Path Generation with LSTM Recurrent Neural Networks in the context of the Multi-agent Patrolling

Mehdi Othmani-Guibourg, Amal El Fallah-Seghrouchni, Jean-Loup Farges

Path Generation with LSTM Recurrent Neural Networks in the context of the Multi-agent Patrolling

Mehdi Othmani-Guibourg
ONERA, Toulouse, France
Sorbonne Université - Faculté des Sciences
Emails: Mehdi.Othmani-Guibourg@onera.fr
Mehdi.Othmani@lip6.fr

Amal El Fallah-Seghrouchni
Sorbonne Université - Faculté des Sciences
CNRS, UMR 7606, LIP6
F-75005, Paris, France
Email: amal.elfallah@lip6.fr

Jean-Loup Farges
ONERA, Toulouse, France
Email: Jean-Loup.Farges@onera.fr

***Abstract*—We propose a conceptually simple new decentralised and non-communicating strategy for the multi-agent patrolling based on the LSTM architecture. The recurrent neural networks and more specifically the LSTM architecture, as machines to learn temporal series, are well adapted to the multi-agent patrol problem to the extent that they can be viewed as a decision problem over the time. For a given scenario, a LSTM neural network is first trained from data generated in simulation for that configuration, then embedded in agents that shall use it to navigate through the area to patrol choosing the next place to visit by feeding it with their current node. Finally, this new LSTM-based strategy is evaluated in simulation and compared with two representative strategies, a cognitive and centralised one, and a reactive and decentralised one. Preliminary results indicate that the proposed strategy is globally not better than the representative strategies for the aggregating criterion of average idleness, but better than the decentralised representative for the evaluation criteria of mean interval and quadratic mean interval.**

***Index Terms*—Multi-agent systems, Multi-agent coordination, Multi-agent patrolling, Artificial Neural Networks, Long Short-Term Memory**

I. Introduction

Multi-agent patrolling (MAP), also known as multi-agent patrol task, is a generic task modelled as a multi-agent system where agents must visit as soon as possible different places, in order to protect or supervise them. There are a wide variety of problems that may be reformulated as particular multi-agent patrolling. As a concrete example, the task of monitoring an area by a swarm of patrolling drones in order to detect the presence of intruders or any other event such as, for example, a start of fire in a forest. Beyond drones patrolling an area, performing task efficiently can be useful for various application domains where distributed surveillance, inspection or control are required. For example, MAP could be useful, for detecting recently modified or new web pages to be indexed by search engines, or even for distributing computations over calculators.

For over fifteen years different types of strategies were proposed: centralised [13], emergent [13], idleness-based [13], heuristic (idleness and distance) with pathfinding [1], hamiltonian-cycle-based [5], TSP-heuristic-based [4], reinforcement-learning-based [6] and even auctions-based [7] strategies. Until now, few works concentrate upon the problematic of using Artificial Neural Networks (ANNs) for the multi-agent patrolling and none tackles the advantages that may be afforded by deep artificial neural networks in order to outperform the previous strategies. *This paper thereupon, proposes a first model regarding the use of the Long Short-Term Memory (LSTM) architecture as part of the multi-agent patrolling problem. In this way, a new strategy based on the LSTM architecture is introduced. Finally, the performances are evaluated according to the original and aggregating evaluation criterion used until now in this field of study, namely the average idleness.*

The Section II presents the background on multi-agent patrolling and the LSTM networks useful to understand proposed developments as well as previous works using ANNs as part of the multi-agent patrolling. Then, Section III introduces LSTM Path-Maker, the new strategy for the multi-agent patrolling based on the LSTM architecture. In Section IV the new strategy is analysed. Finally, Section V draws some conclusions and indicates directions for further works.

II. Background

This section presents the background on multi-agent patrolling and the LSTM architecture.

A. Multi-agent patrolling

1) Formal definition: The MAP model consists formally of a society of agents noted $\mathbf{A}$, able to move in an environment with the same mobility parameters, and a graph noted $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ representing a discretisation of the area to patrol. Here, $\mathbf{V} = \{1, .., N\}$ is the set of nodes identified by their indexes and standing for the places to visit. $\mathbf{E}$, is the set of edges of $\mathbf{G}$ accounting for the paths between the places. With each edge $\{v, w\}$ corresponds a *transit time* $c_{v,w}$ representing the travel time of the edge $\{v, w\}$. At the beginning of an instance of a patrol task, agents are positioned on nodes of $\mathbf{G}$. To each node is associated a dynamic variable named *idleness*, indicating the time elapsed since it has not been visited by any agent [4]. The idleness of a node v at time t, noted $i_t(v)$, is defined as being the amount of time elapsed since that node has received the visit of an agent. The idleness of all nodes at the beginning of the patrolling task is set to 0. Finally, each time an agent arrives at a node v, it shall decide, among the edges including v, the next edge to travel.

2) Strategies: A *strategy* of agent is an information processing method, or algorithm, allowing each agent to take a decision each time it arrives at a node. In the MAP, whatever the strategy considered, each agent intends actions based on its appropriated perceptions from the environment and its knowledge about idlenesses of nodes. Indeed, agents make idleness estimates that can be produced assuming different hypotheses. Two extreme hypotheses are:

- *individual idleness*: each agent considers only its own visits to reset its estimated node idleness. It corresponds to the case where communication between agents is not possible. In case of a mission with only one agent, *individual idleness* corresponds to *real idleness*, also called *global idleness*.
- *shared idleness*: all agents consider visits of all agents to reset estimated node idleness. In the case of perfect instantaneous communication between agents, or a mission with only one agent, *shared idleness* corresponds to *real idleness*, also known as *global idleness*.

