ALGERIAN DEMOCRATIC AND POPULAR REPUBLIC
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
ABOU-BAKR BELKAID UNIVERSITY - TLEMCEN

THESIS

Presented at:

FACULTY OF SCIENCE - DEPARTMENT OF COMPUTER SCIENCE

For the graduation of:

DOCTORATE

Specialty: Computer science

By :

Zoulikha KOUDAD

On the subject of

Methods for Automatic Option Discovery in Hierarchical Reinforcement Learning

Defended Publicly on July 15th 2024 at Tlemcen University, in front of the jury composed of:

Mr HADJILA Fethallah	MCA	Université de Tlemcen	Chairman
Mr MERZOUG Mohamed	MCA	Université de Tlemcen	Supervisor
Mr HADJILA Mourad	MCA	Université de Tlemcen	Reviewer
Mr MALIKI Fouad	MCA	Ecole ESSAT de Tlemcen	Reviewer
Mr BRAHAMI Mustapha	MCA	Ecole ESSAT de Tlemcen	Reviewer

Laboratory of Research in Computer Science of Tlemcen LRIT.
BP 119, 13000 Tlemcen - Algeria.

Abstract

The hierarchical reinforcement learning framework breaks down the reinforcement learning problem into subtasks or extended actions called options in order to facilitate its resolution. Different models have been proposed where options were manually predefined or semi-automatically discovered. However, the automatic discovery of options has become a real challenge for research in hierarchical reinforcement learning.

In this thesis we propose two automatic option discovery method for hierarchical reinforcement learning. The first method that we call Fast Automatic Option Discovery (FAOD). In this contribution, we took inspiration from robot learning methods to categorize the sensorimotor flow during navigation. Thus, FAOD agent moves along the walls to discover the rooms' contour, closed spaces, doors and bottleneck regions to define terminate states and initiation sets for options.

In the second contribution our learning agent uses his sense of direction to discover the shortest paths and shortcuts after an exploration based on intrinsic motivation, without resorting to the algorithms of the graph theory, these discoveries subsequently serve to discover the termination conditions and the initiation states of the options. For the learning of options policies, the agent uses his experience of exploration as well as learning by temporal difference strategy. We tested and validated this approach on different maze problems and on the tic-tac-toe game.

Keywords

Hierarchical reinforcement learning; Reinforcement learning; Option Discovery; Markov decision process; Actor-criticlearning, Wayfinding; Intrinsic motivation.

Acknowledgements

I would like to thank my supervisor Dr MERZOUG Mohamed for his role as advisor during my doctoral career, and for his constructive proposals and his availability, that allowed me to progress steadily in my research. I would like to thank my first (ex) supervisor, Dr BENAMAR Abdelkrim for his commitment at the beginning of my doctoral training.

I would also like to thank the President Pr HADJILA Fethallah and the other jury members Dr HADJILA Mourad, Dr MALIKI Fouad and Dr BRAHAMI Mustapha for kindly evaluating this work.

I am infinitely grateful to Pr Mehdi KHAMASSI for introducing me to the field of reinforcement learning, and for the constructive and insightful discussions.

I would also like to thank the computer science team at the Higher School of Applied Sciences in Tlemcen for their encouragement and support as well as the administrative team for making my teaching task easier during my last year of Ph.D preparation.

I gratefully acknowledge the love and support of all my family, specially my parents, my sisters for their unconditional support and my brothers. Last but not least, for being very supportive, I would to thank my husband and I dedicate this work to my children.

Contents

List of Figures

List of algorithms

List of Tables

Acronyms

ANN Artificial Neural Network. 32

BHNN Budgeted Hierarchical Neural Network. 46

BIC Bayesian Information Criterion. 56

CPT Conditional Probability Tree. 56

DBN Dynamic Bayesian Network. 56

DG Directed Graph. 49

ER Experience Replay. 54

FAOD Fast Automatic Option Discovery. 2

FiGAR Fine Grained Action Repetition. 48

FMDP Factored MDP. 56

GPI Generalized Policy Iteration. 23

GRU Gated Recurrent Unit. 47

HRL Hierarchical Reinforcement Learning. 15

MC Monte Carlo. 25

MDP Markov Decision Process. 15

PAC Probably Approximately Correct. 50

PVF Proto-Value Function. 51

RGI Relevant Goal Information. 55

RL Reinforcement Learning. 15

SCC Strongly Connected Component. 49

SMDP Semi-Markov Decision Process. 37

SVI Structured Value Iteration. 57

TD Temporal-Difference. 27

Introduction

A large part of artificial intelligence techniques such as neural networks, fuzzy systems, genetic algorithms, and others are inspired by human or animal behavior or mechanism. Reinforcement Learning (RL) is similar to animals' ability to learn to predict reward through trial-and-error learning Sutton and Barto [1981]. RL consists of an agent (software or hardware) that interacts with his environment, and is at every moment t in a state s and must choose an action a to perform. As a result of this action, the agent will end up in a new state s' and receive a scalar reward r. The learning algorithm consists of testing and learning from mistakes in order to progressively maximize the amount of reward obtained from the environment, Sutton and Barto [1998]. To do so, the agent has to find an optimal policy, which is defined as the policy which maximizes the cumulative total reward calculated by a value function over a finite or infinite horizon, depending on the task. However, RL suffers from the curse of dimensionality, when the environment becomes complex or the number of states and actions is very large. This leads to a combinatorial explosion which makes learning slow or inefficient. Another type of problem relates to its inability to coordinate different scales of representation. For example, when an agent has to navigate between different positions inside a room, and then between different places in a city, the human designer has to manually predefine this change in scale because it cannot be discovered by the learning algorithm. One solution consists in dividing the problem to solve into several levels, and several subproblems at each level, which makes the algorithm hierarchical. For this reason, this method is called Hierarchical Reinforcement Learning (HRL) ,McGovern et al. [1998], Precup and Sutton [1998].

The Hierarchical Reinforcement Learning is a recent field of research in artificial intelligence that has emerged as a solution to the dimensionality problem of conventional reinforcement learning systems. The solution carried by the HRL consisted of dividing a problem, modeled generally in the form of Markov Decision Process (MDP), in two or more hierarchical levels, each lower level is divided into several sub problems to be solved, subsequently called options, the solutions or policies made for the options are

subsequently used as atomic actions to solve the problem of the higher level. With the emergence of the HRL, the decomposition of problems, and the choice of options was done manually, which quickly became a problem, hence the emergence of automating this process as a new field of research. New works has been done in this new field, consisting of one side automatically discovering the hierarchical levels of a problem, and / or discovering the options of a given level on the other side, and it is the latter under fields that interests us in this thesis.

An option is characterized by a termination condition that corresponds to the sub-goal, a set of initiation states, and a policy that provides the solution to the sub-problem. The options offer to the agent the opportunity to use them as many times as he needs, in different tasks. However, discovering the relevant options remains a real challenge for the research community, several approaches are proposed, starting with a manual definition of the options, Botvinick et al. [2009], later semi-automatic and automatic methods appeared.

Objectives

The aim of the work reported in this thesis was initially to study existing approaches in the field of hierarchical reinforcement learning by focusing on automatic option discovery approaches.

Artificial intelligence researchers have introduced several methods and algorithms to divide a problem, modeled as a Markov decision process, into sub-problems to be solved, starting with the manual definition of options and moving towards automatic definitions sometimes requiring parameters which must be defined manually, other methods are very complex or very time consuming.

Hence our objective is to develop a method that is completely automatic and less slow in execution. On the other hand, the options discovered must be reusable for other problems of the same type as the initial problem, with an effective strategy for discovering initiation states and termination conditions or sub-goals and for learning option policies in a short time.

Scientific contributions

This dissertation present the following contributions;

- Classification of methods of automatic option discovery

- Review of several approaches in the field of automatic option discovery for HRL and their comparison.
- We introduce a fast approach for automatic option discovery applied on maze problems.
- We introduce a second approach for automatic option discovery based on an agent endowed with a sens of direction and an intelligent exploration based on intrinsic motivation of the agent.

Structure of the thesis

The chapters are organized as follows

- Chapter 2 is reserved for the description of reinforcement learning and hierarchical reinforcement learning as well as the Markov and semi-Markov decision-making process.
- Chapter 3 presents the state of the art of automatic option discovery methods, with method classification, comparison, and critical review.
- In Chapter 4, we describe our first method called FAOD for "Fast Automatic Option Discovery in Hierarchical Reinforcement Learning" applied to spatial and maze problems. This method discovers the closed spaces and bottlenecks that will become the future initiation and termination states.
- In Chapter 5, we present our second method based on agent with a sense of direction. This method will be tested on maze type problems and on the Tic-Tac-Toe game.
- In the conclusion and future work we conclude our work and discuss prospects for future research.

1 Background on reinforcement learning and hierarchical reinforcement learning

Reinforcement Learning is a branch of artificial intelligence where an agent learns to make optimal decisions by interacting with its environment. The theoretical foundations of RL are based on Markov Decision Processes and dynamic programming methods, such as policy iteration and value iteration. This chapter provides a detailed introduction to these concepts, along with an exploration of RL algorithms, including Monte Carlo methods and temporal difference methods.

Monte Carlo methods use samples to estimate policy values, while temporal difference methods combine ideas from Monte Carlo and dynamic programming for more efficient learning. Hierarchical Reinforcement Learning is then introduced, aiming to overcome the limitations of traditional RL methods by structuring problems into hierarchical subtasks. We discuss options and semi-MDPs, key concepts that allow complex problems to be decomposed into more manageable sub-problems. Finally, this chapter addresses intrinsic motivation and the sense of direction, concepts independent of but often related to the learning context. Intrinsic motivation drives the agent to explore and learn without the need for explicit extrinsic rewards, while the sense of direction helps the agent navigate a partially observed environment.

1.1 Markov Decision Processes

Markov Decision Processes (MDPs) allow problem modeling very suitable for reinforcement learning. These are issues related to the decision-making process by an agent in an environment, where the results of the agent's actions are partially stochastic. Markov Decision Processes (MDPs) as presented in figure 1.1, named after Andrei Andreyevich Markov (1856 - 1922), is a mathematical model for the random evolution of a memory-

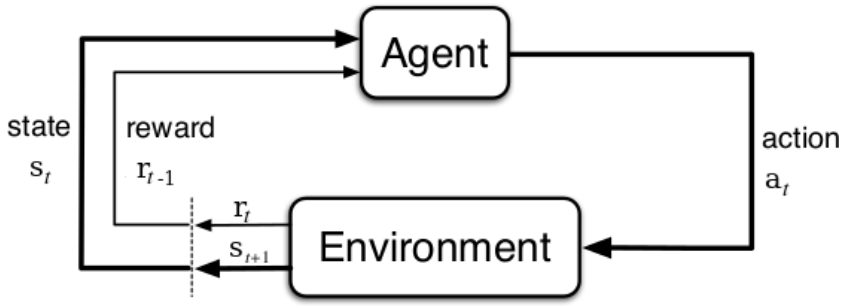


Figure 1.1: Interaction between agent and environment in a Markov decision process. Sutton and Barto [2018]

less system. Such a system respects the Markov property, that is a given future state depends only on the present state and where knowing more about the past does not bring any further information.

An MDP, as a discrete time stochastic control process, provides a mathematical framework for modeling decision making in situations where the outcome is partially controlled by the agent, Kozlova [2010].

1.1.1 formal definition

Formally, an MDP is a tuple $(S, A, P, r,)$ where;

S is a finite set of states.

A is a finite set of actions.

$P(s' \mid s, a)$ is the function of probability of transition to the state s' after taking the action a at the state s.

$r(s, a)$ is an immediate reward received after selecting the action a at the state s.

The set of terminal or absorbing states are states from which the agent can no longer exit once entered and his task is completed, so he can start over from the beginning; we then speak of episodic tasks or with a finite horizon.

The agent's goal is not to maximize the immediate reward but to maximize the expected cumulative reward of future actions, subsequently called return R; which at a time t can simply be defined by

$$R_t = r_t + r_{t+1} + r_{t+2} + ... + r_T, \tag{1.1}$$

where T is the final number of steps.

However in most cases T is not defined, and the number of steps may not be limited. For this reason, the concept of discounting is added as follows:

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... = \sum_{k=0} \gamma^k r_{t+k}, \tag{1.2}$$

where $\gamma \in [0, 1]$ is the discount factor, adjusts the emphasis on future returns relative to immediate returns, Sutton and Barto [1998], Degris [2007], Kozlova [2010].

1.1.2 Policy and value function

The value function can be a state function or a function of pair (state-action), its role is to estimate how much it is beneficial for the agent to be in a given state, or how much it is beneficial for the agent to perform a given action while in a given state, this by estimating the expected future rewards when the agent is in a given state or when he chooses to perform an action in a given state.

A policy or strategy can be defined as the application which, in the deterministic case, associates with each state s the action that the agent must take, and is noted $\pi(s)$. In the stochastic case, a policy associates with each state s the probability of choosing each possible action a, and will be noted $\pi(s, a)$.

Thereafter, the value function $V_\pi(s)$ of a state s is defined as the cumulative rewards expected in state s by following the policy π, and is defined as follows :

$$V_\pi(s) = \mathbb{E}[R_t \mid s_t = s] \tag{1.3}$$

Likewise, we define the value function of a state-action pair for a policy π as follows :

$$Q_\pi(s, a) = \mathbb{E}[R_t \mid s_t = s, a_t = a] \tag{1.4}$$

1.1.3 Bellman equation

The value function can be written as Bellman equation which demonstrates the relationship of the value of a state with the values of successor states.

$$\begin{aligned} V_\pi(s) &= \mathbb{E}[R_t \mid s_t = s] \\ &= \mathbb{E}\left[\sum_{k=0} \gamma^k r_{t+k} \mid s_t = s\right] \\ &= \mathbb{E}\left[r_t + \sum_{k=1} \gamma^k r_{t+k} \mid s_t = s\right] \\ &= \mathbb{E}\left[r_t + \gamma \sum_{k=1} \gamma^{k-1} r_{t+k} \mid s_t = s\right] \\ &= \mathbb{E}[r_t] + \gamma \mathbb{E}\left[\sum_{k=1} \gamma^{k-1} r_{t+k} \mid s_t = s\right] \\ &= \mathbb{E}[r_t] + \gamma \mathbb{E}[R_{t+1} \mid s_t = s] \\ &= \mathbb{E}[r_t] + \gamma [V_\pi(s_{t+1})], \end{aligned} \tag{1.5}$$

where $\mathbb{E}[r_t]$ is the average of rewards for all possible actions by the policy π in the current state s, where :

$$\mathbb{E}[r_t] = \sum_{a \in A(s)} \pi(s,a) \sum_{s' \in S} P(s' \mid s,a)\, r(s,a) \tag{1.6}$$

Therefore, given a policy π, the value of the state s is defined by :

$$V_\pi(s) = \sum_{a \in A(s)} \pi(s,a) \sum_{s' \in S} P(s'|s,a) [r(s,a) + \gamma V_\pi(s')] \tag{1.7}$$

This equation is called the Bellman equation.

1.1.4 Optimal Policies

The optimal policy is the policy with the greatest expected return. There can be several optimal policies π^* With the same optimal value function V^* :

$$V^*(s) = \max_\pi V_\pi(s) \tag{1.8}$$

Likewise for the state-action value function :

$$Q^*(s,a) = \max_{a \in A(s)} Q_\pi(s,a) \tag{1.9}$$

Hence the Bellman optimality equation for V^* is written as follows :

$$V^*(s) = \max_{a \in A(s)} \sum_{s' \in S} P(s' \mid s, a) \left[r(s, a) + \gamma V^*(s') \right], \tag{1.10}$$

and for Q^* the Bellman optimality equation is :

$$Q^*(s, a) = \sum_{s' \in S} P(s' \mid s, a) \left[r(s, a) + \gamma \max_{a'} Q^*(s', a') \right] \tag{1.11}$$

To find and build the optimal policy, the main idea is to use the value function and improve it iteratively. In the following sections we will introduce the two most efficient methods to solve this problem; dynamic programming and reinforcement learning.

1.2 Dynamic programming

Dynamic programming refers to a set of algorithms for calculating the optimal policies π^* of a finite MDP, Degris [2007] provided that the dynamics of the environment are known a priori. The dynamics of the environment are defined by the transition function P, and the reward function r. These algorithms are called model-based algorithms, and indirect learning algorithms, and are considered offline learning algorithms due to the fact that the agent does not need to interact with the environment to know the results of his actions.

In the following subsections we present two dynamic programming algorithms, the policy iteration algorithm and the value iteration algorithm.

1.2.1 Policy Iteration

The policy iteration algorithm Sutton and Barto [1998], Kozlova [2010], Geist [2009] begins with a random initialization of the policy and value function for all states of the MDP, and then proceeds by iterations. At each iteration the algorithm evaluates the policy by calculating the value function V using the Bellman equation, then performs an improvement of the current policy.

The evaluation of the policy consists in calculating V_π for each state $s \in S$ and repeating this process until values stabilisation, because the value of one state s depends also on values of the successor states in the trajectories defined by the policy π.

Policy improvement is done for every state s in a greedy fashion by choosing the action

for it that maximizes its value $V(s)$ given by the Bellman equation.

The algorithm iterates evaluation and improvement until the policy stops improving. In this case, we have reached an optimal policy.

Algorithm 1 Policy Iteration.

Require: θ
Ensure: V^*,π^*.
Initialize $V_\pi(s)$ and $\pi(s)$ arbitrarily.
{Policy Evaluation}
repeat
  $\Delta \leftarrow 0$
  for all $s \in S$ **do**
    $v \leftarrow V_\pi(s)$
    $V_\pi(s) \leftarrow \sum_{s' \in S} P(s'|s, \pi(s)) [r(s, \pi(s)) + \gamma V_\pi(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V_\pi(s)|)$
  end for
until $\Delta < \theta$
{Policy improvement}
$stable \leftarrow True$
for each $s \in S$ **do**
  $b \leftarrow \pi(s)$
  $\pi(s) \leftarrow argmax_{a \in A(s)} \sum_{s' \in S} P(s' \mid s, a) [r(s, a) + \gamma V_\pi(s')]$
  if $b \neq \pi(s)$ **then**
    $stable \leftarrow False$
  end if
end for
if not $(stable)$ **then**
  GOTO 2
end if
return V,π.

The policy iteration algorithm introduced a concept that will be used by all reinforcement learning algorithms, namely the idea of alternating between two processes, the policy evaluation process and the process of policy improvement. Regardless of the granularity and other details of the two processes, the two processes converge to give the optimal value function and the optimal policy. This concept is called Generalized Policy Iteration (GPI).

