.14 Distribution during the training of the biases of the input x_t, upon the input (i_t), forget (f_t), output (o_t) gates and the cell state (g_t), respectively, for the $(2, 50)$-LSTM network trained on {HPCC 0.5, A, 15}

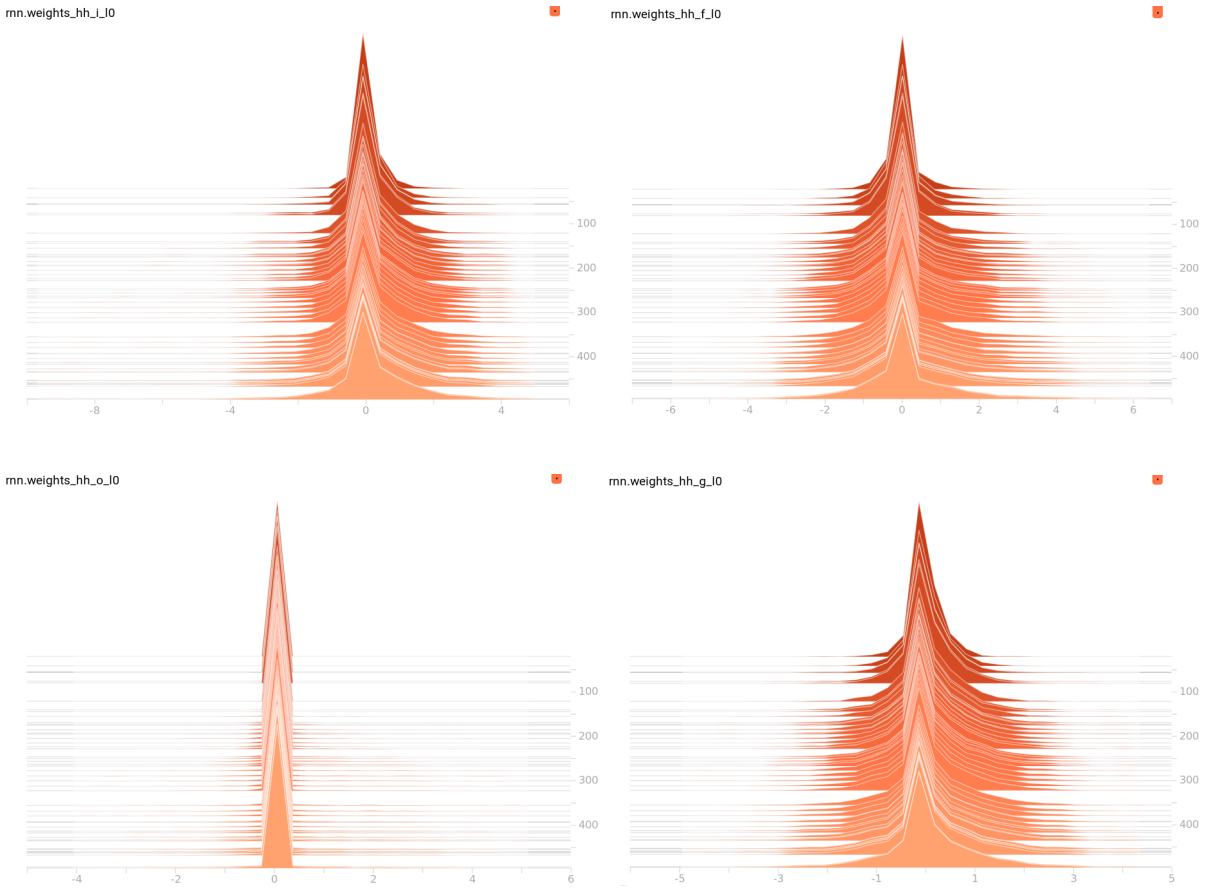

Fig. B.15 Distribution during the training of the weights of the input h_t, upon the input (i_t), forget (f_t), output (o_t) gates and the cell state (g_t), respectively, for the $(2, 50)$-LSTM network trained on {HPCC 0.5, A, 15}