Thus, some strategies of agent can be viewed as decision procedures divided into two stages: first the *information processing* stage corresponding to the retrieval and processing of the information about the environment, and then, the *decision-making* stage wherein the agent makes a decision with regard to its new state of knowledge produced in the previous stage.

Among the wide family of strategies, two are relevant for this work as representative strategies: *Conscientious Reactive* (CR) and *Heuristic Pathfinder Cognitive Coordinated* (HPCC).

CR consist of selecting the next node to visit as the one with the highest idleness in its neighbourhood. There is no communication between agents: idlenesses are estimated by each agent on the basis of its own path. CR can be thought of as a good representative and thereby a comparison strategy for the reactive and decentralised ones.

For HPCC, there is a perfect communication between agents: idlenesses are estimated by a coordinator on the basis of all paths of agents. The decision process includes two steps:

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

The selection of the target node takes 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 $\mathbf{V}$ corresponds here to the shortest path between these two nodes. Idleness and time to go are normalised by scaling them between 0 and 1. A zero normalised value is attributed to the maximum idleness (edges with high idleness values shall be traversed) whereas a value equal to 1 is attributed to the minimum idleness. Intermediary values are calculated by means of proportions as shown in **(1)**:

$$\begin{aligned} &\text{If } min_{v\in\mathbf{V}}\{i_t(v)\} \neq max_{v\in\mathbf{V}}\{i_t(v)\}, \forall v_0 \in \mathbf{V} \\ &\bar{i}_t(v_0) = \frac{max_{v\in\mathbf{V}}\{i_t(v)\} - i_t(v_0)}{max_{v\in\mathbf{V}}\{i_t(v)\} - min_{v\in\mathbf{V}}\{i_t(v)\}} \end{aligned} \tag{1}$$

where $i_t(v)$ and $\bar{i}_t(v)$ are the global and normalised idleness, respectively.

Normalised time to go is calculated similarly. For that purpose, at the minimum time to go is attributed a zero normalised value (edges with short distances shall be traversed) whereas at the maximum time to go is attributed a value equals to one. Intermediary values are calculated by means of proportions as shown in **(2)** :

$$\begin{aligned} &\forall d(v_0, v) \text{ a time to go from } v_0 \text{ to } v, \\ &\bar{d}(v_0, v) = \frac{d(v_0, v) - min\{d\}}{max\{d\} - min\{d\}} \end{aligned} \tag{2}$$

where $max\{d\}$ and $min\{d\}$ are the maximum and the minimum time to go respectively, over all the $v, w \in \mathbf{V} : v \neq w$.

Finally, for an agent at the position v_0 at time t, the values associated to nodes are given by **(3)**:

$$\begin{aligned} &\forall r_H \in [0,1], \forall v \in \mathbf{V}, val_{r_H}(v,t) = \\ &\qquad r_H \times \bar{i}_t(v)) + (1 - r_H) \times \bar{d}(v_0, v)) \end{aligned} \tag{3}$$

where the weighting factor r_H must be chosen by the strategy designer. Minimising the node values according to that expression i.e. selecting the nodes with the minimum value, allows agents to visit nearby nodes with higher idleness first and foremost. Moreover, there is a mechanism forbidding the coordinator to select nodes that are currently assigned to other agents.

The path computation takes into account the idleness of the nodes between the current location and the goal to compute the best path leading there. For that, it weights the edges as shown in **(4)** :

$$\begin{aligned} &\forall r_P \in [0,1], \forall e \in \mathbf{E} : e = \{v, w\}, \\ &\qquad c_{r_P}(e) = r_P \times \bar{i}_t(w) + (1 - r_P) \times \bar{c}_{v,w} \end{aligned} \tag{4}$$

where the weighting factor r_P must be chosen by the strategy designer. In that case, it is the normalised transit time $\bar{c_{i,j}}$ of edge and not the normalised time to go $\bar{d}(i,j)$ of path that is used to value edges.

Minimising the edge weights according to that expression allows agents as well to visit nearby nodes with higher idleness first and foremost.

HPCC as a communicating, fully-informed, coordinated, and thereby centralised strategy is one of the best online - namely without pre-calculation of paths - strategy. It can be then regarded as a representative and thereupon a comparison strategy for the coordinated and centralised ones.

3) Evaluation criteria: Graph idleness, also known as *average idleness*, noted I_{av} or $I_{\mathbf{G}}^{T}$, if it does not lead to confusion, noted $I_{\mathbf{G}}^{T}$, measures the average idleness over all the nodes of $\mathbf{G}$ and the duration T, as follows:

$$I_{\mathbf{G}}^{T} = \frac{1}{|N| \times T} \sum_{t \geq 0}^{T} \sum_{v \in V} i_t(v) \tag{5}$$

Sampaio et al. [16] introduced evaluation criteria relevant to establish aggregation measures not based on idleness but on the intuitive concept of *interval between visits* to a node. In this class of evaluation criteria, the size of intervals between visits for each node is calculated by registering the value of idleness just before an agent's visit. All intervals for all nodes are used to make an aggregated calculation. Two main interval-based evaluation criteria are the *Mean Interval* (MI) and the *Quadratic Mean Interval* (QMI), the mean and the root-mean-square respectively, on all intervals between visits of a mission execution. The latter penalises strategies that leave nodes unvisited, or which produces wide intervals between visits during the simulation and provides an additional precision upon the distribution of visits over the nodes: one node with wide intervals have a little impact upon MI while it has upon QMI.