1.2.2 Value Iteration

The Policy Iteration Algorithm is an iterative algorithm that contains in each iteration another computationally expensive iterative algorithm which is the Policy Evaluation Algorithm. The latter continues its iterations over all MDP states until convergence to move to the improvement phase. However, the number of iterations of the evaluation phase can be truncated so that only one remains, which will be combined with the policy improvement step into a single operation. This operation will repeat until convergence, which constitutes a new algorithm called value iteration.

The resulting improvement evaluation operation is always based on the Bellman equation and the optimal policy is deduced after convergence of iterations.

Algorithm 2 Value Iteration.

Require: θ
Ensure: V^*,π^*.
  Initialize V arbitrarily.
  repeat
    $\Delta \leftarrow 0$
    for all $s \in S$ **do**
      $v \leftarrow V(s)$
      $V(s) \leftarrow \max_a \left[r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V(s') \right]$
      $\Delta \leftarrow \max(\Delta, |v - V_\pi(s)|)$
    end for
  until $\Delta < \theta$
  for each $s \in S$ **do**
    $\pi(s) \leftarrow argmax_{a \in A(s)} \sum_{s' \in S} P(s' \mid s,a) [r(s,a) + \gamma V(s')]$
  end for
  return V,π.

1.3 Reinforcement learning

Classical DP algorithms are of limited utility in reinforcement learning both because of their assumption of a perfect model and because of their high computational cost, Sutton and Barto [1998]. These algorithms are considered as model-base reinforcement learning, and are based on the assumption that the transition and reward functions of the MDP to be solved are known. However, this assumption does not suit many problems. On the other hand, model free RL uses experience to directly learn one or both of the two quantities state/action values or policies that can achieve the same optimal behavior

without estimating or using the environment model, Dayan and Niv [2008].

Temporal Difference Learning (TD-LEARNING) like Q-LEARNING or SARSA and ACTOR-CRITIC are examples of modelless algorithms, Sutton and Barto [1998], Kozlova [2010]. Indeed, these reinforcement learning algorithms make online or direct learning.

Reinforcement learning (RL) is similar to animals' ability to learn to predict reward through trial-and-error learning, Sutton and Barto [1981]. It traces back work on animal learning by the psychologist Edward Thorndike in 1911, Thorndike [1911]. It later developed within the field of computer science in the 1950s with the advent of artificial intelligence.

To our knowledge, the first theoretical work on trial and error learning was from Turing's, Minsky and Fardy's, and Clark's publications. In parallel, still in the 1950s, the term "optimal control" was introduced for dealing with dynamic systems. The solution was given by dynamic programming algorithms based on the work of Richard Bellman, Hamilton and Jacobi, and Ron Howard. Subsequently in 1961, Michie in Michie [1961] described a system capable of learning to play the tic-tac-toe game, and Samuel in Samuel [1959] realized a software that teaches the game of checkers.

The merger between trial and error learning and dynamic programming was done by K Lopf H [1975] between 1972-1975, and it is in 1981 that Sutton and Barto, Sutton and Barto [1981] "considered as pioneers of modern RL" implemented a linear perceptron whose learning equation is derived and extended from the work of Rescorla and Wagner in 1972 in experimental psychology, Rescorla et al. [1972]. This equation introduces the notion of temporal difference: the ability to learn to predict reward based on the comparison between two temporally consecutive estimations of this prediction. Sutton, Barto and Anderson proposed in 1983 the actor-critic architecture, Barto et al. [1983], which is based on temporal difference learning.

1.3.1 Monte Carlo Methods

Monte Carlo (MC) methods do not assume complete knowledge of the environment and require only experience, they sample sequences of states, actions, and rewards from actual or simulated interaction with an environment.

Monte Carlo methods are used for episodic tasks, thus, experience is divided into episodes, and all episodes eventually terminate no matter what actions are selected. It is only at the end of an episode that the value estimates and policies are changed. Hence Monte Carlo methods are incremental in an episode-by-episode sense, but not in

a step-by-step sense, Sutton and Barto [1998].

In reinforcement learning, Monte Carlo methods are a family of methods based on the same principle of trajectory plotting to make prediction, estimate of state value function or state – action value function, and control. In this section we will detail two algorithms, the state value function estimation algorithm, and the control algorithm for the estimation of the optimal policy.

Monte Carlo Prediction

Here we present one Monte Carlo algorithm for learning the state-value function for a given policy, that is the value expected of the cumulative future discounted reward, starting from the actual state. An obvious way to estimate it from experience, that is simply to average the returns observed after visits to that state. As more returns are observed, the average should converge to the expected value.
For the estimation of $V_\pi(s)$, the value of a state s under policy π, we have a set of episodes obtained by following π and passing through s. Each occurrence of state s in an episode is called a visit to s. Of course, s may be visited multiple times in the same episode; the first time it is visited in an episode is called the first visit to s. The first-visit MC method estimates $V_\pi(s)$ as the average of the returns following first visits to s, whereas the every-visit MC method averages the returns following all visits to s. These two methods are very similar but have slightly different theoretical properties. First-visit MC is shown in procedural form in the algorithme 3. First-visit MC algorithm converge to $V_\pi(s)$ as the number of first visits to s goes to infinity, Sutton and Barto [1998].

Monte Carlo Control

In model based algorithms like policy iteration and value iteration, it is useful to estimate only value function to determine policy, one simply looks ahead one step and chooses whichever action leads to the best combination of reward and next state. But here in reinforcement learning there is no model for determining the action leading to a state and a reward, and we cannot determine the policy, so one of the main purposes of the methods of Monte Carlo is to estimate the policy, thus one of primary goals for Monte Carlo methods is to estimate state-action pairs values $Q*$ rather than state values.

The estimation of state-action value $Q_\pi(s,a)$ is to estimate the expected return when starting in state s , taking action a, and thereafter following policy π. Thus , the Monte Carlo methods are then adapted to state-action visit rather than state visit. A state-action pair (s,a) is said to be visited in an episode if the state s is visited and action a

Algorithm 3 First-visit Monte Carlo Prediction.

Require: a policy π
Ensure: V_π.
  Initialize V arbitrarily for all $s \in S$.
  $Returns(s) \leftarrow$ an empty list for all $s \in S$.
  loop
    Generate an episode following $\pi : s_0, a_0, r_1, s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T$
    $G \leftarrow 0$
    for each step of episode,$t = T-1, T-2, ..., 0$: **do**
      $G \leftarrow \gamma G + r_{t+1}$
      Unless s_t appears in $s_0, s_1, ..., s_{t-1}$
      Append G to $Returns(s_t)$
    end for
    $V(s_t) \leftarrow average(Returns(s_t))$
  end loop

is taken in it. The first-visit MC method averages the returns following the first time in each episode that the state was visited and the action was selected, the same adaptation was done for the definition of the every visit MC method.

The goal of MC control algorithms is to approximate optimal policies, as the majority of RL methods, the MC control algorithm alternates between policy evaluation and policy improvement, that is the principle of generalized policy iteration. The first-visit MC control algorithm 4 start with an arbitrary policy π_0. The policy evaluation is done by estimating state-action value function, an infinit number of episodes are generated to observe first-visits of state-action pairs, for exploration purposes, a new start state-action is generated at each episode.

The policy improvement step is greedy as in policy iteration algorithm, thus for any action value function Q, the corresponding greedy policy is the one that for each $s \in S$ deterministically chooses an action with maximal action-value :

$$\pi(s) = \arg\max_a Q(s, a), \tag{1.12}$$

π_{k+1} is therefor constructed greedily with respect to Q_{π_k}.
In Sutton and Barto [1998] it is proved that π_{k+1} is uniformly better than π_k, or just as good as π_k , in which case they are both optimal policies.

Algorithm 4 First-visit Monte Carlo Control.

Ensure: $\pi*$, $Q*$.
  Initialize:
  $\pi(s) \in A(s)$ arbitrarily for all $s \in S$.
  $Q(s,a) \in \mathbb{R}$, for all $s \in S$,$a \in A(s)$.
  $Returns(s,a) \leftarrow$ an empty list for all $s \in S$,$a \in A(s)$.
  loop
    Choose $s_0 \in S$, $a_0 \in A(s_0)$ randomly with probability > 0.
    Generate an episode form s_0,a_0 following $\pi : s_0, a_0, r_1, s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T$
    $G \leftarrow 0$
    for each step of episode,$t = T-1, T-2, ..., 0$: **do**
      $G \leftarrow \gamma G + r_{t+1}$
      Unless the pair s_t,a_t appears in $s_0, a_0, s_1, a_1 ..., s_{t-1}, a_{t-1}$
      Append G to $Returns(s_t, a_t)$
    end for
    $Q(s_t, a_t) \leftarrow average(Returns(s_t, a_t))$
    $\pi(s_t) = \arg\max_a Q(s_t, a)$
  end loop

1.3.2 Temporal-Difference Learning

"If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be Temporal-Difference (TD) learning", Sutton and Barto [1998]. TD learning methods combine the advantages of ideas from Monte Carlo methods and ideas from dynamic programming. TD learning is similar to Monte Carlo methods in that it does not need a model and learns directly from experience. At the same time, the method estimates the value function based on other estimations (they bootstrap[1]) without waiting for the final return of the episode, and this is what the methods of dynamic programming do.

TD method is based on the principle of generalized policy iteration (GPI), the difference between this method and the previous methods is mainly found in the phase of policy evaluation, or the prediction problem.

TD Prediction

TD learning is model-free learning and is based on experience or interaction with the environment. It is an episodic method like the MC technique. While the latter waits for the end of the episode to be able to update the $V(s)$ values, the TD methods wait only

[1] bootstrapping is to make an estimate based on other estimates

for a single step.
At step t, TD method waits for the next step $t+1$ to retrieve the reward r_t and the estimated value $V(s_{t+1})$ to update s_t as follows:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_t + \gamma V(s_{t+1}) - V(s_t) \right], \tag{1.13}$$

where $\alpha \in [0,1]$ is the learning rate.
Indeed the target for updating the TD learning is the term $r_t + \gamma V(s_{t+1})$, this method is called TD(0), or one step TD which is a special case of TD(γ) or the n-step (for more details see chap 7 and chap12 of the book, Sutton and Barto [1998]). The TD(0) method is presented in the algorithm 5 below.

Algorithm 5 TD(0) for estimating V_π.

Require: the policy π to be evaluated
Ensure: V_π.
parameter: step size $\alpha \in [0,1]$.
Initialize: $V(s)$ arbitrarily for all $s \in S$.
for each episode **do**
Initialise s. {the initial state}
repeat
$a \leftarrow \pi(s)$ action given by π for actual state s
Take action a; observe reward r and next state s'.
$V(s) \leftarrow V(s) + \alpha \left[r + \gamma V(s') - V(s) \right]$
$s \leftarrow s'$
until s is terminal.
end for

TD like MC methods uses sample updates, i.e. they use the successor value (or values) along the way to calculate the value of a state s which has just been visited, unlike DP methods which do not wait interact with the environment to see the successors, since they maintain a full proability distribution of all possible successors.

TD methods are more advantageous compared to dynamic programming methods and Monte Carlo methods. for the first case it is due to the fact that the TD methods do not need an environment model to predict the dynamics of the MDP, and compared to the MC methods which at each iteration of learning must wait for the end of the episode to compute the value function, the TD methods only wait for one step of time. And with regard to the convergence of the algorithm, it was proved in [Sutton 98] that for any fixed policy π the algorithm TD converges to V_π.

TD Control: policy improvement

The temporal difference methods also follow the Generalized Policy Iteration model, after having seen a TD prediction method which represents the policy evaluation phase we will see two TD control methods i.e. the policy improvement phase.

Policy learning is faced with the exploration / exploitation dilemma. Exploration consists in choosing the actions already known and which give the best return, while the exploration chooses new actions in the hope of receiving even greater cumulative rewards, but at the risk of sometimes adopting an under-optimal behavior.

Methods of learning policies must find a trade off between exploration and exploitation. In this perspective there are two classes; on-policy and off-policy methods, Sutton and Barto [1998], Léon [2019]. In on-policy methods, the learned policy is used at the same time as it is improved. The disadvantage of these methods is that the same policy must make it possible to learn the optimal actions to be carried out while exploring the environment, therefore by using a behavior that is not necessarily optimal.

The off policy methods allow two policies to be used at the same time, an explorer policy which generates random actions, this policy is called bihavioral policy. The second policy is the target policy which is evaluated and improved. Off policy methods are often of higher variance and converge more slowly, but at the same time they are the most powerful.

ϵ-greedy method

Among the exploration methods, we cite the ϵ-greedy method, Degris [2007], where at each time step a random action is chosen with a low probability ϵ where $(0 < \epsilon < 0.2)$. The choice of this action follows a uniform distribution independent of the actions values. And with greater probability $(1 - \epsilon)$ the action with the best action value is chosen. The greedy method makes it possible to obtain a good trade off between exploration and exploitation.

Sarsa

Sarsa is an on-policy TD control method. It is a free model method, that is to say that we do not know in advance the transition function and the reward function, the reason why we must learn the value function (state-action) instead of the state value function. This is possible by adapting the TD estimation method as follows :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \tag{1.14}$$

This update is performed after each transition from a non-terminal state s_t, if s_{t+1} is terminal then $Q(s_{t+1}, a_{t+1})$ is set to zero. At each transition, all the elements of the quintuplet $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ are used, hence the name of the Sarsa method.
The Sarsa algorithm estimates the action value Q_π for the applied policy π and at the same time changes π according to Q_π as presented in the algorithm 6.

Algorithm 6 Sarsa (on-policy TD control) for estimating Q.

parameter: step size $\alpha \in [0, 1]$, small $\epsilon > 0$.
Initialize: $Q(s, a)$ arbitrarily for all $s \in S$, $a \in A(s)$.
for each episode **do**
  Initialise s. {the initial state}
  Choose a from s using policy derived from Q (e.g., ϵ-greedy)
  repeat
    Take action a; observe reward r and next state s'.
    Choose a' from s' using policy derived from Q (e.g., ϵ-greedy)
    $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma Q(s', a') - Q(s, a) \right]$
    $s \leftarrow s'$;$a \leftarrow a'$;
  until s is terminal.
end for

Q-learning

This is a off-policy TD control algorithm, it is one of the oldest RL algorithms, see Watkins [1989], Watkins and Dayan [1992], its formula for updating the action value function is as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right] \tag{1.15}$$

The action value function Q directly approximates the optimal action value function Q* independently of the policy followed during learning.

The difference between the Sarsa algorithm and Q-learning lies in the fact that Sarsa performs updates according to the actions actually chosen, while the Q-learning algorithm modifies the values of the function according to the optimal actions even if the agent had not chosen them, hence the expression $\max_a Q(s_{t+1}, a)$ in the formula of the value function. The use of two policies at the same time, one of which is completely arbitrary, allows a good compromise between exploitation and exploration. The Q-learning

algorithm performs an update after each (s_t, a_t, s_{t+1}, r_t) transition as indicated by the algorithm 7.

Algorithm 7 Q-learning (off-policy TD control) for estimating π.

parameter: step size $\alpha \in [0, 1]$, small $\epsilon > 0$.
Initialize: $Q(s, a)$ arbitrarily for all $s \in S$, $a \in A(s)$.
for each episode **do**
  Initialise s. {the initial state}
  repeat
    Choose a from s using policy derived from Q (e.g., ϵ-greedy)
    Take action a; observe reward r and next state s'.
    $Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_a Q(s', a) - Q(s, a)]$
    $s \leftarrow s'$;
  until s is terminal.
end for

1.3.3 Policy Gradient Methods

All methods already learned was action-value methods; they learned the values of actions and then selected actions based on their estimated action values. Now, we consider methods that learn parameterized policy that can select actions without consulting a value function, Sutton and Barto [2018]. Let be θ the policy's parameter vector, thus $\pi(a \mid s, \theta) = \Pr\{a_t = a \mid s_t = s, \theta_t = \theta\}$ define the probability that action a is taken at time t in the state s with parameter θ. Methods for learning policy are based on the gradient ascent to maximise a performance measure $J(\theta)$ as follow :

$$\theta_{t+1} = \theta_t + \alpha \widehat{\nabla J(\theta)}, \tag{1.16}$$

where $\widehat{\nabla J(\theta)}$ is a stochastic estimate whose expectation approximates the gradient of the performance measure.
Methods that follow this schema are called policy gradient methods, among them methods that learn approximations to both policy and value function are called actor-critic methods, where 'actor' is a reference to the learned policy, and 'critic' refers to the learned value function. The advantage of policy gradient methods is that they can be applied for episodic case as well as the continuing case. In this thesis we will treat only the episodic (discret) case.
If the action space is discrete and not too large, then it is useful to form parameterized numerical preferences $h(s, a, \theta) \in \mathbb{R}$ for each state-action pair. The actions with the

highest preferences are given the highest probabilities of being selected, for example, according to an exponential soft-max distribution:

$$\pi(a, s, \theta) \doteq \frac{e^{h(s,a,\theta)}}{\sum_b e^{h(s,b,\theta)}} \tag{1.17}$$

The action preferences can be parameterized arbitrarily, by a deep Artificial Neural Network (ANN), where θ is the vector of all the connection weights of the network.
For the episodic case, the performance measure can be defined to be the value of the start state of the episode. By assuming that every episode starts in some particular state s_0, the performance will be :

$$J(\theta) = V_{\pi_\theta}(s_0), \tag{1.18}$$

where V_{π_θ} is the true value function for π_θ, the policy determined by θ.
Performance depends on action selections and the distribution of states in which those selections are made, and both of these are affected by the policy parameter which effect on the actions, and thus on reward, can be computed, but the effect of the policy on the state distribution is a function of the environment and is typically unknown. Thus we can't estimate the performance gradient with respect to the policy parameter when the gradient depends on the unknown effect of policy changes on the state distribution. The **policy gradient theorem**, Sutton and Barto [2018] gives the solution to this problem by providing an analytic expression for the gradient of performance with respect to the policy parameter. The policy gradient theorem for the episodic case establishes that

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a q_\pi(s, a) \nabla \pi(a \mid s, \theta), \tag{1.19}$$

where the gradients are column vectors of partial derivatives with respect to the components of θ, and π denotes the policy corresponding to parameter vector θ. The symbol $\propto$ means "proportional to". The policy gradient theorem gives an exact expression proportional to the gradient without involving the derivative of the state distribution; all that is needed is some way of sampling whose expectation equals or approximates this expression.