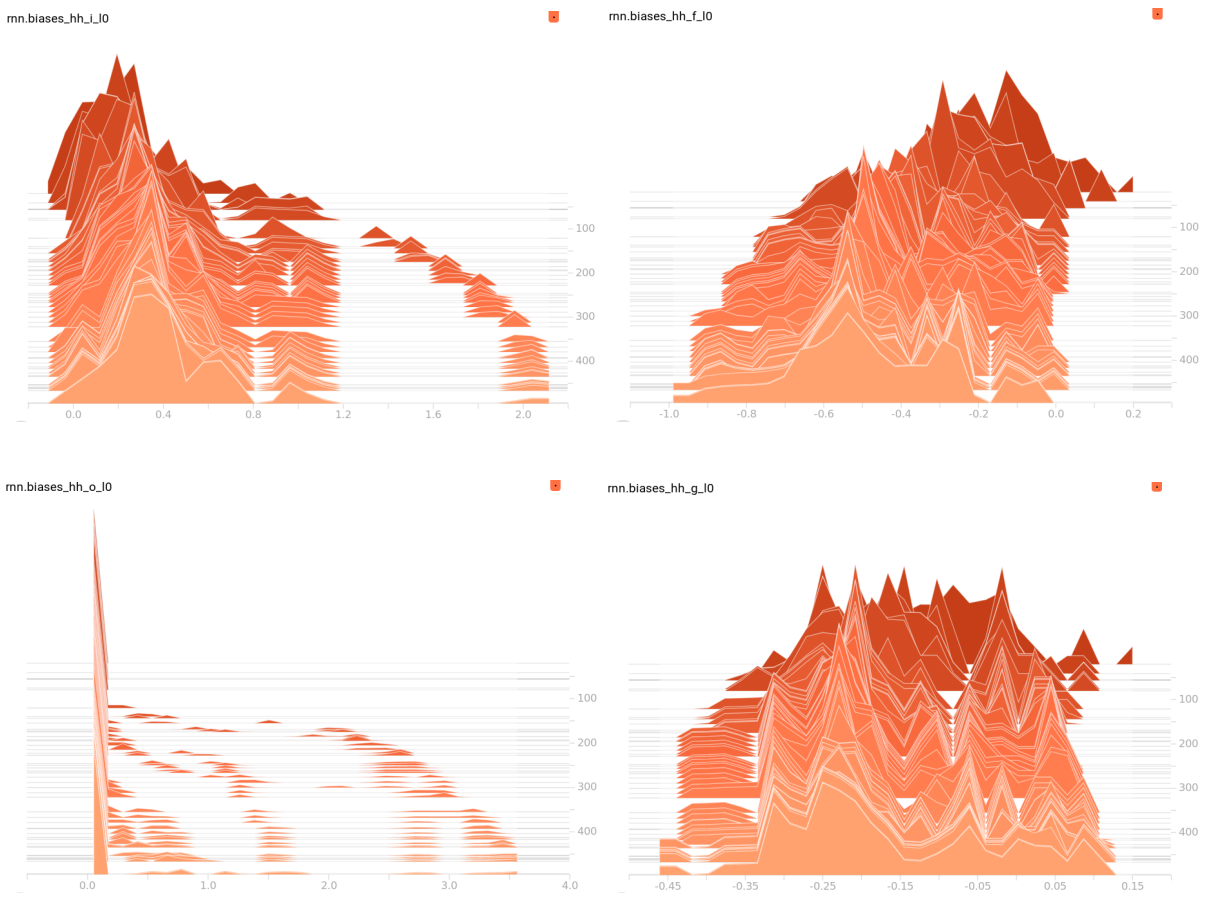

Fig. B.16 Distribution during the training of the biases of the input h_t, upon the input (i_t), forget (f_t), output (o_t) gates and the cell state (g_t), respectively, for the $(2, 50)$-LSTM network trained on {HPCC 0.5, A, 15}