Finally, these two criteria are relevant insofar as the average idleness of a simulation execution I_G^T can be defined from those as follows:

$$I_{\mathbf{G}}^{T} = \frac{N_{Int}}{2NT}(QMI^2 + MI) \tag{6}$$

with N_{Int} the total number of intervals during the simulation execution.

In order to better evaluate the contribution of each agent when the population size varies, these evaluation criteria are normalized by multiplying values by the number of agents.

B. LSTMs

Recurrent Neural Networks (RNNs) are neural networks that process an input sequence one element at a time, 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. Classical feed-forward networks cannot perform tasks depending upon sequential information. With RNNs the relevant sequential information is preserved in their hidden state due to the memory that enables to find correlations between events separated by several time steps. This memory is contained in the *hidden layers* which have a *feedback loop*, and therefore they constitute *recurrent layers*.

Generally, the main purpose of this kind of architectures is to learn long-term dependencies but theoretical and empirical evidences show that it is difficult to learn to store information

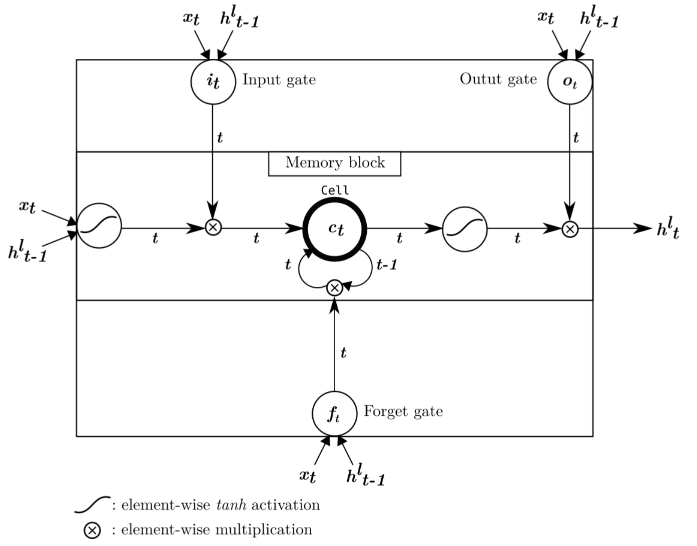

Fig. 1: Layered LSTM unit: the core composant of the LSTM architecture

for very long time [2]. A correction for that was to augment the network with specialised memory cells. The first proposal in that direction has been the *Long Short-Term Memory* (LSTM) networks that use a special hidden unit, with the natural behaviour to remember inputs for a long time.

LSTM are a special kind of RNN introduced and designed to take into account the long-term dependencies. They have the same general chain structure as the classical RNNs except that the repeating module has a different structure more complicated as shown in **Fig. 1**. In the first place, as stated by Hochreiter et al. [11], an LSTM network was a recurrent neural network with one input layer, one fully self-connected hidden layer containing purpose-built *memory cells*, *gate units*, and an output layer. Originally, an LSTM's hidden layer was assumed to have a dense connectivity: each gate unit and each memory cell have seen all non-output units [11]. This memory unit corresponds to a neuron with a recurrent self-connection. Thus, a cell referred originally to an object with a single scalar output. The activations of those neurons within the memory units compose the *state* noted c_t, sometimes called *cell state*, of the LSTM network.

As stated by Graves et al. [9] an *LSTM layer* consists of a set of recurrently connected blocks, known as *memory blocks*, which in turn consists of cells. One cell, as neuron, outputs one scalar. Originally, each memory block has contained one or more layered recurrently connected neurons called memory cells and sharing the same three multiplicative units: i_t the *input gate*, o_t the *output gate* and f_t the *forget gates*, i.e. all the cells of a memory block are connected to the same gate units. The gate units provide continuous analogies of write, read and reset operations for the cells. A memory block of size 1 is then a simple memory cell [11] connected to $tanh$ activations. 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. Besides, the memory block and the gates form the *LSTM unit* as shown in the **Fig. 1**, 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 base RNN model in which they cover 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 and the latter the output of the hidden layers.

Also, 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 (for a multi-layer LSTM) noted h_t^l, respectively c_t^l.