REINFORCE: Monte Carlo Policy Gradient

Monte Carlo policy gradient algorithm called REINFORCE, Sutton and Barto [2018] derive from the policy gradient theorem as follow:

$$\begin{aligned}\nabla J(\theta) &\propto \sum_{s} \mu(s) \sum_{a} q_{\pi}(s,a) \nabla \pi(a \mid s, \theta) \\ &= \mathbb{E}_{\pi}\left[\sum_{a} q_{\pi}(s_t,a) \nabla \pi(a \mid s_t, \theta)\right]\end{aligned}$$

At time t, the classical REINFORCE algorithm involve only a_t the action actually taken, thus the sum over the random variable's possible values is replaced by an expectation under π, and then the expectation is sampled, it is in this way that we introduced s_t in the last equation and that we are going to introduce a_t in the following equation. However, for an expectation under π, the terms must be weighted by $\pi(a_t \mid s_t, \theta)$, so the right hand side of the equation is then multiplied and divided by $\pi(a_t \mid s_t, \theta)$.

$$\begin{aligned}\nabla J(\theta) &= \mathbb{E}_{\pi}\left[\sum_{a} \pi(a \mid s_t, \theta) q_{\pi}(s_t,a) \frac{\nabla \pi(a \mid s_t, \theta)}{\pi(a \mid s_t, \theta)}\right] \\ &= \mathbb{E}_{\pi}\left[q(s_t,a_t) \frac{\nabla \pi(a_t \mid s_t, \theta)}{\pi(a_t \mid s_t, \theta)}\right] \\ &= \mathbb{E}_{\pi}\left[R_t \frac{\nabla \pi(a_t \mid s_t, \theta)}{\pi(a_t \mid s_t, \theta)}\right],\end{aligned}$$

where R_t is the return as usuel. The expression in brackets is the quantity that can be sampled on each time step whose expectation is equal to the gradient. Thus REINFORCE update becomes

$$\theta_{t+1} = \theta_t + \alpha R_t \frac{\nabla \pi(a_t \mid s_t, \theta_t)}{\pi(a_t \mid s_t, \theta_t)} \tag{1.20}$$

Algorithm 8 REINFORCE: Monte-Carlo Policy-Gradient Control (episodic).

Require: a differentiable policy parameterization $\pi(a \mid s, \theta)$
  Initialize step size $\alpha > 0$.
  Initialize policy parameter θ.
  loop
    Generate an episode : $s_0, a_0, r_1, s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T$ following $\pi(\cdot \mid \cdot, \theta)$
    for each step of episode,$t = 0, 1, ..., T-1$: **do**
      $R \leftarrow \sum_{k=t+1}^{t} \gamma^{k-t-1} R_k$
      $\theta = \theta + \alpha \gamma^t R \nabla \ln \pi(a_t \mid s_t, \theta)$
    end for
  end loop

From the identity $\nabla \ln x = \frac{\nabla x}{x}$, we can use $\nabla \ln \pi(a_t \mid s_t, \theta)$ instead of $\frac{\nabla \pi(a_t|s_t,\theta_t)}{\pi(a_t|s_t,\theta_t)}$. Monte Carlos policy gradient pseudo-code is presented in algorithm 8.

REINFORCE with Baseline

The policy gradient theorem can be generalized to include a comparison of the action value to an arbitrary baseline $b(s)$, Sutton and Barto [2018]:

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a (q_\pi(s, a) - b(s)) \nabla \pi(a \mid s, \theta) \tag{1.21}$$

The update rule of the new version of REINFORCE that includes a general baseline:

$$\theta_{t+1} = \theta_t + \alpha (R_t - b(s_t)) \frac{\nabla \pi(a_t \mid s_t, \theta_t)}{\pi(a_t \mid s_t, \theta_t)} \tag{1.22}$$

One natural choice for the baseline is an estimate of the state value,$\hat{V}(s_t, w)$, where w is a weight vector.

REINFORCE is a Monte Carlo method for learning the policy parameter, θ, thus Monte Carlo method is used to learn the state-value weights, w. The pseudocode for REINFORCE with baseline using a learned state-value function as the baseline is given in the algorithm 9, this algorithm has two step sizes, denoted α^θ and α^w For policy parameter and state-value weights updates respectively.

Algorithm 9 REINFORCE: with Baseline (episodic), for estimating π_θ.

Require: a differentiable policy parameterization $\pi(a \mid s, \theta)$
Require: a differentiable state-value function parameterization $\hat{V}(s, w)$
Initialize step size $\alpha^\theta > 0$, $\alpha^w > 0$.
Initialize policy parameter θ and state-value weights w.
loop
Generate an episode : $s_0, a_0, r_1, s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T$ following $\pi(\cdot \mid \cdot, \theta)$
for each step of episode, $t = 0, 1, ..., T-1$: **do**
$R \leftarrow \sum_{k=t+1}^{t} \gamma^{k-t-1} R_k$
$\delta \leftarrow R - \hat{V}(s_t, w)$
$w \leftarrow w + \alpha^w \delta \nabla \hat{V}(s_t, w)$
$\theta = \theta + \alpha^\theta \gamma^t \delta \nabla \ln \pi(a_t \mid s_t, \theta)$
end for
end loop

Actor–Critic Methods

REINFORCE-with-baseline method learns a policy and a state-value function, but it is not an actor-critic method since the state-value function is not used for updating the value estimate for a state from the estimated values of subsequent states), but only as a baseline for the state whose estimate is being updated. REINFORCE with baseline as all Monte Carlo methods has slow learning and his estimates show a high variance, these inconveniences can be eliminated with temporal-difference methods.

One-step actor–critic methods, Sutton and Barto [2018], are analog to the TD methods such as TD(0), Sarsa(0), and Q-learning, they are fully online and incremental, they replace the full return of REINFORCE with the one-step return as follows:

$$\begin{aligned}
\theta_{t+1} &\doteq \theta_t + \alpha \left(R_{t:t+1} - \hat{V}(s_t, w) \right) \frac{\nabla \pi(a_t \mid s_t, \theta_t)}{\pi(a_t \mid s_t, \theta_t)} \\
&= \theta_t + \alpha \left(R_{t+1} + \gamma \hat{V}(s_{t+1}, w) - \hat{V}(s_t, w) \right) \frac{\nabla \pi(a_t \mid s_t, \theta_t)}{\pi(a_t \mid s_t, \theta_t)} \\
&= \theta_t + \alpha \delta_t \frac{\nabla \pi(a_t \mid s_t, \theta_t)}{\pi(a_t \mid s_t, \theta_t)}
\end{aligned}$$

Pseudocode for the One-step actor–critic methods is given in algorithm 10.

Algorithm 10 One-step Actor–Critic (episodic), for estimating π_θ.

Require: a differentiable policy parameterization $\pi(a \mid s, \theta)$
Require: a differentiable state-value function parameterization $\hat{V}(s, w)$
  Initialize step size $\alpha^\theta > 0$, $\alpha^w > 0$.
  Initialize policy parameter θ and state-value weights w.
  for each episode **do**
    Initialize s. {the initial state}
    $I \leftarrow 1$
    while s is not terminal **do**
      $a \sim \pi(. \mid s, \theta)$
      Take action a; observe reward r and next state s'.
      $\delta \leftarrow r + \gamma \hat{V}(s', w) - \hat{V}(s, w)$
      $w \leftarrow w + \alpha^w \delta \nabla \hat{V}(s, w)$
      $\theta = \theta + \alpha^\theta I \delta \nabla \ln \pi(a \mid s, \theta)$
      $I \leftarrow \gamma I$
      $s \leftarrow s'$
    end while
  end for

1.4 Hierarchical Reinforcement Learning

Reinforcement learning encounters a dimensionality problem which is called the curse of dimensionality. This term was coined by Richard Bellman and refers to phenomena observed when analyzing and organizing data in high dimensional spaces wherein, as the problem dimensionality increases, the volumes of the search space for solutions increases so fast that the available data becomes sparse resulting in more efforts for finding an optimal solution. In the context of reinforcement learning when the environment becomes complex or the number of states and actions is very large, This leads to a combinatorial explosion which makes learning slow or inefficient. Another type of reinforcement learning problem relates to its inability to coordinate different scales of representation. For example, when an agent has to navigate between different positions inside a room, and then between different places in a city, the human designer has to manually predefine this change in scale because it cannot be discovered by the learning algorithm. One solution consists in dividing the problem to solve into several levels, and several subproblems at each level, which makes the algorithm hierarchical. For this reason, this method is called "hierarchical reinforcement learning" (HRL), McGovern et al. [1998], Precup and Sutton [1998], Barto and Mahadevan [2003].

It seems natural to try to use a hierarchical decomposition of the problem into smaller

pieces that are easier to solve individually, since humans naturally use hierarchical representations to act in the world. In some cases, hierarchical decomposition can be designed by a human operator. But in many situations such as large, complex or unusual problems, providing the correct decomposition is difficult if not impossible. That is why the automatic hierarchical task decomposition is one of the most challenging problems in the domain of RL, Kozlova [2010].

1.4.1 Semi-Markov Decision Process

Semi-Markov Decision Process (SMDP), Sutton et al. [1999] can be seen as MDP where the amount of time taken by an action is a random variable, which can take either real or integer values. In the real-valued case, the SMDPs model continuous-time discret-event systems, Puterman [1994]. In the case of integer values, SMDPs model discret-time systems, Howard [1971], and that is what we deal with in this thesis. Due to its relative simplicity, the discrete-time SMDP formulation underlies most approaches to hierarchical RL, but there are no significant obstacles to extending these approaches to the continuous-time case, Barto and Mahadevan [2003]. The actions extended in time in SMDPs are also called abstract actions.

Let the random variable τ be the positive waiting time for state s when action a is executed. Given an agent in state s, the joint probability of transiting to state s' after τ time steps when action a is executed is written $P(s', \tau \mid s, a)$. The immediate rewards expected, $r(s, a)$ is the amount of discounted reward expected to accumulate over the waiting time in s given action a. The Bellman equations for V^* and Q^* become

$$V^*(s) \leftarrow \max_{a \in A(s)} \left[r(s,a) + \sum_{s',\tau} \gamma^\tau P(s', \tau | s, a) V^*(s') \right], \tag{1.23}$$

for all $s \in S$; and

$$Q^*(s,a) = r(s,a) + \sum_{s',\tau} \gamma^\tau P(s', \tau \mid s, a) \max_{a' \in A(s')} Q^*(s', a'), \tag{1.24}$$

for all $s \in S$ and $a \in A(s)$.

1.4.2 Options

The term options is used for the first time in Sutton et al. [1999], as the generalization of primitive actions to include temporally extended courses of action. Options consist

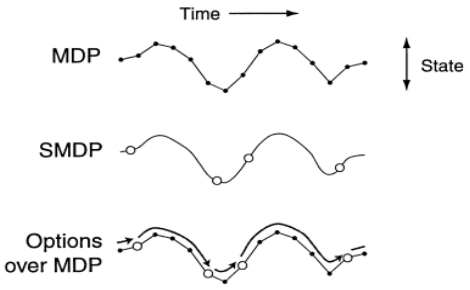


Figure 1.2: Discrete-time transitions in MDP vs SMDP and options, Sutton et al. [1999]

of three components:

- a policy $\pi : S \times A \rightarrow [0, 1]$,
- and an initiation set $I \subseteq S$
- a termination condition $\beta : S \rightarrow [0, 1]$,

An option $o = \langle I, \pi, \beta \rangle$ is available in all states s belonging to the initiation set I. Thus, if an agent take the option o in a state $s_t \in I$, then actions are selected according to π until the option terminates stochastically according to β. In particular, a Markov option executes as follows. First, the next action a_t is selected according to probability distribution $\pi(s_t, .)$. The environment then makes a transition to state s_{t+1}, where the option either terminates, with probability $\beta(s_{t+1})$, or else continues, determining a_{t+1} according to $\pi(s_{t+1}, .)$, possibly terminating in s_{t+2} according to $\beta(s_{t+2})$, and so on. When the option terminates, the agent has the opportunity to select another option.

It has a great advantage to limit the field of application of an option by the initiation set and the termination condition, this amounts to defining a sub-problem much less complex than the main problem and with a search space very limited which solves the problem of dimensionality mentioned above. Thus an option policy will be defined on a limited space as in the example cited in Sutton et al. [1999] where a handcrafted policy π for a mobile robot to dock with its battery charger might be defined only for states I in which the battery charger is within sight. The termination condition β could be defined to be 1 outside of I and when the robot is successfully docked.

A Markov option is an option in which all states where an option could continue are also states where the option could be taken or started. In a Markov option that start at time t and take some actions for a number of steps k; the decision in one step i, $t \leq i \leq k$ may depend only on s_i, whereas the decision in a semi-Markov option may depend on the entire history from t to i. Thus in semi-Markov option, Sutton et al.

[1999] the policy and termination condition are function of possible histories. Note that options that select other options in hierarchical structures are also semi-Markov.

From the initiation sets of options is defined a set of available options $\mathcal{O}_s$ for each $s \in S$, thus primitive actions available in state s could be considered as options of one step. So we can consider the agent's choice at each time to be entirely among options, some of which persist for a single time step, others of which are temporally extended, Sutton et al. [1999]. The set of all these available options is noted $\mathcal{O} = \bigcup_{s \in S} \mathcal{O}_s$.

Another important concept is policies over options, Sutton et al. [1999]. When initiated in a state s_t , the Markov policy over options $\mu : S \times \mathcal{O} \to [0,1]$ selects an option $o \in \mathcal{O}_{s_t}$ according to probability distribution $\mu(s_t, \cdot)$. The option o is then taken in s_t , determining actions until it terminates in s_{t+k}, at which time a new option is selected, according to $\mu(s_{t+k}, \cdot)$, and so on, see figure 1.2. In this way a policy over options, μ, determines a conventional policy over actions, or flat policy, $\pi = flat(\mu)$, which may not be Markov even though all the options it selects are Markov, when any of the options are temporally extended. The action selected by the flat policy in state s_i depends not just on s_i but on the option being followed at that time, and this depends stochastically on the entire history since the policy was initiated at time t. These policies that may depend only on events back to some particular time are semi-Markov.

Thus, the generalizations of the conventional value functions for a semi-Markov flat policy π, in a state $s \in S$ is given as the expected return given that π is initiated in s :

$$V^{\pi}(s) = E\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \mid \mathcal{E}(\pi, s, t)\right\}, \tag{1.25}$$

where $\mathcal{E}(\pi, s, t)$ denotes the event of π being initiated in s at time t. The value of a state under a general policy μ can then be defined as the value of the state under the corresponding flat policy : $V^{\mu}(s) =^{def} V^{flat[\mu]}(s)$, for all $s \in S$. Action-value functions generalize to option-value functions. Sutton and colleagues in Sutton et al. [1999] define $Q^{\mu}(s, o)$, the value of taking option o in state $s \in I$ under policy μ, as

$$Q^{\mu}(s, o) = E\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \mid \mathcal{E}(o\mu, s, t)\right\}, \tag{1.26}$$

where $o\mu$, the composition of o and μ, denotes the semi-Markov policy that first follows o until it terminates and then starts choosing according to μ in the resultant state.
The optimal policy over a set of options $\mathcal{O}$ can be learned by generalizing learning methods on actions, since each option is viewed as an indivisible, opaque unit. When the execution of option o is started in state s, we next jump to the state s' in which

o terminates. Based on this experience, an approximate option-value function $Q(s,o)$ is updated after each option termination as in one-step Q-learning, which is called in Sutton et al. [1999] SMDP Q-learning :

$$Q(s,o) \leftarrow Q(s,o) + \alpha \left[r + \gamma^k \max_{o' \in \mathcal{O}_{s'}} Q(s',o') - Q(s,o) \right], \tag{1.27}$$

where k denotes the number of time steps elapsing between s and s', r denotes the cumulative discounted reward over this time, and α is step-size parameter.

1.5 Intrinsic Motivation

The behavior of human or animal is referred to be intrinsically motivated when it is inherently enjoyable, Barto et al. [2004], Vigorito and Barto [2010]. So, humans or animals engage for activities as exploration, play, and other behaviors driven by curiosity, for their own sakes, without expecting an external explicit reward. Intrinsically motivated behavior is essential to accumulate knowledge and competences for solving future problems. Earlier in psychology, Harlow Harry. F and R. [1950], it has been shown that young human or animal subjects often tend to exhibit intrinsically motivated behavior when they are not preoccupied with foraging, survival, or reproduction. This allowed them, over time, to acquire complex skills and know-how that they can benefit from in their future lives, Vigorito [2016].

1.5.1 Intrinsic vs extrinsic motivation

While intrinsic motivation is defined as the doing of an activity for its inherent satisfaction, the extrinsic motivation is when an activity is done in order to attain some separable outcome, Oudeyer and Kaplan [2009]. Thus, instrumentalization or goal is the central feature that differentiates intrinsic and extrinsic motivation. The experimental psychology, Harlow Harry. F and R. [1950], Oudeyer and Kaplan [2009] revealed that exploratory activities are more likely to be intrinsically motivated, thus some properties like (the 'fun', the 'challenge', the 'novelty', the 'cognitive dissonance' or the 'optimal incongruity') are crucial to the definition of intrinsic motivation.

1.5.2 Intrinsically motivated reinforcement learning

Early work on intrinsic motivation focused on approaches that provide intrinsic reward to agents proportional to errors in the predictions of their environment, leading the agent to discover new areas of the environment, thereby focusing learning on those areas so as to reduce that unpredictability, Schmidhuber [1991]. In reinforcement learning, the extrinsic reward is done by the critic as an environment response associated with the task to be solved, but the intrinsic reward is function of the agent's current state of knowledge, and thus, the reward function is continually changing as the agent continues to learn.

In Barto et al. [2004], authors have used intrinsic reward to learn a collection of skills in a very hierarchical environment as the playroom domain described in, Barto et al. [2004], thus when the agent encounters an unpredicted salient event, it is intrinsically rewarded few times for the same event, the intrinsic diminishes and the agent gets "bored" with that event, so it moves for another activity when it encounters another unpredictable salient event using skills already learned, and so on. complex skills are learned faster when agent learn sub-skill using intrinsic reward.

Authors in Şimşek and Barto [2006] define two value functions, and therefore two policies, task policy for solving the task MDP and the behavior policy for improving the agent behavior. This last policy is based on an intrinsic reward defined on the base of the improvement in the the task policy value function, thus, each time the improvement in that value function is significant, the agent will have more chances of having a significant positive intrinsic reward. As a result, the agent learns to focus its exploration in regions where the learning updates improve the agent value function the most.

In Vigorito and Barto [2010], the authors use the intrinsic reward to accumulate structural knowledge during the exploration of the environment, thus, the agent builds options incrementally, each option in this framework corresponds to a controllable variable , the discovered options are used to gradually discover other options until all the complex structure of the environment is discovered.