For some years and hitherto, most implemented LSTM architectures contain only one cell in their LSTM units. The LSTM units of a same layer can thereupon be "layered" into only one LSTM unit 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 turn 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 thereby 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 then computed from the following composite function:

$$i_t^l = \sigma(W_{xi}^l\ x_t^l + W_{hi}^l\ h_{t-1}^l + b_i^l) \tag{7}$$
$$f_t^l = \sigma(W_{xf}^l\ x_t^l + W_{hf}^l\ h_{t-1}^l + b_f^l) \tag{8}$$
$$o_t^l = \sigma(W_{xo}^l\ x_t^l + W_{ho}^l\ h_{t-1}^l + b_o^l) \tag{9}$$
$$c_t^l = f_t^l * c_{t-1}^l + i_t^l * \tanh(W_{xc}^l\ x_t^l + W_{hc}^l\ h_{t-1}^l + b_c^l) \tag{10}$$
$$h_t^l = o_t^l * tanh(c_t^l) \tag{11}$$

The parameters of an LSTM layer that must be learned for a layer l are thereby:

- $W_{xi}^l, W_{hi}^l, W_{xf}^l, W_{hf}^l, W_{xo}^l, W_{ho}^l, W_{xc}^l$ and W_{hc}^l
- b_i^l, b_f^l, b_o^l and b_c^l

The structure corresponding to several memory blocks in a layer l can be derived from its more general architecture by setting the elements of W_{hi}^l, W_{hf}^l, W_{ho}^l which are not block-diagonal to 0.

It must be pointed out that the *deep recurrent neural networks* i.e. recurrent neural networks with several layers were investigated for the first time by studying the performances of the *deep LSTMs* [10]. Deep LSTMs combine the multiple levels of representation that have proved so 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 presented previously apart from the fact that there are several LSTM layers.

C. Related works

Guo et al. [8] studied the use of neural network-based methods for planning a complete coverage patrolling path. In that work, the area to patrol is discretised into fixed radius disks corresponding to the coverage range of robot's sensor. Every disk can then be thought of as a node to visit. The neural network used in this work is non-hierarchical to the extent that each neuron encoding a specific region, has an output into, and receives an input from, each one of its neighbours. Each neuron as a state variable, represents a region and is activated negatively or positively, in function of presence of obstacle or non-coverage by the patrol, respectively. The activities of all these neurons compose a dynamic landscape such that the non-visited areas globally attract the robot in the entire space, and the obstacle locally repel the robot to avoid collisions.

Sales et al. [15] developed an autonomous patrolling system composed of four intelligent robots that can freely move through an indoor environment and detect intruders. The robots are endowed with a localisation/navigation system composed of an artificial neural network (ANN) used in combination with a Finite State Machine (FSM) of which the states correspond to the key features of the environment. The FSM associates a sequence of actions to execute with a sequence of states. The ANNs process the sensors' data to identify and classify the FSM states to determine the actions to perform. After being trained offline to identify the key features of the environment such as corridors, crossroads and turns, they are fed into data obtained from robots' sensors, and they output the FSM states. Finally, each robot calculates the shortest path by using A* to reach the enemy's position with taking into account its teammates, which have informed it when they detected the intruder.

III. An LSTM-based strategy

In this work we specifically created and assessed one LSTM-based strategy named *LSTM Path Maker* (LPM). The LSTM network was trained over simulation traces of a fully-informed, coordinated and communicating strategy: HPCC. Finally, it was tested and compared to HPCC and CR.

This section presents our contribution, that is LPM, a new LSTM-network-based decentralised agent strategy, which learns to navigate the nodes composing the area to patrol, from series of histories collected from numerous simulation executions of a fully-informed and coordinated strategy: the HPCC strategy. The main goal of this work as well as our first assumption was that whether agents learn in average, to behave similarly to the coordinator which have all information over the area (the shared idleness of nodes and the real position of agents), by using an LSTM network, then agents may approach performances reached by the coordinated strategy.

A. Formal definition

The LPM strategy is an ANN-based strategy: the decision-making process is carried out by means of an LSTM network which outputs the next node from the current node given as input of the network. This strategy can be thought of as a reactive strategy that uses an artefact for guidance through the area to patrol, such as a compass, which takes implicitly into account the idleness of nodes and the agents' positions. In our context, the temporal series representing the successive visited nodes by an agent is called a *path*. Any path's vertex, has one of its neighbours for next vertex.

For a given scenario, the LSTM network temporally learns the next node to visit v_{t+1} from a *model strategy*, according to the previous ones $v_t, v_{t-1}, ..., v_0$ in the path and that for all paths regardless of the path: each path, as a temporal series accounting for the path of an agent, is fed into the network node after node. It follows that, with defining f as the decision procedure of the model strategy - and thereby the strategy itself -, the LSTM network of the scenario that approximates f can be defined as follows:

Let $I_t = \{i_t(v_0), i_t(v_1), ..., i_t(v_N)\}$ being the set of shared idlenesses at the time t and v_a the node from which a next node to visit, noted $\bar{v}$, must be selected as a decision process, by an agent a. Then, the next node to visit $\bar{v}$ will be selected from the procedure f, the *requesting node* v_a and the set of shared idlenesses I_t such as:

$$\forall t \in \mathbf{T}, \bar{v} = f(v_a, I_t) \tag{12}$$

Thus, with considering $\tilde{f}$ the vertex-purpose-LSTM-network-based decision procedure as a function approximator of f, and $v_t \in \mathbf{V}$ the node visited by an agent a at the time $t \in \mathbf{T}$, we have:

$$\forall t \in \mathbf{T}, v_{t+1} = \tilde{f}(v_t, ..., v_1) \tag{13}$$

This equation pertains to the formal definition of an LSTM network: each output depends on the previous outputs.

Let N the number of nodes in the graph. With the aim of feeding the LSTM network with the most appropriate and relevant information about the nodes, each node has been encoded as an N-dimensional one-hot vector: for the vertex v_i all the coordinates of the vector will have 0 as value except the i-th coordinate which will have the value 1. The output of the network thereupon, is an N-dimensional vector whose the values are in $[0, 1]$; these values can be regarded as probabilities. Thus, to ensure that all values are positive and their sum equal to one, the output layer of the networks is a softmax layer. The node represented by the maximum output vector's coordinate will be selected as the next one to visit.

Let (L, H) be the *profile of parameters* of an LSTM architecture so that L and H stand for the number of layers and the number of hidden units (or memory cells) per layer, respectively, of a given LSTM architecture. Formally, by defining $b : \mathbf{V} \to \mathbf{V_{bin}}$ as being the function mapping all the indices of nodes into their one-hot representation, the proposed architecture can then be described with:

$$x_t^1 = b(v_t) \tag{14}$$

$$\text{If } L > 1, \forall l \in \{1, ..., L-1\} \; x_t^{l+1} = h_t^l \tag{15}$$

$$L_{net} = softmax(W \cdot h_t^L) \tag{16}$$

where $dim(h) = H$ and W is a $card(V) \times H$ matrix of parameters.

Finally, upon training stage's completion, each LPM agent will be endowed with the same parametrised LSTM network.

B. Network training

The training of the LSTM network is performed from the logged paths of any high-performance strategy f. Generally, the high-performance strategies make use of communications and centralised decision-making process. The purpose here, is then to approach the performances of these strategies without communicating and thereupon *distributing* and *decentralising* the decision-making process. Indeed, for example in the context of a drones' reconnaissance mission or even silent bots penetrating a network, communications may be impossible or discouraged. Such a strategy to learn will be called the *model strategy* or simply the *model* if that does not lead to confusion.

For each *scenario* $\{f, \mathbf{G}, N_a\}$, also called *simulation configuration* or simply *configuration*, with N_a the number of agents, whether it does not cause any confusion, the LSTM network is first pre-trained with the purpose of learning the topology i.e. the structure of the graph representing the area. Thereafter, the network is trained over all the paths retrieved from the executions of configuration for $\{f, \mathbf{G}, N_a\}$, 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 the sequence is a path of nodes; here the sequence modelling corresponds to a *path generation*.

As aforementioned in the **Subsection III-A**, the network's output layer is a softmax layer in order to return a probability distribution as output. Path generation aims at learning a probability distribution over paths by minimising the cross-entropy of a model given a set of N training sequences of length T:

$$\min_{\boldsymbol{\theta}} \; -\sum_{n=1}^{N} \sum_{t=1}^{T} log \; p(v_t^n | v_1^n, ..., v_{t-1}^n; \boldsymbol{\theta}) \tag{17}$$

where $\boldsymbol{\theta}$ is the set of the model's parameters, of dimension $dim(\boldsymbol{\theta}) = 4(2L-1)H^2 + (4L + 5Card(V))H$, and p is our predicted probability for the current element of the observed sequence (v_t^n).

C. Decision

Generally, despite the pre-training stage, the network may output the highest probability for a node that is not in the neighbouring of the one given in input. In doing so, the decision shall be made only among the output probabilities standing for the neighbour nodes. It follows that each time the one-hot vector of the current vertex is presented to the network, the decision procedure $\tilde{f}$ concerning the next node to visit consist of selecting the next node among the neighbours of the current vertex with the maximum probability. This can be mathematically rewritten as bellow:

Let:

- $\mathbf{V_{bin}} \subset \{0, 1\}^{card(V)}$ be the set of nodes formatted into one-hot vectors,

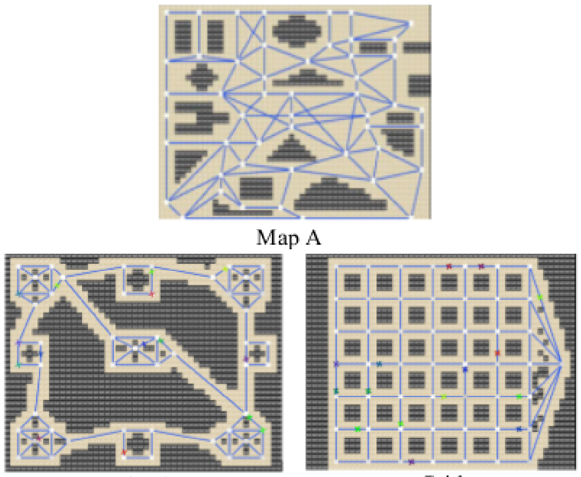

Fig. 2: Graphs used during assessment.

- $L_{net} : \{0,1\}^{card(\mathbf{V})} \to [0,1]^{card(\mathbf{V})}$ the function represented by the LSTM network used here,
- $Ng : [0,1]^{card(\mathbf{V})} \times \mathbf{V} \to [0,1]^{card(\mathbf{V})}$, the function setting to zero the values of the coordinates not corresponding to the neighbours of a given node's one-hot vector.

Then,

$$\begin{aligned} \forall t \in \mathbf{T}, \forall v_t \in & \mathbf{V_{bin}}, \\ v_{t+1} = & argmax(\ Ng(L_{net}(b(v_t)), v_t)\ ) \end{aligned} \tag{18}$$

It then follows that:

$$\begin{aligned} \forall t \in \mathbf{T} :\ & v_t \in \mathbf{V}, \\ \tilde{f}(v_t, ..., v_1) = & argmax(\ Ng(\ L_{net}(b(v_t)), v_t\ )\ ) \end{aligned} \tag{19}$$

with $\tilde{f}$ depending upon $v_1, ..., v_t$ due to their relevant features being stored in the memory of L_{net}.