1.6 Wayfinding

Orientation behavior is an intentional, directed and motivated movement from an origin to a specific distant destination, and which cannot be directly perceived by the traveler. This involves interactions between the traveler and the environment Allen [1999] and Golledge [1999]. The ultimate goal of human orientation is to find the way from one

place to another. The traveler must be able "to achieve a specific destination within the confines of pertinent spatial or temporal constraints and despite the uncertainties that exist". The sense of orientation is necessary when the human has only a partial description of his environment and he must navigate in an extended space to collect the information allowing him to reach his destination. Allen [1999] proposes a taxonomy of orientation tasks based on functional objectives. and consists of three categories:

- traveling with the aim of reaching a familiar destination,
- exploratory journey with the aim of returning to a familiar point of origin,
- travel with the aim of reaching a new destination.

1.6.1 People's wayfinding abilities and spatial knowledge

According to Golledge [1999], the origin of human orientation is due to the cognitive and behavioral abilities of people to find a path from an origin to a destination. These abilities allow people to use environmental information and/or knowledge in the head to successfully orient. Allen's work Allen [1999] groups people's spatial abilities according to the tasks and situations in which they are applied. This classification is based on previous research in the psychometric, information processing, developmental, and neuropsychology traditions. It consists of interactions between

- a fixed observer and small manipulable objects,
- an observer and moving objects,
- a mobile observer and large fixed objects.

Although there are encounters with people and moving objects, orientation in a building mainly concerns the third group because people move through an environment that contains large, stationary objects. The foundation of this group of spatial abilities is based on sensitivity to available perceptual information Martin [2001].

People's spatial abilities seem to depend mainly on the following four interactive resources: perceptual abilities, basic information processing abilities, previously acquired knowledge and motor abilities Allen [1999]. These resources support different means of orientation.

Cognitive abilities also depend on the task at hand. Finding your way around a road network requires a different set of cognitive abilities than navigating from one room to

another in a building. People are generally good at applying their individual skills to the task at hand. If their spatial skills are weak, they use verbal skills to navigate, when people get lost, they usually ask for help Martin [2001].

1.6.2 Spatial reasoning and decision making

People use topological information instead of metric information. The topological properties of objects remain invariant under transformations such as translations, rotations, and scalings. Piaget and Inhelder [1967] demonstrated that the fundamental spatial concepts are topological, but not at all Euclidean. They showed that children begin to conceptualize space by constructing and using elementary topological relationships, such as proximity, separation, order, and closure. Golledge and Stimson [1997] argue that in many cases human decision-making is not strictly optimizing in an economic and mathematical sense - as the algorithms of classical decision-making theories propose - and therefore emphasize on behavioral decision theory, Martin [2001].

Instead of making exact calculations, people apply qualitative methods of spatial reasoning, Frank [1992], Martin [2001], that rely on relative rather than absolute magnitudes and values. When people perceive space through different channels, they arrive at different types of information, which are usually qualitative in nature.

1.6.3 Cognitive maps

When people travel with the goal of reaching a familiar destination, they use enduring internal representations of spatial knowledge of the orientation environment. A useful metaphor suggests that people have a cognitive map in their head, Kuipers [1982], a mental representation that matches people's perception of the real world. The term cognitive map was first used in an article by TOLMAN [1948]. People construct and develop their cognitive maps based on recording information through perception, natural language, and inference. Complex environmental structures can lead to slower development of cognitive maps and also representational inaccuracies Martin [2001].

1.6.4 Human orientation performance

Performance in human orientation relies on collecting individuals' perceptions of distances, angles, and locations. Lynch in his book Lynch [1964], divided the content of city images into paths, edges (boundaries), regions, nodes and landmarks. Weisman

[1981] identified four classes of environmental variables that influence wayfinding performance in built environments: 1. visual access, 2. degree of architectural differentiation, 3. use of signs and numbers to provide identification or orientation information, and 4. map configuration. people's familiarity with the environment and frequency of previous use also have a large impact on orientation performance.

1.6.5 Computer models for orientation

Cognitive computer models typically simulate a guide that can solve route planning tasks using a cognitive map-like representation. The goal of these models is to discover how spatial knowledge is stored and used, and what cognitive processes act on it. One can distinguish between computational process models where cognition is conceptualized as sets of rules acting on symbolic representations, and biologically inspired models that model cognition using lower-level physiologically plausible mechanisms.

Among the first computer models is the TOUR model, Kuipers [1978], which simulates learning and problem solving while traveling in a large-scale urban environment. The TOUR model is based on the cognitive map and divides knowledge into five categories: 1. routes, 2. a network of streets, 3. the relative position of two places, 4. the division of borders, and 5. regions.

Other cognitive computer models exist, such as TRAVELER Leiser and Zilbershatz [1989], SPAM McDermott and Davis [1984] and ELMER, McCalla et al. [1982], which simulate learning and problem solving in spatial networks. ARIADNE, Epstein [1997] learns the facilitators and obstacles for pragmatic two-dimensional navigation. NAVIGATOR, Gopal et al. [1989], Gopal and Smith [1990] constructs a hierarchical cognitive map based on spatial learning. Orientation learning is performed using spatial and temporal measurements. O'Neill [1991] presents a model of spatial cognition and orientation based on the biological approach.

These computer models mainly perform the creation and exploration of the cognitive map; they largely neglect the processes by which people immediately perceive and assign meaning to their spatial environments as they navigate through them Martin [2001].

1.7 Conclusion

In this chapter, we have introduced the fundamental concepts necessary for our work on the automatic discovery of options in hierarchical reinforcement learning. We have cov-

ered the basics of Markov Decision Processes, dynamic programming, key RL algorithms, and the principles of hierarchical learning through options and semi-MDPs.

Moving forward, we will employ the option framework as a core component of our approach. Additionally, intrinsic motivation and a sense of direction will be utilized in our second contribution to enhance the learning process. However, before diving into our contributions, we will review the state of the art on existing methods for option discovery.

2 Related works

With the emergence of the HRL, the decomposition of problems, and the choice of options was done manually, which quickly became a problem, hence the emergence of automating this process as a new field of research.

New works has been done in this new field, consisting of one side automatically discovering the hierarchical levels of a problem, and / or discovering the options of a given level on the other side, and it is the latter under fields that interests us in this thesis. Much work has been done in the field of automatic option discovery, and many approaches have been used, some researchers use the graphical representation of MDPs, and graph theory to derive important features from states, to infer options as in Hengst [2002], others uses theoretical foundations for the efficient processing of information as in van Dijk and Polani [2011], other research are based on probabilistic models and causal network representations, Vigorito and Barto [2010], more recent research are based on mathematical models applied to spatial information, Machado et al. [2017], X. Xu and Li [2018].

In this chapter, we propose a detailed study of recent methods in the field of the automatic discovery of options, beginning with a classification that we propose, then we explain each method separately, and we end with a comparative and summary study.

2.1 Classification of approaches

Discovering option is to discover all option parameters, initiation states, terminate state, and the option policy. We distinguish two large classes according to the order of discovery of the parameters, which are the greedy methods and the two-step methods, we deduce another class which is a kind of hybridization of the first two classes. Figure 2.1 illustrates our classification of the existing approaches.

2.1.1 Greedy methods

In these methods, the policy of the option is discovered along with the states of the option, which are the initiation states and the terminal state which corresponds to the sub-goal of the option. These methods are usually powerful and suitable for problems with a very large state and action spaces. Among these methods we cite Leon and Denoyer [2018], Brunskill and Li [2014], van Dijk and Polani [2011], Vigorito and Barto [2010].

2.1.2 Two-step methods

As the name suggests, these methods take place in two steps, the first step is to discover the states of the option, the most common is to discover the terminal state first, then the set of initiation states. The second step consists in applying a reinforcement learning algorithm on the discovered states to learn the policy of the option, the majority of the methods of this class are graphical methods and have very satisfactory results, of which we quote, Hengst [2002], Şimşek and Barto [2009], X. Xu and Li [2018], Lakshminarayanan et al. [2016]

2.1.3 Hybrid methods

These methods generally start by discovering the terminal states of the options in the first step, then the policy is learned in parallel with the discovery of the initiation states, these methods take advantage of the first two classes and show very good results as in Machado et al. [2017].

2.2 Approaches of automatic option discovery in hierarchical reinforcement learning

2.2.1 BHNN: Budgeted Hierarchical Neural Network

In this work, Leon and Denoyer [2018], a neural network is applied on a particular case of partially observed MDP. Here the agent always observes x_t and can require another observation y_t considered to be supplementary but necessary to have a good choice for the next action. Budgeted Hierarchical Neural Network (BHNN) is composed of three parts; option model, actor model and acquisition model, and its architecture is close to

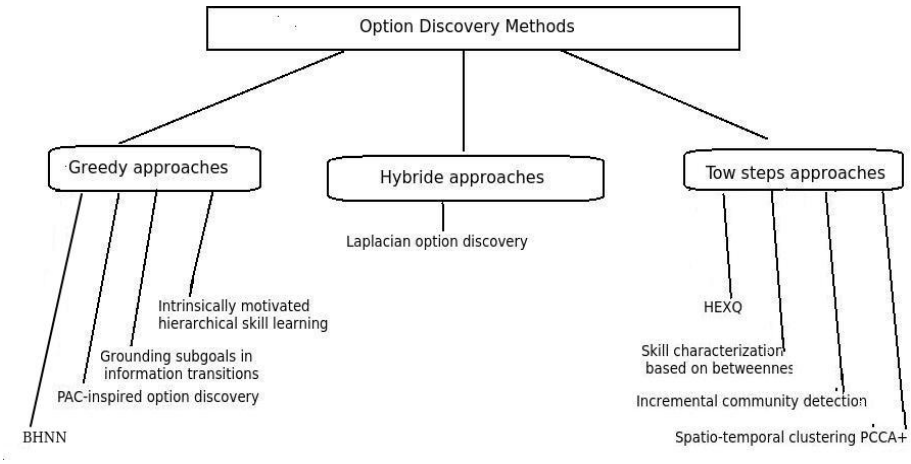


Figure 2.1: Classification of the existing approaches of automatic option discovery

hierarchical recurrent network with two hidden states o_t for option model, and h_t for actor model.

Acquisition model

Decides whether a new high-level observation y_t is required, the decision depend of the value of $\sigma_t \in \{0, 1\}$ that follows a Bernoulli distribution, thus only if $\sigma = 1$ the agent uses the new high-lelvel observation y_t to compute a new option o_t, in the other case it uses only x_t to decide for the next action.

Option model

Computes a new option if $\sigma = 1$ using a Gated Recurrent Unit, Cho et al. [2014], (gru_{opt}) that has as inputs the observations x_t and y_t and the last computed option o_{last}.

Actor model

Updates the actor state and computes the next action in the current option. The update of actor state depends on the acquisition model and is decided by a Gated Recurrent Unit (GRU), (gru_{act}) that has as input the both observations x_t and y_t. The softmax distribution of the actor model gives next action to be executed.

Option discovery

Each time the model choose to acquire a high-level observation, it computes a new intrinsic option when $\sigma = 1$. Then the policy will behave as a classical recurrent policy until another new observation y_t will be acquired. Since the option is defined directly according to the actual observations, there is no need to have an explicit initiation set. BHNN discovers hierarchy since it discovers when the high-level observation has to be acquired and when to start a new sub policy. In BHNN the acquisition cost is integrated to the immediate reward, the associated discounted return will be the new objective to be maximized using the policy gradient algorithm, Wierstra et al. [2010] and A3C algorithm, Mnih et al. [2016]

Results

BHNN has been compared to a recurrent policy gradient (R-PG) method, Wierstra et al. [2010] which is implemented with GRU cells and use x_t and y_t at every time step, and with Fine Grained Action Repetition (FiGAR), Sharma et al. [2017], which enables the agent to decide how many times it will repeat the chosen action, and uses similar network architecture. Experiences were applied on this approach were tested on three environments; Cart Pole, Lunar Lander, and multi-room Maze. In each environment the observation y_t depend on the environment, for example in multi-room maze y_t contains the observed doors. And as result, BHNN need only a few amount of high-level observation in the CartPole environment to have the same accuracy as FiGAR. BHNN succeed in discovering options with random initiations of the environments at each experience.

2.2.2 HEXQ Discovering hierarchy in reinforcement learning

HEXQ, Hengst [2002] is hierarchical reinforcement learning algorithm that automatically decomposes the MDP hierarchically and discovers and learns options in each level of the hierarchy. This approach is applied on taxi domain and Tower of Hanoi Puzzle for the experimentations.

Automatic hierarchical decomposition

HEXQ decompose the MDP on levels where the number is the same number of the state variables. The levels are then ordered in the way that the bottom level is associated to the state variable that changes value most frequently; this level only interacts with

the environment and uses primitive actions. The next level is associated to the second variable in the frequency of value changes, and so on until the top level. This primary decomposition needs a random exploration of the environment by the agent to take statistics on the frequency on state variables changes.

Option discovery

After the hierarchical decomposition, HEXQ starts with the first level associated with the most frequently value changeable variable, another exploration is needed for the modelization of state transitions and rewards for this limited MDP. So a Directed Graph (DG) is generated, in which the state values represent vertices and the transitions associated with primitive actions represent edges. The exploration results also on some unpredictable transitions (called exits) that will not belongs to the graph. An entry state is defined to be a state that can be reached after an exit, for example, for the Taxi location variable; all states are entries because the Taxi agent can start at any location. The exit transitions are not predictable because they are caused by changing in state variable belonging to a higher level.

The DG of the actual level is decomposed into Strongly Connected Components (SCCs), the SCCs connected form regions, and the regions are abstracted to form higher level states. The resulted regions can be connected by edges, these connexions could be broken to form additional exits and entries. This process is repeated until no additional regions are formed. Options are then formed on each region, the exits will form the sub-goals of the options, so one option is constructed for each exit in each region. The option policy is next learnt on line.

For the next level, regions already discovered form abstract states and policies form abstract actions, the same process of discovering regions and policies is repeated for higher levels until achieving the top level. Hengst defined here a recursively optimal hierarchical value function that computes automatically the value function on any level in the hierarchy.

Results

HEXQ experiments and trials perform better results than the flat version of reinforcement learning and MAXQ (the predecessor of HEXQ). HEXQ converge rapidly after the hierarchy decomposition and the sub MDP discovering and become faster than the flat algorithm and MAXQ. Another point is that all the steps of the method are performed automatically, and no parameter is given manually, so the agent is totally autonomous.

2.2.3 PAC-inspired option discovery in lifelong reinforcement learning

In this approach, Brunskill and Li [2014], the authors provide theoretic elements for measuring the learning speed of RL with options, by defining the sample complexity. The sample complexity of an RL algorithm A in an SMDP is the number of epochs where A takes non -ϵ- optimal actions, calculated as follows;

$$\sum_t \tau_t . \mathtt{I}\left(V^{A_t}(s_t) \leq V^*(s_t) - \epsilon\right), \tag{2.1}$$

where

$$\mathtt{I}(c) = \begin{cases} 1 & if\, c\, occure \\ 0 & otherwise \end{cases},$$

with $\epsilon > 0$ fixed, and τ_t the duration of the option selected at the epoch t.

Then, the authors define an RL algorithm to be Probably Approximately Correct (PAC) in SMDP denoted PAC-SMDP when the sample complexity is bounded by some polynomial function, see Brunskill and Li [2014] for detail. The authors propose an option discovery algorithm, thus the options discovered are those who minimize the sample complexity of future RL tasks and assure a high quality performance.

The PAC-SMDP algorithm operates in two phases, in phase 1, the algorithm performs E3 RL algorithm, Kearns and Singh [2002] for tasks which results in MDPs with learned policies, on which the option discovery algorithm could be performed. In phase 2 the discovered options are used for new tasks by applying SMDP-Rmax algorithm, Brafman and Tennenholtz [2003] for high level reinforcement learning.

Option discovery

The option discovery algorithm requires as input a set of MDPs. The options are discovered in scalable greedy manner with one initiation state for each option. The algorithm chooses repeatedly an initiation state and tries to build an option in order to cover e-optimal policies of a subset of state-MDP pairs. The option grows by adding reachable states, so the algorithm follows the actual option policy and add to the option new states reachable by this policy. Next the option policy is determined for each new state, thus an actions are assigned for each state successor, the first action that improve the sample complexity for the set of all options is selected. If no option can improve this value then all new states are considered as terminal states and the option is added to the options

set.

Results

For the experiments, the algorithm is tested on four-room maze, Sutton et al. [1999]. In phase 1 the agent learns the policies of 40 MDPs, on the same structure with difference in reward distributions, using the E3 PAC MDP algorithm for each task. The resulting policies are used for executing the option discovery. In phase 2, the SMDP-RMAX algorithm is used on the discovered options to learn new tasks. This algorithm was compared to primitive action only algorithm, Hand Coded option , and Policy Blocks algorithm, Pickett and Barto [2002], PAC inspired algorithm have better result in term of rewards than primitive action and Policy Blocks algorithms, and it is better than Hand Coded method because of autonomy of the PAC inspired agent, and the difference in the results is minimal.

2.2.4 Laplacian Framework for option discovery in reinforcement learning

In this work, the authors Machado and colleagues, Machado et al. [2017] use Proto-Value Functions (PVFs) to discover options in HRL. PVFs were already introduced in learning representation for RL in, Mahadevan [2005]. In this approach, PVFs are calculated using the normalized graph Laplacian;

$$L = D^{-\frac{1}{2}} (D - A) D^{-\frac{1}{2}}, \tag{2.2}$$

where A is the MDP's graph adjacency matrix, and D the diagonal matrix where the entries are the sums of rows of A. PVFs are the eigenvectors of L, so they have the advantage that they capture the characteristics and the geometry of the environment like symmetries and bottleneck.

Option discovery

For discovering options, the authors define the eigenpurpose as the intrinsic reward function of PVF. Eigenpurposes are goals for options to be discovered, this makes options independent of the MDP task, and thus options are related only on the environment structure. For learning option, an MDP is associated to a purpose, defined as the original MDP except the reward function, it is replaced by the eigenpurpose, and a new

action "terminate" is added to the set of actions. Learning can now be applied using value iteration algorithm or Qlearning, the policy determined is called eigenbehavior. For the determination of option initiation and termination states, the agent execute the action terminate when he reaches the largest value in the eigenpurpose (or a local maxima), achieving this termination state, any further reward will be negative, thus, any state where there is an action with a positive value mean that the agent can achieve the purpose from it, so it belong to the initiation set of the option called here the eigenoption.