Finally, the first experiments showed that using LSTM network as it stands, tends to lead agents to converge indefinitely towards a small set of nodes, leaving thereupon others nodes non-visited until the end of the execution. In doing so, the decision procedure was slightly improved: henceforth, with the aim to make the system more robust, the next vertex to visit from the current one will be randomly selected according to the distribution of probability output by the LSTM network, normalised over the neighbourhood of the current vertex using the Bayes' theorem over the distribution of neighbours. This new procedure enables therefore to add a little randomness in the decision process when selecting the next node in the neighbourhood, leading to increase the robustness of the system, and thereby to avoid agents to visit only a restricted set of nodes. This new resulting strategy was called *Random-Next-Neighbour-LSTM-Path-Maker*, abbreviated *RLPM*.

IV. Experiments and results

A. Scenarios

Following Othmani et al. [14], three topologies were selected to evaluate the new strategy: the maps *Islands*, *Grid* and *A*, as shown in the **Figure 2**. For each map we tested the strategies CR, HPCC with a value of 0.2 for r and RLPM was trained from HPCC's simulation with the same value for r as well. To that end we used *Pytrol*, a new Python multi-agent-patrolling-purpose simulator that has been specially designed for this purpose. These tests were performed over population sizes of 1, 5, 10, 15 and 25 agents and for each size 100 random starts, also called executions, were selected. For each start, each strategy was tested over 3000 time steps. Considering that each move takes exactly one time step in the proposed model, after excluding the moves upon edges, an agent visits in average 650 nodes during one execution of 3000 time steps. In doing so, the paths used to train the LSTM networks have approximately a length of 650 nodes.

B. Training results

For each scenario seven architectures: $(1,1)$, $(2,2)$, $(4,10)$, $(1,50)$, $(2,50)$, $(3,50)$, and $(50,2)$ were trained, with an end-to-end training i.e. a non truncated back-propagation through time. For any architecture we trained over 10000 epochs one LSTM network for each simulation configuration in two stages using the *Pytorch* library. First, the network is pre-trained over 2×10^6 epochs to capture as far as possible the structure of the topology from 2-length series which stand for the edges of the graph. Then, the network is trained during 10000 epochs over the paths of agents where the parameters are initialised with the values learnt during the pre-training stage. The **Figure 3** shows the initial and final values of the cost, that is the cross-entropy, during the validation stage for each architecture, averaged over the maps and numbers of agents. Here, the initial cost corresponds to the validation cost after the first epoch. This figure shows that the better networks are $(4,10)$, $(1,50)$ and $(2,50)$. Interestingly, the initial and final cost for the architecture $(50,2)$ are almost identical and it has the worst cost with a value of 3.87. This result tends to show that the network's parameters converged very quickly, namely in 1 epoch. The number of parameters for $(1,1)$, $(2,2)$ and $(50,2)$ are 258, 564 and 2484 respectively. It seems that those numbers are too low for a satisfactory approximation of the sequences. At the opposite, $(3,50)$ has 63100 parameters. It is likely that this number is too large to avoid over-fitting and a relatively bad performance in term of validation cost. Indeed, for one agent the size of the training data is approximately 50000, that is lower than the number of parameters of the $(3,50)$ network.

Considering the bad final validation costs of $(1,1)$, $(2,2)$ and $(50,2)$, of 3.03, 2.08, and 3.87 respectively, we tested and evaluated the four LSTM architectures $(4,10)$, $(1,50)$, $(2,50)$, and $(3,50)$. Thus, each architecture has given rise to four variants of RLPM named RLPM-*L*-*H*.

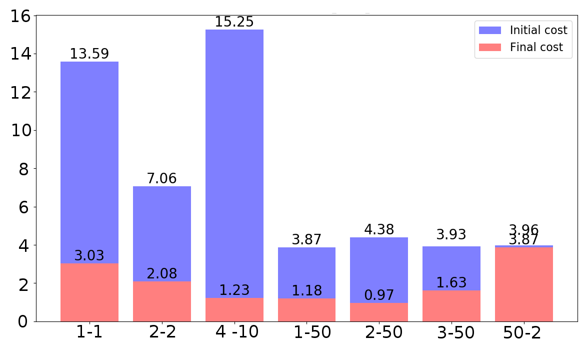

Fig. 3: Costs averaged over the maps and numbers of agents for each architecture

C. Performance results

To evaluate their performances, the RLPMs were tested and compared with CR, the reactive and decentralised strategy, and HPCC, the cognitive one wherefrom they were trained. We used the average idleness as evaluation criterion.

The **Fig. 4** shows the normalised average idleness for the topology Islands. For the sake of clarity, we showed the RLPM variant corresponding to the best value over this criterion, i.e. the RLPM variant with the lowest value. Not surprisingly, HPCC always outperformed all the other strategies for all the population sizes, except for 1 agent where CR is slightly better. Such an observation stems from the fact that CR being a decentralised strategy, for 1 agent the visits are better distributed over nodes, penalising the average idleness. In that topology, RLPMs have poor performances. They are always worse than HPCC and CR with values of 347 periods (RLPM-3-50) for 1 agent, 550 (RLPM-1-50) for 5 agents, 1078 (RLPM-4-10) for 10 agents, and 1793 (RLPM-1-50) for 15 agents, 1153 (RLPM-1-50) for 25 agents. Besides, the evolution of RLPMs' performances over the normalised I_{av} with respect to the number of agents shows a peak for 15 agents. This result suggests that from 1 to 15 agents the RLPMs do not exploit the advantage provided by additional agents. However, for 25 agents the performances are slightly better regarding this issue considering that there are 25 agents for 50 places to visit, namely 1 agents for 2 places.

The **Fig. 5** shows the normalised average idleness for the topology A. As well as for the **Fig. 4**, we showed the RLPM variant corresponding to the best value over this criterion. As previously, HPCC remains the best strategy. The best RLPM is better than CR by 60 periods in average with RLPM-1-50 for 5, 10 and 15 agents, with values of 339, 366 and 424 periods, respectively, but worse by 207 periods for 1 agent, with a value of 308 periods, and by 266 periods for 25 agents with RLPM-3-50, with a value of 789 periods. The results for 1 and 25 agents tend to show that the strategy lead agents to visit very frequently few nodes as explained bellow in the last paragraph of this section. Particularly, further investigations for 25 agents showed that the bad distribution of visits over the nodes is amplified by the benefit afforded by the presence of additional agents not taken into account: many agents visit the same set of nodes as previously stated, leading the normalised criterion to be penalised.

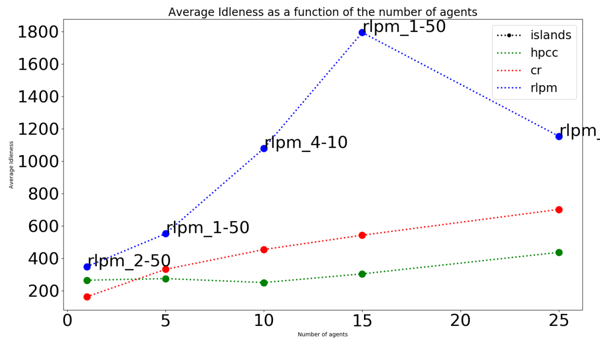

Fig. 4: Normalised average idleness of the evaluated strategy for the map Islands in ordinate for the three maps and the population sizes of agents in abscissa.

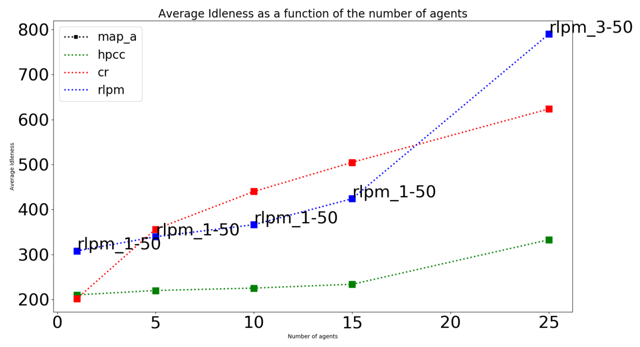

Fig. 5: Normalised average idleness of the evaluated strategy for the map A in ordinate for the three maps and the population sizes of agents in abscissa.

The **Fig. 6** shows the normalised average idleness for the topology Grid. As previously, HPCC remains the best strategy. For this map, RLPM-1-50 is always the best RLPM variant except for 1 agent where it is RLPM-2-50. The best RLPM is better than CR only for 15 and 25 agents by 43 periods in average, with values of 347 and 384 periods, respectively. For 5 agents, RLPM-1-50 and CR are approximately equal with 313 periods for the former and 302 for the latter. Lastly, as well as for the map A, the best RLPM is worse than CR for 1 agent with a value of 231 periods against 167 for CR, and for 25 agents with a value of 716 periods against 558.

Finally, RLPM shows the worst performances for the map Islands where it is always worse than CR, the reactive reference. Also, even though it is better than, or approximately equal to CR for 5, 10 and 15 agents on the topologies A and Grid, it is worse for 1 and 25 agents. Further analyses finally showed that the best RLPM is always and by far better than CR over the MI criterion, it is better or approximately identical to CR for QMI, except for Islands. However, considering the **Eq. 6** depicting $I^T_{\mathbf{G}}$ as a function of MI, QMI, and N_{Int} the

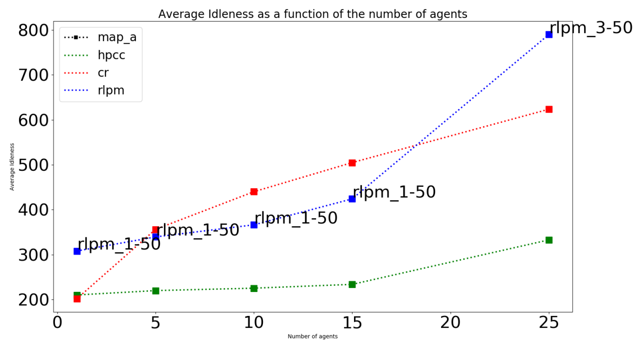

Fig. 6: Normalised average idleness of the evaluated strategy for the map Grid in ordinate for the three maps and the population sizes of agents in abscissa.

number of intervals, the QMI values and N_{Int} of the best RLPM are not low enough to offset the quadratic growing of QMI in that equation.