Results

This approach is experimented on I-maze, open room, and the 4 room domain. Eigenoptions are associated to eigenpurposes that correspond to the smallest eigenvalues. Eigenpurposes are not necessarily bottleneck states, this make the approach useful for environments where there are no bottlenecks, and the authors proved the efficiency of these options for exploration because of their disponibility in all the state space and at different time scales. The experiments show a large number of options discovered for the domains cited above. The eigenoptions could be used in other tasks in the same environment since they are task independent. In complex problems, it will be impossible to use adjacency matrix for the representation of MDP, so an approximation method is proposed for the eigenoption discovery. The algorithm was applied on Freeway and Montezum's Revenge games and had better result than random walk methods.

2.2.5 Skill characterization based on betweenness

In this approach, Şimşek and Barto [2009] contribute with a new method for discovering options based on "betweenness", a measure of centrality on graphs, this measure help to find bottlenecks which will be considered as subgoals or terminate states for options. Considering an MDP, an interaction graph is the directed graph generated by the agent interaction with its environment; such that, the vertices represent the states of the MDP and the edges represent the transitions between vertices. So the value of betweenness of a vertice v is computed by the equation :

$$\sum_{s \neq t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}} w_{st}, \tag{2.3}$$

where σ_{st} is the number of shortest paths from the state s to the state t, $\sigma_{st}(v)$ is σ_{st} passing through the state v, and w_{st} is a weight assigned to paths from s to t. The

weights are a representation of the reward function of the MDP.

Option discovery

The option discovery starts with the discovery of subgoals, which are terminate states of options, these states correspond the local maxima of betweenness. The initiation set are constructed of states with a little distance to terminate states if the domain is very large, else the options can be available on all states of the domain. The option policies are formed by the optimal trajectories for achieving subgoals. On the HRL top level learning, the authors use Q-learning algorithm for determining the top level policies.

Experiments and discussion

The method is experimented on taxi domain Hengst [2002], play room domain, Barto et al. [2013], Tic-Tac-Toe game, and rooms domain. Local maxima of betweenness led to bottleneck for every domain, which are doors in room domain, forks in Tic-Tac-Toe game and changes in some variable in play room and taxi domains. For HRL top level learning, the results of learning with the options generated by the betweenness technique are compared with learning with options generated randomly and with learning based on primitive actions only. The betweenness based approach has obtained significantly better results in all domains. However, the limitation of the approach resides in the fact that to compute the betweenness we need a complete knowledge of the entire interaction graph, and the cost time become very high in complex domains. As a solution the authors proposed an incremental algorithm that search for local maxima in sub-graphs instead of the global graph, repeated local maxima on state make of that stat a subgoal. This method is tested on grid world domain, and the algorithm succeeds to identify subgoals.

2.2.6 Constructing option through incremental community detection

The work of X. Xu and Li [2018] has two objectives, to create options automatically, and to update and optimize options during an online learning. For the first objective, the first step is the detection of communities on the state transition graph constructed from a prior exploration of the environment, at the next step; options are constructed from these communities.

Community detection

The detection of communities consists of partitioning the transition graph on sub graphs, each subgraph will be a community. The connections intra-community are dense and connections inter-community are sparse, there are no intersection states between communities, and the union of all communities results on the initial graph. To construct communities, the authors use the Louvain algorithm, which is an incremental algorithm starting with considering each state as a separate community, and greedily gathers adjacent communities to ensure better modularity measure. The modularity measure serves to evaluate the quality of a community, and is calculated by

$$Q\left(P\right) = \sum_{C_k \in P} \left(\frac{m(C_k)}{M} - \frac{d\left(C_k\right)^2}{4M^2} \right), \tag{2.4}$$

where $m(C_k)$ is the total of internal edges of the community C_k, M the total number of edges in the initial graph, and $d(C_k)$ is the sum of the degrees of all vertices of C_k.

Option discovery

For the option discovery, the authors create an option between every two adjacent communities, such that the initiation set is composed of all states in the first community, and the termination condition is defined by the first states achieved in the second community. For the generation of the option policy, the authors use the Experience Replay (ER) process, the ER use trajectories generated in the community detection phase for reinforcement learning with a completion reward and the environment reward to generate the optimal policy.

Updating option for an online hierarchical reinforcement learning

For the complex problems, it is impossible to have the entire transition graph in the first exploration, so an incremental learning is needed. For this solution, the first exploration result in a sample graph on which the Louvain algorithm is applied for detecting communities and so discovering options. In the next step during learning, the agent can discover new states and periodically ask for updating the community partition. Seeing the position of the new state, the algorithm can decide to create a new community, to join two communities, or do not give change in the partition. For each case a new value of modularity measure is computed. The policies of the options modified are updated and if a new option is discovered, the ER process is applied for the policy learning.

Results

The option discovery process is tested on the four rooms' domain and compared with Q-primitive RL, manual option agent and the betweenness agent, Şimşek and Barto [2009]. The Louvain option agent and the betweenness agent have the same better result than the q-primitive agent, and catch the manual option agent after some learning episodes. The incremental approach was tested on Pac-Man game; compared with the Q-primitive learning, the incremental Louvain method had faster learning speed and higher average score.

2.2.7 Grounding subgoals in information transition

In this work, van Dijk and Polani [2011], the authors introduce a new theoretic foundation based on Shannon information theory to define subgoals, so they define the Relevant Goal Information (RGI) to be the quantity of goal information required by the agent to maintain and perform its policy. The RGI is defined as the mutual information between the goal G and the action A_t at a state s_t, and is formulated as follow;

$$I\left(G; A_t \mid s_t\right) = H\left(A_t \mid s_t\right) - H\left(A_t \mid G, s_t\right), \tag{2.5}$$

where $H\left(A_t \mid s_t\right)$ is the conditional entropy.

To discover options, the authors start by discovering subgoals or terminate states of options. The achievement of a subgoal marks a qualitative transition of the RGI, for example, an agent want to achieve a room entry in a maze, only when this task is terminated, he can enter the room and search for goal. So when a transition in the relevant goal information is high, a subgoal is achieved.

Option discovery

The algorithm is multi-goal, thus it starts with the initialization where the number of goals is fixed, and then operates in two phases;

- (i) Learning phase; a new goal g is selected and a policy p is learned using hierarchical $Q(0)$ algorithm.
- (ii) Subgoal discovery; a number of trail run are executed, each execution is related to one goal gr already learned. The start position is random and the option to execute is selected from the set of available options according to the policy of

the goal gr, the goal distribution is updated at each execution and the RGI is calculated. After that executions, the states with interesting transition are selected as subgoals, so options are created and learned with all states of the domain as initiation sets and $Q(0)$ algorithm with primitive action only. These options will be available for learning hierarchical policies for future goals.

- The algorithm loop from phase (i) until the number of goals is achieved.

Results

The algorithm was tested on 6-room domain and as result the learning is accelerated comparing to flat learning after some episodes with a negative skill transfer, the agent rapidly success to avoid these bad options and accelerate learning.

2.2.8 Intrinsically motivated hierarchical skill learning

In this work, Vigorito and Barto [2010], the authors Vigorito and Barto present an incremental algorithm for option and environment structure learning. The learning agent uses an intrinsic reward for the option learning. The structured environment is modeled as a Factored MDP (FMDP) see Vigorito and Barto [2010], Boutilier et al. [1995], .T et al. [2006] for detail. FMDPs are presented by Dynamic Bayesian Networks (DBNs). Thus, for each action executed is associated a DBN which is a two layer directed acyclic graph, the first layer represents the FMDP variables at time t, and the second layer is for the same variables at time t+1, one edge from a node of the first layer to a node belonging to the second layer designates a dependency between the associated variables given the action executed. For each DBN, there is one Conditional Probability Tree (CPT) for each variable S_i in the DBN. The leaves of CPT contain the probability distributions over the domain of the variables S_i. To learn the structure of DBNs with their CPTs, the authors use an approximation method, that is the Bayesian Information Criterion (BIC) and the network is constructed incrementally in a greedy manner, so when a new data point is added to a leaf of a CPT, the BIC score is computed on that leaf, and for all resulted leafs of each possible refinement (split of a leaf in new leafs according to the distribution of the input value of the variable considered in that first leaf).

Option discovery

The options are constructed incrementally based on the structural information actually available. The options use an active learning with intrinsic reward, and are subsequently used for learn more of the environment. The intrinsic reward changes as the agent learns to give more importance for the exploration of new structures, so to perform an active learning. The algorithm of option discovery maintains a set C of controllable variables; one option will be associated to each possible value of each controllable variable discovered.

The set C is initially empty, the execution start with a local active learning algorithm using primitive actions, after each action, the CPTs are updated for each variable S_i, and a BIC score is computed for each new refinement, thus refinements with best BIC scores are kept and the corresponding variables becoming controllable are added to C, if all ancestors of the new controllable variable are controllable than new option is created for each leaf value as a subgoal. The option policy is then learned using the Structured Value Iteration (SVI) algorithm, once an option created, it can be selected in next steps or iterations for discovering more other options hierarchically resulting in discovering options and hierarchy at the same time. When an option is selected, it is executed to completion, if it fails repeatedly to reach the subgoal after a fixed number of steps, the option will be eliminated.

Results

The algorithm was tested on the light box domain which has strict hierarchical dependencies in the structure. The algorithm of intrinsically motivated active hierarchical learning succeeds to have far better results in finding correct refinements than the random agent and the non hierarchical intrinsically motivated agent. The algorithm gives better results in number of step and time to achieve goal comparing to agent with primitive action only.

2.2.9 Option discovery using spatio-temporal clustering

In this work, Lakshminarayanan et al. [2016], the authors propose a new method for state and action abstraction. The idea is to divide the state space into clusters; this partitioning is performed using the spectral clustering algorithm PCCA+. Based on an adjacency matrix of a graph; the Laplacian matrix is calculated, thus the algorithm computes the eigenvalues of that matrix and choose the k best ones, a linear transfor-

mation is than performed around the k resulting vertices to find the final clusters of the graph.

Option discovery

The PCCA+ is applied to the adjacency matrix of the MDP graph to find the clusters of the state space, each cluster constructs an abstract state, states are belonging to different abstract states according to different membership values which are computed using a projection method, see Lakshminarayanan et al. [2016]. One option is constructed between each two adjacent abstract states to move from one abstract state to another, the option policy is learnt by following the positive difference in the membership value of state to the destination abstract state; this method is called "hill climbing". Note that a state s belongs to an abstract state S with a maximal membership value comparing to its membership values to other adjacent abstract states. The termination condition is given by the states belonging to the two abstract states with closest or equal membership values, for example a door between two rooms belong to the first and the second room equally. The algorithm is adapted for online agent by sampling trajectories, and performing option discovery using PCCA+ algorithm, the resulting options are used in SMDP Q learning for sampling new trajectories, and the process is repeated until convergence.

Results

The algorithm is tested on the 3-room maze, the agent succeed to discover the three abstract states, and the correct options, the doorway as bottleneck correspond to the termination condition, and the policies learned were very functional for navigating agent from one room to another.

2.3 Discussion

In this section, we will review all methods mentioned above from another point of view, that of critics. We have set some criteria for a relevant comparison, starting with the definition of the sub-goals, being very important elements in the definition of options, many studies have shown that the best sub-goals are those that correspond to the bottlenecks of the environment, giving as a result the necessary and sufficient sub-goals for a given problem. The second criterion is that of the definition of the sets of initial states

for options, the more these sets are limited, the faster the execution of the options, at the same time the union of these sets must cover all the state space of the MDP. Another interesting criterion is the possibility of re-use of options, a hierarchical reinforcement learning agent can pass a fast traverse if he does not have to discover options for each new task, he can be a winner only if he has all the options at his disposal for high-level learning, so we examine if the options discovered by each algorithm are reusable for other tasks in the same environment. Autonomy is also an important criterion, a learning agent who does not need the intervention of a human to set a few parameters is very practical in cases where the man can not be present, thus the agent can adapt with its environment.

2.3.1 Critical study

In this sub-section, we criticize each method separately, and a comparison according to the criteria already cited is presented in table 1.

methode	sub-goals	initiation states	reuse	autonomy
BHNN	+	not defined	+/-	+/-
HEXQ	+	limited	+	+
PAC-inspired	+	limited	+	+
Laplacian	+	not limited	+	+
Betweenness	+	not limited	-	+
Community detection	+	limited	+	+
Information transition	+	not limited	+	+
Intrinsically motivated	+	limited	+	+
PCCA+	+	limited	+	+

Table 2.1: Comparison table of option discovery methods

Budgeted Hierarhcical Neural Network

The option that define a passage from one regular situation to another, like from one room to another in room maze domain are reused, but when the agent uses a new initial position, he must learn how to go to the door or to the goal using a high-level observation, because there is no complete definition of the initiation set for options. In the experiments on maze domain, the option model requires the information on doors, note that the doors position is the most important information in that domain because

it represent bottleneck regions, see Botvinick and Weinstein [2014], this information was given as an observation y_t and not automatically discovered.

HEXQ algorithm

There is one problem and it is cited in the article, is that HEXQ cannot be applied on all types of problems, like multi-room maze in which the discovery of exit transitions is ambiguous and it is the most important indicator for the HEXQ approach.

PAC-inspired option discovery

In this algorithm, the greedy approach used in discovering option is very useful to avoid the exponential complexity problem. However, the PAC inspired option discovery needs as inputs a set op MDPs sufficiently representative. In the experimentations, the algorithm needed 40 MDPs with a predefined reward function for discovering options in 4-rooms maze, which is very greedy in computing time.

Laplacian option discovery

This algorithm uses mathematical techniques as indicated by its name, which make it very powerful and simple at the same time. Note that the method is more exploratory than exploitative.

Skill characterization based on betweenness

In this approach, the use of the reward function of the MDP in the option discovery process makes that the options created are not task independents, which reduce the possibility of reusing these options, thus, the initiation states are not limited in the case of small MDPs, and are formed by all states of the environment, on the other hand, in the case of complex problem, the set of initiation states are formed by states with little distances from the terminal states, these sets are not necessarily optimal.

Constructing option through incremental community detection

This graphical approach is very powerful and gave a satisfying results in identifying options, the identification of sub goals is very relevant, and so for the initiation sets, but the method needs a full exploration of the environment for generating the transition graph, which is impossible in complex problems, so the authors use an online learning with incremental community detection, that is a power solution too.

Grounding subgoals in information transitions

This algorithm executed on several MDPs with different goals gives the possibility of reusing the options discovered for future tasks, another advantage of this method is that the agent doesn't require any prior world knowledge. The disadvantage is that the algorithm try to discover too many goals before being applicable on real task, and the option discovered perform an action abstraction, but there is no state abstraction in the sense that all options have the same initiation set, that is all the states of the domain.

Intrinsically motivated hierarchical skill learning

This algorithm based on FMDP representation use a DBN for each action executed and a CPT for each variable in the DBN, seem to have a very complex representation with a high spatial complexity, thus a probability distribution and an entropy are computed for each leaf of a CPT at each step, with the BIC score at each refinement. These calculations are necessary to have the ability to decide of the controllability of a variable, which makes this system very ingenious but very complex at the same time. Another limitation of this work cited in the article is the number of steps fixed manually to decide if an option succeeds or fails.

Option discovery using spatio-temporal clustering PCCA+

This graphical and mathematical approach has been successful in identifying sub goals and limiting the state space for options in optimal strategy and so for the policies learning. As many graphical approach, this method needs a full exploration of the environment for the construction of the adjacency matrix, which is not always possible with large and complex problem, so as solution , the authors proposed an online method with an incremental learning, that is the actual solution proposed by several research.

2.4 Conclusion

The goal of this study is to find the means to be able to design an option discovery method that is optimal in terms of re-usability, autonomy, and efficiency in the discovery of termination states and initiation states, without forgetting the main purpose of the options, which is to remedy the dimensionality problem.

From this study we concluded that the most efficient methods in terms of temporal complexity are the graphical methods like in Machado et al. [2017], Lakshminarayanan

et al. [2016], Hengst [2002], Şimşek and Barto [2009], X. Xu and Li [2018], these methods are autonomous, and efficiently discover the initiation states and termination states, and their options are reusable. Among these methods, the first three uses an adjacency matrix to present the states and transitions of the problem, this requires a full exploration of the environment, which is very time consuming when the problem to study is quite large, however the incremental methods build their environment as they discover the options, X. Xu and Li [2018] which is very profitable even with the non-graphical methods, Vigorito and Barto [2010], van Dijk and Polani [2011].

However the graphical methods are mainly classified as two phases. Methods that use intrinsic motivation demonstrate high efficiency and intelligence in the discovery process, and methods using information theory are strong enough at the same time very complicating, while connectionist methods as in Leon and Denoyer [2018] need a more in-depth study to be able to discover more reusable options.

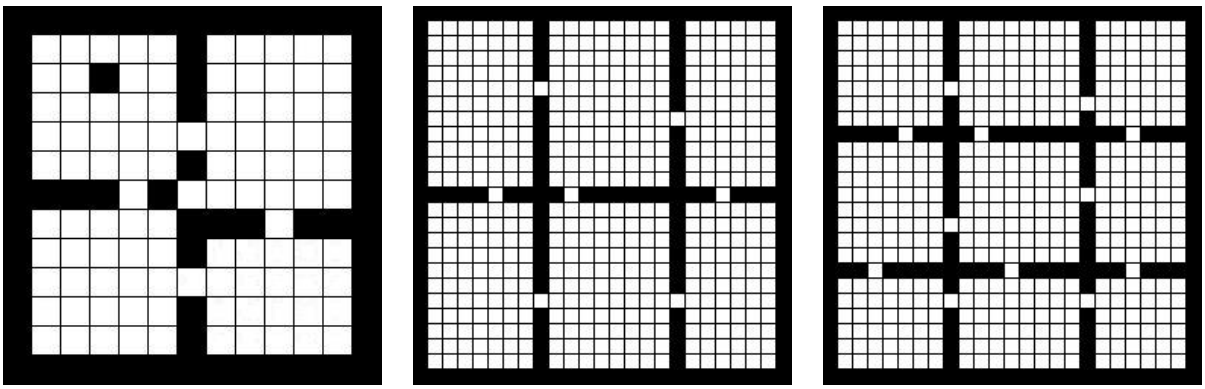
Subsequently, we hope to be able to design methods which make a compromise between graphical and incremental methods, and which use the principle of intrinsic motivation and may well be a connectionist methods; by trying the applicability to heterogeneous domains

3 FAOD: Fast Automatic Option Discovery in Hierarchical Reinforcement Learning

The automatic discovery of options has become a real challenge for research in hierarchical reinforcement learning and the new proposed approaches are very greedy in learning time or space. Thus we opt for a faster and less consuming approach. In this chapter we adopted a new perspective for the problem of automatic discovery of options in HRL assumed at one depth level. thus, we propose an automatic option discovery method for hierarchical reinforcement learning, that we call FAOD (Fast Automatic Option Discovery). We take inspiration from robot learning methods of Tani and Nolfi [1999] to categorize the sensorimotor flow during navigation. Here, the agent moves along the walls to discover the rooms' contour, closed spaces, doors and bottleneck regions to define initiation sets and termination states for options. The learning method consists in RL of the options by temporal-difference learning with actor-critic architecture, Barto et al. [1983], before moving on to the top level learning of the hierarchy.