V. Conclusion and perspectives

We have presented a new decentralised multi-agent patrolling strategy based on LSTM networks, as well as the way these networks were trained to be used online by the patrolling agents. Our results show that in our proposed framework, networks have globally good training performances: the difference between the initial and final costs are globally significant. Considering the bad final validation costs of certain network architectures, only the four LSTM architectures $(4, 10)$, $(1, 50)$, $(2, 50)$, and $(3, 50)$ were embedded into agents and tested in simulation.

To generate multi-agent patrolling training data and assess this new strategy, we developed a new fully-fledged simulator in Python, specially designed for the multi-agent patrolling; this simulator, that we named Pytrol, allowed to gather data for learning, test and evaluate the new strategies which were confronted to the reactive and the cognitive standard strategies.

The evaluation showed that RLPM-1-50 is globally the best RLPM strategy and better than the decentralised representative CR upon the graphs A and Grid, but regardless of the RLPM version used, this new strategy is not well-adapted to the topology Islands. Indeed, the average idleness $I^T_{\mathbf{G}}$ takes into account the distribution of visits over the nodes, due to its QMI component, and the number of intervals. Preliminary investigations showed that for Islands, the RLPMs have a high QMI and many intervals, leading to the conclusion that the new strategy distributes poorly the visits of agents over the nodes.

In order to address this problem of generating a larger variety of nodes to visit, new deeper and more complex architectures will be investigated, as well as cost functions to optimise more appropriate to our problematic of node visits. Also, in further works, it will be more significant to take into account the mean interval and the quadratic mean interval criteria with the aim of analysing separately the central tendency and the distribution of intervals between visits.

References

[1] Almeida A., PM Castro, TR Menezes et GL Ramalho. Combining idleness and distance to design heuristic agents for the patrolling task. In *II Brazilian Workshop in Games and Digital Entertainment*, pages 33–40, 2003.
[2] Bengio, Y., Simard, P. & Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Networks* 5, 157–166 (1994).
[3] Bennett, M., Schatz, M.F., Rockwood, H., Wiesenfeld, K.: Huygens's clocks. *Proceedings of the Royal Society of London*. Series A: Mathematical, Physical and Engineering Sciences 458(2019) (2002) 563–579
[4] Chevaleyre, Y.: Theoretical Analysis of the Multi-agent Patrolling Problem. In: *Proc. of the Int. Conf. On Intelligent Agent Technology*, Beijing, China, pp. 302–308 (2004)
[5] Elmaliach, Y., Agmon, N., Kaminka, G.: Multi-Robot Area Patrol under Frequency Constraints. In: *Int. Conf. on Robotics and Automation*, Rome, Italy, pp. 385–390 (2007)
[6] Santana, H., Ramalho, G., Corruble, V., Ratitch, B.: Multi-Agent Patrolling with Reinforcement Learning. In: *Proc. of the Third Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, New York, vol. 3, pp. 1122–1129 (2004)
[7] Menezes, T., Tedesco, P., Ramalho, G.: Negotiator Agents for the Patrolling Task. In: Sichman, J.S., Coelho, H., Rezende, S.O. (eds.) IBERAMIA 2006 and SBIA 2006. LNCS (LNAI), vol. 4140, pp. 48–57. Springer, Heidelberg (2006)
[8] Guo, Y., Parker, L., Madhavan, R.: 9 Collaborative Robots for Infrastructure Security Applications. In: *Studies in Computational Intelligence (SCI)*, April 22, v 2007, vol. 50, pp. 185–200. Springer, Heidelberg (2007)
[9] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM networks," *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, Montreal, Que., 2005, pp. 2047-2052 vol. 4.
[10] Alex Graves, Abdel-rahman Mohamed, Geoffrey E. Hinton: Speech recognition with deep recurrent neural networks. *ICASSP 2013*: 6645-6649
[11] Hochreiter, S. & Schmidhuber, J. Long short-term memory. *Neural Comput.* **9**, 1735–1780 (1997).
[12] Yann LeCun, Yoshua Bengio, Geoffrey E. Hinton: Deep learning. *Nature* 521(7553): 436-444 (2015)
[13] Machado, A., Ramalho, G., Zucker, J., Drogoul, A.: Multi-Agent Patrolling: an Empirical Analysis of Alternative Architectures. In: Sichman, J.S., Bousquet, F., Davidsson, P. (eds.) MABS 2002. LNCS (LNAI), vol. 2581, pp. 155–170. Springer, Heidelberg (2003)
[14] Mehdi Othmani-Guibourg, Amal El Fallah-Seghrouchni, Jean-Loup Farges, Maria Potop-Butucaru. Multi-agent patrolling in dynamic environments. *2017 IEEE International Conference on Agents (ICA)*. 2017.
[15] D. O. Sales, D. Feitosa, F. S. Osorio and D. F. Wolf, "Multi-agent Autonomous Patrolling System Using ANN and FSM Control," 2012 *Second Brazilian Conference on Critical Embedded Systems*, Campinas, 2012, pp. 48-53.
[16] P.A. Sampaio, G. Ramalho et P. Tedesco. The Gravitational Strategy for the Timed Patrolling. In *22nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI) 2010*, volume 1, pages 113–120. IEEE, 2010.