3.1 Fast Automatic Option Discovery

Our approach for option discovery starts with a first exploration phase of the state space in order to discover bottleneck regions. But before we should discover the closed spaces like rooms. Thus the bottleneck regions will correspond to states where one of the actions lead to the discovery of a new state in a new room. The bottleneck regions in multi-room problems correspond to the doors, and define the subgoals (i.e., terminate states) of options. This description corresponds to the maze problem whose type 2x2 rooms was initially described in Sutton et al. [1999], then other types (2x3, 3x3, ...) appeared, as illustrated in the figure 3.1. Maze type problems are the most used RL problems to test and validate the approaches and solutions proposed in the field of RL



(a) (b) (c)

Figure 3.1: Maze examples (a: 2x2, b: 2x3, c: 3x3 rooms problem) which are gridworld environments with stochastic cell-to-cell actions.

for non-continuous problems.

Our choice for the discovery of the bottleneck regions is based mainly on the results of Solway and colleagues, Solway et al. [2014] whose experiments have shown that the best options are those that end on the bottleneck regions. Simsek and Barto, Şimşek and Barto [2009] calculated values of betweenness for each state; the maxima of betweenness represented the sub-goals, and gave the door states in the room' task. Thus, the work of vanDijk and Polani, van Dijk and Polani [2011] had similar results, by calculating a RGI value, the RGI value decreases drastically in the corners and near the walls. As in Botvinick et al. [2009], the authors manually chose the bottleneck states that are the gates in the maze problem as sub-goals. Although, ten years before that, the authors in Sutton et al. [1999] introduced maze problems for hierarchical reinforcement learning for the first time, and built the options manually, so that the doors between the rooms form the sub-goals or termination conditions.

The original idea of this work is to implement a procedure of exploration that mimics in a simplified way the process of automatic segmentation of the sensorimotor flow of robot navigating different rooms proposed by Tani and Nolfi [1999], where the authors proposed a set of recurrent neural network modules as experts that compete in a self-organized way to learn the internal model of the world. online. This work is based on their previous work, Nolfi and Tani [1999], where an agent has to detect changes happening in the environment. The changes detected are matched to a mixture of experts, each expert is a recurrent neural network to predict the next state of the environment, or sensorimotor flow. The novelty here consists in using this principle, without re-implementing the complex neural network machinery of the original algorithm, in order to define possible

terminal states of options as states in which their is a sudden change in the responding expert.

We tested and validated this work on maze problems introduced in Sutton et al. [1999], that is a grid environment composed of four, six or nine rooms. The cells of the grid correspond to the states of the MDP. The agent can move from one cell to another by taking one of the eight possible actions (north, south, east, west, northeast, northwest, southeast, southwest).

3.1.1 Room discovery

The room discovery phase consists of exploring the state space, Our agent starts with a randomly fixed starting position and looks for the top left corner to start exploration, to get to this corner it uses north and west actions only, until he gets stuck at the upper left corner, which will be the starting point of the rooms exploration.

For the rooms exploration, the agent moves along the walls and marks corners and doors as moments of salient and sudden changes in the sensorimotor flow (mimicking a change of expert in Tani and Nolfi [1999]) until arriving to the room start state. The agent has a wall tracking procedure and as soon as a salient change in its sensorimotor flow occurs, it considers it an important event that deserves to be marked. In this procedure, the robot simply follows the walls and detects the salient changes in its sensorimotor flow: for example the angles that force it to turn, the openings (doors) for which the robot no longer detects a wall neither at its left nor at its right are bottleneck states, Solway et al. [2014] considered later as sub-goals of the options.

Once the agent has returned to its starting position and has memorized a first set of marked states, the agent starts a second step where it explores different actions from each of the marked states. Because two of the four possible actions (North, South, East, West) lead to no displacement when performed in states labeled as corners, while the other two actions produce the same sequence of states experienced during the wall following process, these states are not considered as relevant sub-goals. In contrast, trying different actions in states labeled as doors lead to a room change, which corresponds to different states than those experienced during wall following. They can thus be stored in the set of possible sub-goals, to be later considered as terminal states for the options to learn.

For simplicity, as all work in the HRL field, we consider here that the agent can perfectly discriminate every state of the environment. This is simplistic but still a reasonable assumption since one can consider that different features of each room such as carpets of different colors, paintings or decorations on the walls, and different furnitures,

Algorithm 11 Rooms and subgoals discovery.

Initialize startState sS, minDuration $minD$, $iter = 0$.
list of Marked States $lMS = \{sS\}$
while $(iter < minD)$ and $(!isempty(lMS))$ **do**
  STEP1: Move along walls until back to sS
  if salient change in the direction of movement **then**
    mark state as corner and add to lMS
  end if
  if salient change in the sensors detecting a break in the followed wall **then**
    mark state as door and add to lMS
  end if
  STEP2: Search for new neighbor states
  for each state s in lMS **do**
    Remove s from lMS
    Test all actions from state s
    if one action leads to a new state s' **then**
      mark s as sub-goal
      $sS = s'$
      GOTO STEP1
    end if
  end for
  $iter++$
end while

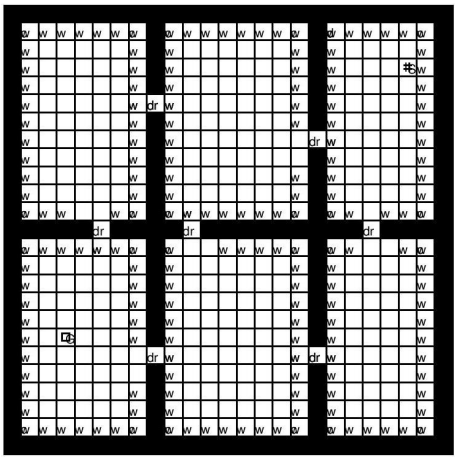


Figure 3.2: The maze (2x3 rooms) problem. Displayed state labels were obtained by the room discovery algorithm. All rooms are discovered with their doors marked (dr). The states next to the walls are marked (w). The start state is indicated by #, and the goal state is indicated by □, and are chosen interactively by the user.

enable the agent to discriminate different rooms.

Then the agent restarts the wall following process in each new room, starting from the door that gave access to it. This process is recursive until the wall following process has been performed from each successor state of a subgoal, and until no subgoal is added to the present set. The result of the room discovery algorithm on a 3*2 maze is shown in the figure 3.2. We fixe an arbitrary minimal duration of exploration so that at the end of this process the agent has visited all the rooms. The pseudo code of the room discovery procedure is given in algorithm 11.

Wall tracking procedure

In the wall tracking procedure, the agent walks next to the wall trying to break the wall with each step, this means that before advancing straight, the agent tries to go in the direction of the side wall. If he cannot pass, there is no change in his sensorimotor flow, and he must continue to walk straight. If he manages to walk in the direction of the wall, this marks a change in the sensorimotor flow, and implies the possibility of finding himself in a bottleneck or door if this passage leads to states or space that has not been explored before, and it must be a narrow passage in the sense that the agent cannot move in the old direction of wall tracking or its reverse direction.

In the example of the image (a) of figure 3.3, the agent is in position x, he walks next to the wall executing the action "west", at each step he tries to execute the action

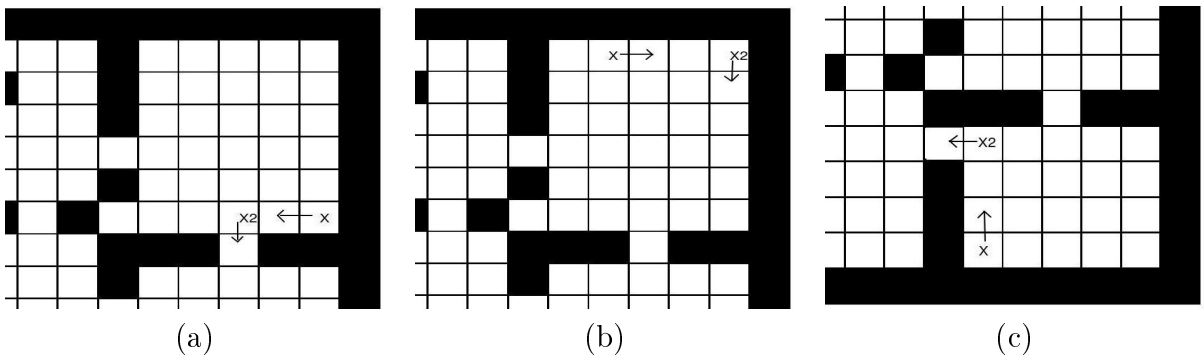


Figure 3.3: The image (a) the agent detects a door in his sensorimotor flow; The image (b) illustrates a corner state detection ; The image (c) illustrates the situation of detecting two sensorimotor changes at the same time, a door and a corner.

"south", which does not give any change, so it continues with the action "west" until he reaches position $x2$, at this state, trying to execute the south action, the agent does not feel the usual blocking, on the contrary he manages to pass, which marks a significant change in his sensorimotor flow. So it is possible that he is at the door position, at this point he can't move "west" or "east", he can only continue with the action "south" where he will end up in a state that he hasn't explored before, implying that the passage was indeed a door and the new space is part of a new room to explore.

The second type of important change in the sensorimotor flow is when the agent while walking along the wall finds itself in a blocking situation, where it can neither continue straight nor go in the direction of the wall. In this case, the agent notices that he is in a corner of the room and he has to turn in the opposite direction of the wall to start walking straight again next to the new wall.

As shown in the figure 3.3.(b) an agent in state x continues straight, taking the action "east", trying the action "north" at each step, but without result. When the agent is in position $x2$ neither of these two actions serves to move, and he is blocked, which is a significant change in the sensorimotor flow. In this case the $x2$ state is marked corner and the agent chooses the action "south" to be able to track the new opposite wall.

A third situation is possible, when the agent detects two changes in its sensorimotor flow at the same time. In this situation, the agent does not manage to continue straight near the wall, on the other hand he can move by pushing the wall, which implies that it is a door which is in a corner of the room as shown figure 3.3.(c), in this case the cell is marked corner and door.

3.1.2 Option discovery

As defined before, an option $o = \langle I, \pi, \beta \rangle$ consists of an initiation set I, a termination condition β, and an option policy π. In this work, the option initiation sets, and their termination conditions are deduced from the results of the room discovery procedure. Although we have explored only the contours of the rooms, their internal cells are easily calculated from these contours, this saves us a lot of exploration time, and this is where the fast feature of FAOD comes from, instead of a random exploration, we did an organized exploration that just follows the walls and infers the rest of cells.

Subsequently, the room cells will correspond to the initiation sets of the options and the doors or bottlenecks will correspond to the sub-goals, i.e. the termination conditions, following the strategy used in Sutton et al. [1999] and in Botvinick et al. [2009] for the manual construction of options where an option is constructed for each door in a room, such that the cells of the room form the initiation set and the door corresponds to the termination condition. So if a room contains two doors this implies that there will be two options in that room with two nearly identical initiation sets and two different termination conditions. A door which belongs to a room and which does not form its terminal state, will be part of its initiation set as illustrated in the figure 3.4.

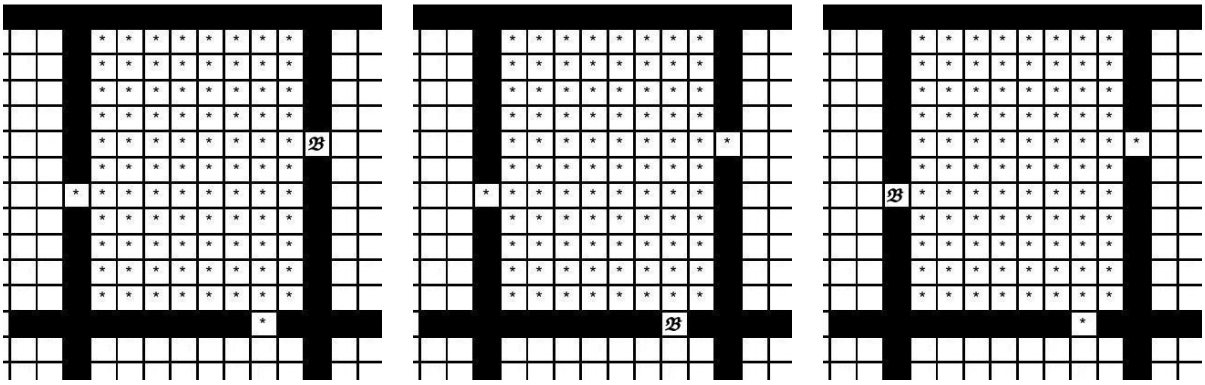


Figure 3.4: The room with three doors makes three options shown in three pictures, each option has a separate door for its sub-goal marked ($\mathfrak{B}$), and the states marked with (*) form the initiation set.

After construction of the apparent options, we add an option in each room such that the initiation set is formed of the cells of the room including the doors, and the termination condition corresponds to the global goal of the problem or the MDP. We must add this option in each room because we do not know in which room the global

goal cell is located, from where we must search in all the rooms in the next phase which is the learning phase.

3.1.3 Learning option policies

Each option whose initiation set and termination function are already discovered is treated as a separate RL problem to learn its policy, thus the initiation set of the option represents the set of states of the RL problem or the MDP's state set. The termination function that defines the sub-goal of the option will define the goal of this RL problem. The transition function which specifies the probability of transition from one state to another after execution of a chosen action will be the same as the original MDP considering the set of restricted states of the option. An option-specific pseudo-reward function is defined separately from the global MDP reward function.

For policy learning we have chosen temporal-difference learning in an actor-critic model that we had already introduced in chapter 2. In the actor-critic implementation, Sutton and Barto [2018], Botvinick et al. [2009], the agent comprises two parts, the actor part and the critic part, as présented in the figure 3.5. The actor chooses the actions to be executed in each state according to the option policy being learned π_o based on the weight of each action in each state. The purpose of the critic is to compute the value function V for each state, the value function indicates an estimate of the cumulative long-term reward expected after visiting that state.

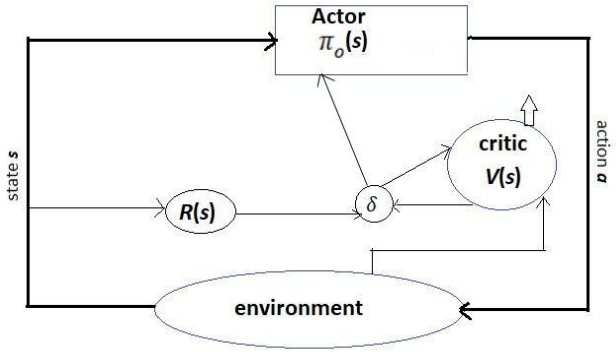


Figure 3.5: An actor-critic architecture. At each time step, the agent according to its actor part chooses an action a to end up in a state s and receives a reward $R(s)$. the critic will be recalculated from the calculation of the prediction error δ with the temporal difference method.

Learning by actor-critic methods is online and incremental learning because the policy to be learned is used at the same time as it is improved. The learning of the policy

is done through a succession of episodes, at the beginning of each episode a random starting position is chosen, when the agent reaches the sub-goal this marks the end of the episode and the actions that guided the agent towards the sub-goal are reinforced by the pseudo-reward.

At the beginning of learning, the actor materialized by the weight W is initialized for each action in each state of the current option, and the critic defined by the value V is initialized for each state s. At each time step the probability of choosing an action a is given by the function softmax defined as follows:

$$P(a) = \frac{e^{W(s_t,a)/\tau}}{\sum_{a' \in A} e^{W(s_t,a')/\tau}}, \tag{3.1}$$

where $W(s_t, a)$ is the weight of the action a in the current state s_t, and τ is the temperature parameter.

After the execution of the action a the agent finds itself in the state s_{t+1} and receives the pseudo-reward r_t, so it uses the difference temporal method to compute the error δ on the prediction of the cumulative reward expected at state s_t as indicated by the following formula :

$$\delta = r_t + \gamma V(s_{t+1}) - V(s_t), \tag{3.2}$$

where γ is the discount factor.

The temporal difference error is used to update the value function of the state s_t and the weight of the action a chosen in that state, as indicated by the following equations:

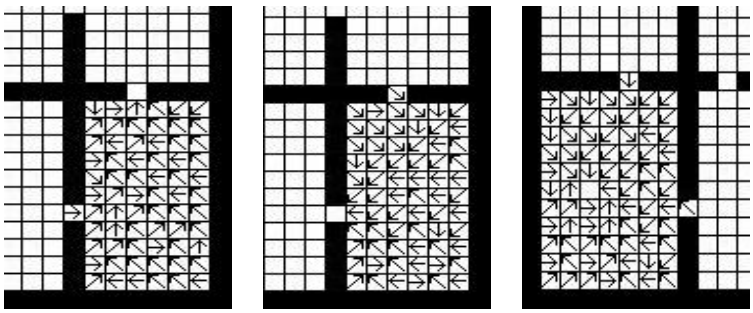
$$V(s_t) \leftarrow V(s_t) + \alpha_V \delta \tag{3.3}$$

$$W(s_t, a) \leftarrow W(s_t, a) + \alpha_W \delta, \tag{3.4}$$

where $\alpha_V \in]0, 1[$ and $\alpha_W \in]0, 1[$ are the learning rates.

This process continues until the sub-goal of the option is reached, then a new episode begins with a new starting state s_0 and the timer reset to $t_0 = 0$.

Once the learning is complete, the policy of the option is defined by the actions having a maximum weight in each of the states of the initiation set. Three examples of option policies are shown in the figure 3.6



(a) (b) (c)

Figure 3.6: Option learning results in the 2x3 rooms maze. After the rooms discovery, 15 options are obtained after learning. One option for each sub-goal of each room, and one option for the room that contains the final goal state. a) The first option policy, b) The second option policy, c) The 13th option policy that leads to the goal state.

3.1.4 Hiearchical learning

In hierarchical learning, we treated options in the same way we treated actions in option policy learning. We created options following the option framework Sutton et al. [1999] in which the notion of option was defined for the first time, and contrary to what was decided in Botvinick et al. [2009], options in our work are not interruptible, so once an option is chosen and launched in execution, we cannot choose another option until after that first one reaches its termination condition. We used the same strategy as for option policy learning; that is learning over options by temporal difference with an actor-critic architecture.

A weight W_o is associated with each option o, once the actor chooses an option, the actions are chosen according to the policy of that option until termination, where the prediction error will be calculated by the difference between the value of the state with which the option ended and the value of the state with which the option started. The critique maintains the value function V_o for the option o. The prediction error is defined as follows

$$\delta = r + \gamma V_h\left(s_{t+1}\right) - V_h\left(s_{init}\right), \tag{3.5}$$

V_h is the high-level policy (policy over options) value function that selected the option that just ended at state s_{t+1} and started at state s_{init} , r is the high-level reward, γ is the discount factor, and t is the number of time-steps elapsed since the relevant option was selected.

At each extended step, an option o is selected via softmax function as below.

$$P(o) = \frac{e^{W_h(s,o)/\tau}}{\sum_{o' \in O} e^{W_h(s,o')/\tau}}, \tag{3.6}$$

where O is the set of available options for the actual state s, $W_h(s,o)$ is the weight specific to option o at state s, and $P(o)$ is the probability of selecting an option o at state s.

After choosing and executing option o, the value function and the strength are then calculated as follow.

$$V_h(s) \leftarrow V_h(s) + \alpha_V \delta \tag{3.7}$$

$$W_h(s,o) \leftarrow W_h(s,o) + \alpha_W \delta \tag{3.8}$$

This cycle shown in the figure 3.7 continues until the overall goal is achieved.

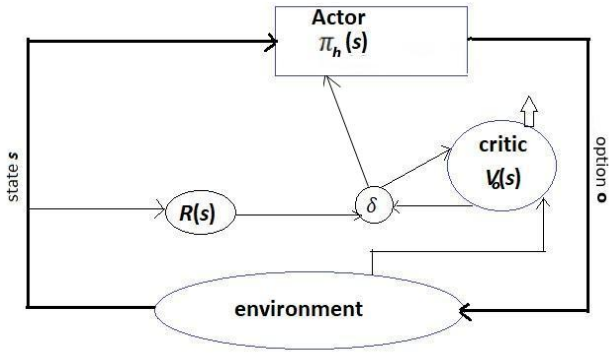


Figure 3.7: A top level actor-critic architecture. At each extended time step, the agent according to its actor part chooses an option o to end up in a state s and receives a reward $R(s)$. the critic will be recalculated from the calculation of the prediction error δ with the temporal difference method.

3.2 Experiments and Results

We have tested our method on multi-rooms maze problems, with maze ($2x2$ rooms, $2x3$ rooms and $3x3$ rooms) (Fig.3.1). For each experience, the user chooses interactively a start state and a goal state on maze image. The agent begins from the start state and then starts its exploration phase (Algorithm 11) to discover the rooms, the sub-goals and to define options. After that, options are learned one by one and a policy is calculated for each option. At the last step, an actor critic learning is applied at the top level,

where the agent chooses options instead of simple actions to reach the goal from the start state already defined. Our room discovery strategy consists of walking next to the walls by going through all the rooms, this implies that our agent must go around the perimeter of each room until returning to the starting square, hence the complexity of the room discovery algorithm is of $O(n * m)$, where n is the number of rooms in the maze, and m is the average number of states in the perimeter of one room. It is because of this low complexity that the algorithm is called fast.

The values of constant parameters γ , α and τ have been discussed in Sutton and Barto [1998] where they have proved that as γ approaches 1, the learning function takes future reward into account more strongly, so a value close to 0.8-0.9 gives a very good results, in our experiences we tried various values of γ, and after tests we chose $\gamma = 0.85$. Sutton and Barto Sutton and Barto [1998] have found that the learning rate α between 0.05 and 0.1 gives best convergence learning curve, so we used $\alpha = 0.1$, the temperature parameter τ have been tested in Sutton and Barto [1998], Botvinick et al. [2009] the best result were given by a value of $\tau = 10$ and that is the value that we adopted in our experiments.
In the experiments, we compared the results of our algorithm with the results of the flat learning algorithm, using the same actor-critical architecture. We performed 35 execution of our hierarchical learning and 35 flat learning on the same MDPs based on mazes of (2x3 and 3x3 rooms). The number of learned options is 300. We have made statistics on the number of iterations per episodes Fig.3.9 (a-b), ie the number of steps for the agent to reach his goal, and statistics on time elapsed in seconds per episode, these last tests are carried out on the same computer.

In the option policies learning, the agent is trained on an option for 150 iterations. The superfluous options are eliminated from the first iteration, when they reach a maximum number of steps without finding a sub-goal. This number changes proportionally with an estimation of the number of states of an option. There are also 150 iterations in the top level HRL learning.

For all maze problems used in the experimentations, the option discovery works perfectly. In the option learning phase, the agent then always eventually finds its path to the sub-goal. The top level HRL learning is similarly efficient, and the agent gets to find the goal in all the experimentations as presented in Fig. 3.8 (a),(b) and (c), and even when there is no passage between tow rooms as in Fig. 3.8 (d), here there is no connection between room 4 in the bottom left and room 5 in the bottom middle, our agent could find another path through room 1 at the top left of the maze.

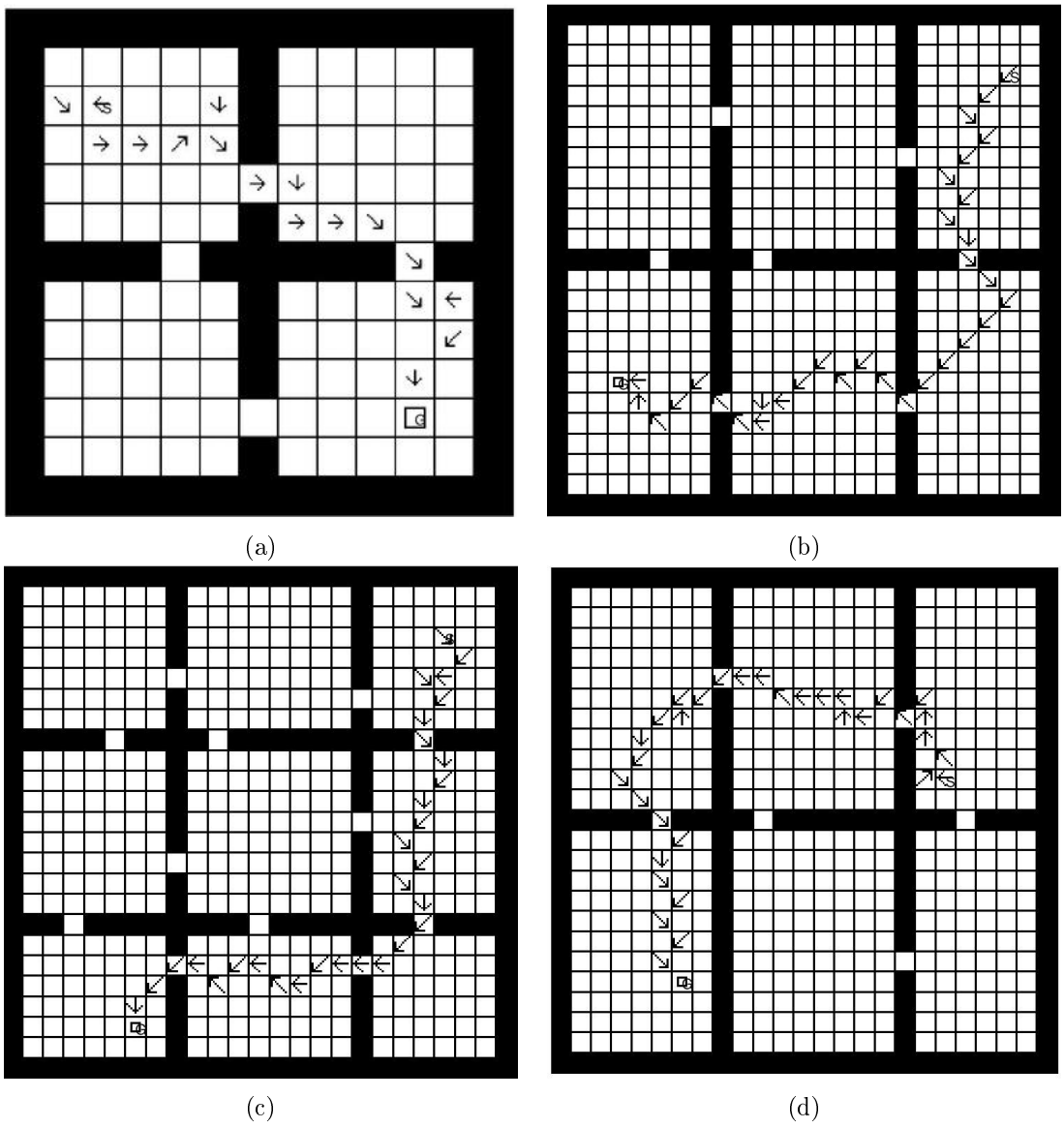


Figure 3.8: Results of the hierarchical reinforcement learning phase applied on mazes of 2x2 rooms (a), 2x3 rooms (b) and 3x3 rooms (c). In these simple examples the agent always eventually finds its path to the goal. In the maze (d) there is no door connection between room 4 (bottom left) and room 5 (bottom middle), in this case too, the agent finds its path to th goal by passing through room 1 (top left).

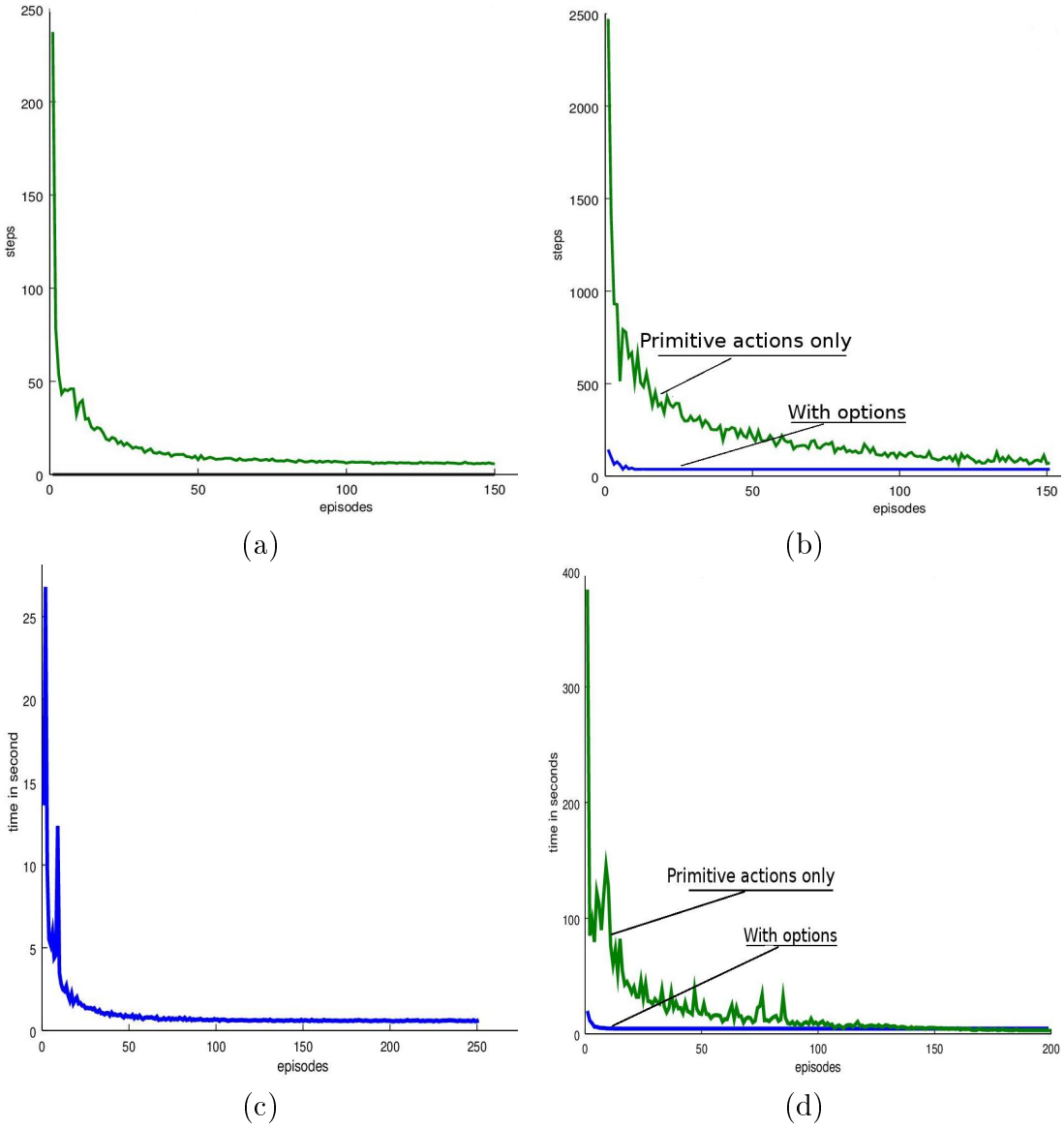
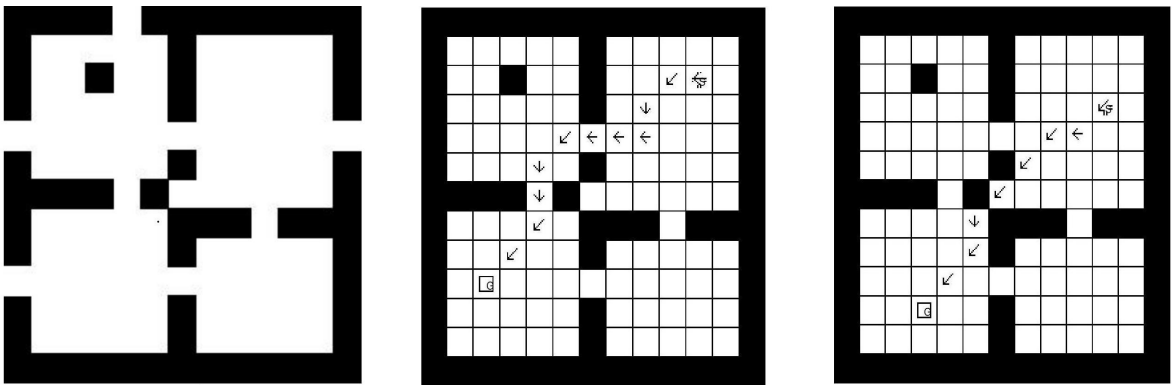


Figure 3.9: Learning curves for the results of execution. (a) curve illustrating the average number of steps to reach subgoal over episodes when learning option policy(146 runs of learning option on 3x2 rooms problem and 168 runs of learning option on 3x3 rooms problem). (b) The green curve illustrates the average number of primitive steps to goal using learning with only primitive actions. The blue curve illustrates the average number of primitive steps to goal using HRL with FAOD method (runs over 3x3 rooms and 2x3 rooms problems). (c) curve showing the average time in seconds to reach the sub-target of the learning options policies, (d) the green curve showing the average time to reach the target with a single primitive action, and the blue curve showing the average time taken by the HRL agent to reach the goal.



(a) (b) (c)

Figure 3.10: The image (a) is the 2x2 rooms problem with "windows" on the extern walls of rooms; The image (b) illustrates the result of executing FAOD with HRL on 2x2rooms problem when "shortcut" is opened up between the upper right and the lower left rooms; The image (c) illustrates the result of executing reinforcement learning with only primitive actions.

As illustrated in Fig.3.9, when the problem becomes complex (2x3 or 3x3 rooms), the number of primitive steps to reach the goal increases exponentially with a flat reinforcement learning using only primitive actions. In contrast, with FAOD this number has to decrease significantly since the problem is divided into sub-problems with sub-goals (a). Therefore the path to the final goal converges very quickly (b) compared to a flat learning. Curves (c) and (d) show statistics on the time taken by the agent to reach its sub-goal for option policy learning (curve c), or to arrive at the overall goal for hierarchical learning based on the FAOD algorithm (curve d in blue), and flat learning (curve d in green). these curves show the efficiency of our method. The time taken by the HRL agent is negligible compared to the time taken by the flat RL agent. The time taken in learning the options is always very small, and once completed, the options become reusable for future HRLs.

We next confronted the algorithm with the problem of rooms with windows; see Figure 3.10 This problem is quoted in Botvinick et al. [2009] , where windows constitute distractor states which induce salient changes in the agent's sensorimotor flow but nevertheless should not be considered as relevant sub-goals. FAOD did not mislead, and always gave 9 options for the problem of 4 rooms which is a good result, and the windows on the outer walls of the rooms are not considered sub-goals for new options. This success is due to the second step of the room discovery process, where the agent tries all possible actions from marked salient states (including windows), and where windows

do not lead to the discovery of new states in the context of the present navigation task. Finally, we confront the algorithm with the shortcut problem also tested in Botvinick et al. [2009] , where a new shorcut between rooms enables the agent to reach the goal state within a smaller number of steps without needing to go through some of the doors; Figure 3.10. In this case, FAOD does not consider the shortcut passage as a sub-goal, which leads to a final path to the goal that is unfortunately not the shortest path. Thus here FAOD suffers from the same limitation as in Botvinick et al. [2009] , where the prevalence of options prevents the algorithm from finding the simplest solution without options. Nevertheless, overall FAOD remains fast and efficient in maze problems with a large number of states explored here.

3.3 Conclusion

In this work, we have proposed a novel method for option discovery based on a pre-exploration procedure where the agent explores the environment by following the walls and detects any salient changes in its sensorimotor flow along this path. This method, inspired by a robot learning algorithm for online categorization sensorimotor flow Tani and Nolfi [1999] led to the discovery of corners, doors and windows, in simple multi-room navigation problems. The novelty here was to combine this sensorimotor categorization method with sub-goal discovery for HRL. The agent successfully learned to consider only doors as relevant sub-goals (i.e., terminal states for future options) because the successor state of these doors was a novel state, thus giving the status of bottlenecks or passages to the doors. The results show a total autonomy in the agent's behavior, who discovers its environment through a rapid strategy, discovers the options and learns them, and realizes a high-level reinforcement learning to reach the final goal.

However, this work suffers from a limitation because it applies mainly to simple and well-defined problems of navigation and space exploration. It is not necessarily straightforward to adapt our approach to other types of problems like the problem of Taxi or the problem of Playroom Barto et al. [2013] . In future work, it would be interesting to investigate whether a mixture of experts can remedy this problem and effectively extend our method.

4 Wayfinding Agent for Automatic Option Discovery in Hierarchical Reinforcement Learning

Discovering abstract actions or options for hierarchical reinforcement learning is challenging, and multiple approaches are proposed. In this chpater, we present our second new method for automatic option discovery, where our learning agent uses his sense of direction to discover the shortest paths and shortcuts after an exploration without resorting to the algorithms of the graph theory, since we use an incremental graphical representation without requiring an adjacency matrix. Our agent uses intrinsic motivation for a less random and more exploratory exploration of the environment. Shortest paths discovered subsequently serve to discover the termination conditions and the initiation states of the options. For the learning of options policies, the agent uses his experience of exploration as well as learning by temporal difference including an intrinsic motivation strategy. We tested our approach on different maze problems and on the tic-tac-toe game and the results were better than those of flat reinforcement learning and other methods in general and special cases.

4.1 An agent with wayfinding sense

The sense of direction or wayfinding is the ability of a human or an animal to find its way. This involves the ability to choose a route leading to the desired destination, as well as following the route, and verifying that the taken route leads to the desired destination. Some animals are sensitive to the Earth's magnetic field which allows them to find their way in a space they have never explored. On the other hand, humans have historically used visual landmarks including the sun and the moon.

For our agent, in order to acquire his sense of direction, he must begin by making an exploration, which consists of random paths, memorizing each time the starting point

and the arrival point, then he have to try to take the shortest path between these two ends which is the path in a straight line, or as straight as possible.

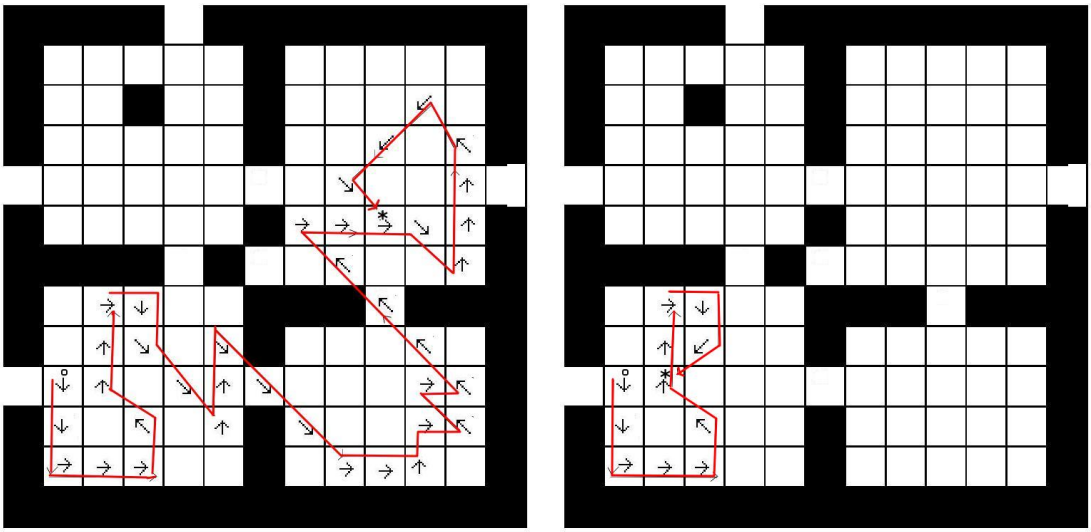
4.1.1 Exploration with intrinsic motivation

In the first phase, the agent explores his state space and tries to discover his environment. This exploration is not totally random, it is guided by the intrinsic motivation, Vigorito and Barto [2010], of the agent to discover its environment.

Psychologists refer a behavior of human or animal to be intrinsically motivated when it is inherently enjoyable, Barto et al. [2004], Singh et al. [2004]. So, humans or animals engage for activities as exploration, play, and other behaviors driven by curiosity, for their own sakes, without expecting an external explicit reward. Intrinsically motivated behavior is essential to accumulate knowledge and competences for solving future problems.

We have applied intrinsic motivation in the exploration phase, where the agent wants to discover new spaces instead of turning in already known spaces. The exploration itself is modeled as a reinforcement learning problem with an intrinsic motivation reward preventing the agent from going back. The exploration takes place in episodes, in each episode the agent starts from a random position, and begins to walk along a fairly long path. The intrinsic reward is zero each time the agent is in a previously unexplored position, and negative if it is in an already explored state. We need to make sure that the paths are long enough to have more possibilities to pass through narrow passages or bottlenecks, which will represent sub-goals for options in a later phase. For this we have two cases: -a. a path can reach the previously defined maximum length, or -b. it cannot reach this maximum length if randomly it is in a position that it has already visited in the same path, which marks the end of this path. In this case two situations arise; i. if the number of steps is large enough even if it is not the maximum, the path will be retained. In the other situation ii. the path does not contain enough steps and will therefore be rejected, because it has a low chance of passing through a bottleneck, as shown in the figure 4.1.

The intrinsic reward is '0' if the agent discovers a new space, and is negative and increases in value if it is in an already discovered space. The first time in a path (episode) that the agent is in an already discovered space, its reward or more appropriately its punishment is '-0.01', and each time it falls into the same error, this punishment increases in value. For example, if the agent returns to the same place for the fourth time, his reward will be '-0.04'.



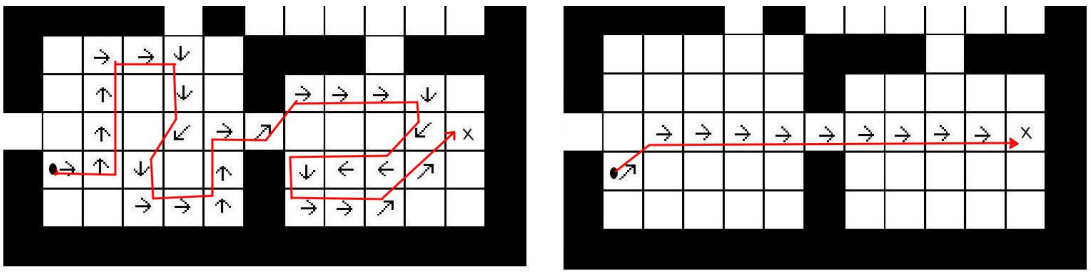
(a) (b)

Figure 4.1: Example of two paths that end when the agent returns to a position on the same path. (a) The path is quite long and will be retained in the list of paths, this path passes through two narrow passages. (b) The path is not long enough and will therefore be rejected.

4.1.2 An agent with wayfinding sense

During exploration, the agent performs random trajectories by applying the intrinsic reward to form paths that are not necessarily optimal. At each iteration, a path starts at a random position. The agent memorizes the start and end positions of the paths he has made, then tries to position himself by estimating the actual distance between the two ends of each path, this distance which is the length of the straight line between the two points is usually much smaller than the length of the initial random path as presented in figure 4.2. For example, for a path starting point (x_1, y_1), and an arrival point (x_2, y_2) the optimal distance estimated is given by the components (x_2-x_1, y_2-y_1). This optimal distance may not be real because of the possible presence of obstacles on the straight path, for this reason, the agent must travel this path, and check the distance at each step with respect to the arrival point, in according to the definition of the direction sense, section 4.1, where the agent must have the ability to choose a route that leads to the desired destination, and follow this route and ensure at all times that the route already taken leads to the goal.

Thus, the agent always chooses the direction of the straight line leading to the desti-



(a) (b)

Figure 4.2: (a) A random path. (b) The shortest path between the start point and the end point is the straightest path possible.

nation (x_2, y_2) and starts taking steps and registering them in the new path which must be the shortest or very close to the shortest in case of obstacles, and with each step, the agent evaluates the distance between the current position and the destination point as indicated in the algorithm 12.

Algorithm 12 The wayfinding strategy to find the shortest paths

```
Input: (x1, y2),(x2, y2) % the start and the end positions
path = {} ; % the shortest path initially empty
d1 = ||x2 − x1||;
d2 = ||y2 − y1||;
while (d1 > 0 && d2 > 0) do
  if (d1 > 0 && d2 > 0) then
    take a diagonal step towards (x2, y2)
  end if
  if (d1 > 0 && d2 == 0) then
    take a horizontal step towards (x2, y2).
  end if
  if (d1 == 0 && d2 > 0) then
    take a vertical step towards (x2, y2).
  end if
  Evaluate d1, d2
end while
```

```
procedure take step(direction)
try a step (direction)
if (no obstacle) then
  save the new position on path
else
  turn right
  try a step (direction)
  if (no obstacle) then
    save the new position on path
  else
    turn left
    try a step (direction)
    if (no obstacle) then
      save the new position on path
    end if
  end if
end if
```

From this strategy, the agent continuously ensures that it is in the correct path. In the event of an obstacle, the agent turns to the right to circumvent the obstacle, and tries to find a passage, if he does not succeed, he seeks to the left. So the agent tries to

get around the obstacle away from the original direction as little as possible, as if the agent is holding a compass and always manages to reach the end point in a minimal number of steps. Once the passage to divert the obstacle is found, the steps that led to this passage are also recorded in the path, and the procedure continues until reaching the destination.

This procedure is applied to all paths traveled by intrinsic motivation and recorded. At the end of this phase, we will obtain the shortest paths by the agent wayfinding sense. The agent then memorizes these best paths to take advantage of in the option discovery phase. Our approach takes much less time than the methods of graph theory in finding the shortest paths. The initial exploration may not be perfect in the sense that the agent is not obliged to visit all the states of its environment what is the case in most classical methods and which requires an adjacency matrix as in Lakshminarayanan et al. [2016], Machado et al. [2017], Şimşek and Barto [2009] . During the discovery of the shortest path the agent can always find himself in states he has never visited in the exploration phase.

4.1.3 Option discovery

Our option discovery method goes through three steps; i- discovery of terminal states of options, ii- discovery of initiation states, iii- learning of option policies. These three steps are completely automatic and require no human intervention. We will detail the first two steps in this subsection and leave the third for the next subsection.

Terminal states discovery

The terminal states of the options are the sub-goals of the small tasks, when you carry out a project consisting of several consecutive tasks, you cannot go from a first task to a second without having finished the first and reaching its goal, despite the fact that there can be a multitude of methods to achieve it. The idea is to exploit this characteristic of sub-goals, knowing that there can be several paths for a single goal, this implies that the goal is a common point between several paths. However, the search for sub-goals does not simply amount to looking for the common points between the shortest paths already discovered, but to looking for the most common or most visited points. Thus, for the first step, which is the discovery of the terminal states of options, we use the shortest paths recorded during the exploration phase. And, we perform a count of the states visited during the step of discovering the shortest paths, the most visited states of the

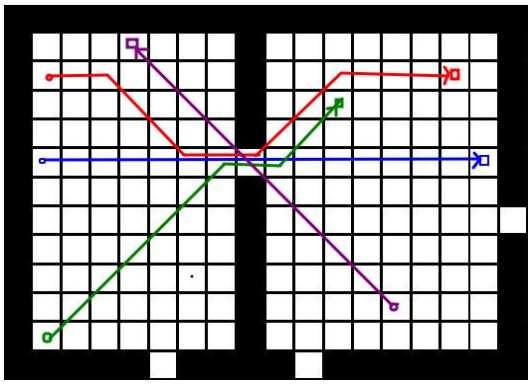


Figure 4.3: Example of terminal state discovery, the box of the door or the passage that connects the two rooms constitutes the state most visited by the paths, and therefore the local maximum of the states visited by these paths.

environment form obligatory passages or bottlenecks, they are states checking the local maximas of the repetitions of visits, and will be considered later as termination states for future options, as presented in figure 4.3. This technique has already been adopted in Şimşek and Barto [2009], in the calculation of betweeness local maxima, however our method is not based on paths taken from a graph or from graph theory but on paths taken from episodes of experiences.

The other states belonging to the shortest paths and which are not terminal states are necessarily initial states of the options.

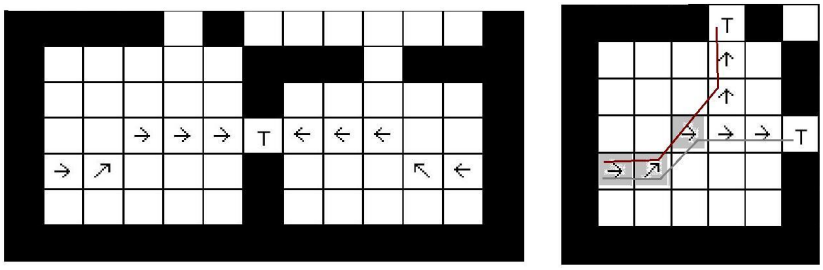
Initiation states discovery

In our approach the initial states are not common for all options, i.e. each option must have a limited space, that accelerates option policy learning and the learning of policy over option.

The states belonging to the shortest paths and which are not terminal states are necessarily states of initiation of the options, but they must be grouped according to the options to which they belong. At this level, we can deduce the initial states for options by bringing together the states of the paths that lead to the same goal, However, in this work we wanted to create optimal options as defined in Sutton et al. [1999], Botvinick et al. [2009], Solway et al. [2014]. For example, in a maze type problem, the best work organization is to associate an option for each door in each room. This implies that for a door that separates two rooms we will have two options that have the same terminal state, and two different initiation sets, each set corresponding to the space of one room.

As a result, a problem arises because two paths that lead to the same terminal state

are not necessarily part of the same option, see figure 4.4-a. On the other hand, if a room has two doors or more, it implies that the space of this room generates two or more options with similar sets of initiation states and different termination states, therefore the states belonging to a single path can belong to two or more options at the same time, as shown in figure 4.4-b.



(a) (b)

Figure 4.4: Conflicting cases in initial state set construction. (a) Two paths that leads to the same terminal state but not belonging to the same option. (b) The states in gray belong to two different destination paths, and to two different options at the same time

The solution consists in considering the paths that lead to a terminal state without passing through another, if a path passes through two or more terminal states, it will be divided into subpaths which each lead to a terminal state, as a result, we will have a list of optimal paths, each leading to a single termination state.

For the discovery of initiation states sets that delimit the space of options, our algorithm; (Algorithme 13) considers each path as an option before starting iterations. These options (path) are clustered into groups according to their termination states. In each group we look for the intersections between the initiation sets of the options two by two, i.e. between the states of paths, and each time we find an intersection we perform a union between them to have only one option. This process is repeated for each group until stabilization i.e. until no new intersection is identified. Therefore, the number of options will be reduced at the end of the iterations to the number defined by Sutton et al. [1999], Botvinick et al. [2009], Solway et al. [2014], and if we have two rooms connected by a passage (sub-goal) we will have an option for each room, because there is no intersection between the spaces of two rooms. After traversing paths, we end up with sets of states, each set corresponds to the initial states of an option.

At the end, options may be in intersection with others, for example, in maze-type problems, if two options have states in common this implies that they belong to the same room and lead to two different doors or terminal states. Therefore, if the set

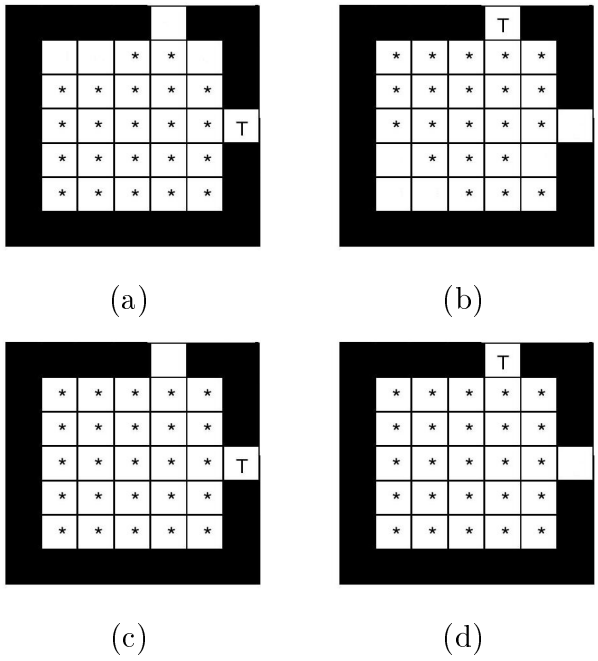


Figure 4.5: Example of complementarity in the initial states sets construction. (a, b) Two different options whose initial states belong to the same part of space, the sub-goal of option (a) is the easternmost state and the sub-goal of option (b) is the most northerly, both options suffer from an incomplete state set. Each option completes the other, option (a) supplemented becomes option (c) and option (b) becomes (d).

of optimal paths towards a sub-goal and which belong to a room have not succeeded in covering all the space of this room, this implies that the initiation state set of the corresponding option is not yet covered. To remedy this problem, the paths leading to another sub-goal and belonging to the same room can cover the missing space. Thus we use complementarity between options, i.e. we use each option to complement another that belongs to the same space as shown in figure 4.5.

The options discovered with their sub-goal are all independent of the global MDP goal, and are therefore reusable for other MDPs that share the same environment. Our goal is to find the global solution for an MDP, for that, we should add options that are global goal dependent. Thus we add options defined by a common termination state which is the goal of the global MDP, and sets of different initiation states, each of which corresponds to a different state space among the spaces already discovered. For example, for a room with two doors we had already discovered two options with the same space of the room and two different sub-goals, so we add an option for this room and the termination state will be the goal of the global MDP. Knowing that the overall goal can

Algorithm 13 The option discovery algorithm

```
Input: Ter % the set of termination states
O = {} ; % the set of options initially empty
nb = 0; % the number of options
for each x ∈ Ter do
  repeat
    for each path ph (not empty) going to x do
      for each path ph2 ≠ ph going to x do
        if ph ∩ ph2 is not empty then
          ph = ph ∪ ph2
          ph2 = {}
        end if
      end for
    end for
  until stabilisation
end for
for each path ph (not empty) going to x do
  if ph is not empty then
    nb++;
    creat a new option O_nb
    O_nb initiation set=all states of ph
    O_nb terminal state=x
  end if
end for
```

only be found in one option, the other useless options will be eliminated in the learning option policies phase.

4.1.4 Learning option policies

The agent has all the options with their initial states and the terminal states, it remains to determine the policy of each option or the strategy to follow, and this is where we apply reinforcement learning, here we use the temporal difference algorithm on an actor-critic architecture, Sutton and Barto [2018], Botvinick et al. [2009]. However our agent has already discovered the shortest paths passing through the sub-goals in a previous step, which implies that a large part of the work of learning option policies has been done, and proceed by a completely random learning will result in a great waste of time. To avoid this loss, we will exploit the results of the experiments of the exploration and discovery of optimal paths phase in order to lighten the learning process.

The idea of exploiting the past experience of an RL agent is not new, Kearns and Singh

[2002] in their E3 algorithm, proposed two policies in the same training, an exploration policy for unknown state spaces and an exploitation policy for known spaces.

For an ordinary RL, when an agent is in a state that quickly leads to a goal, its future cumulative rewards must be encouraging, this value can only be reached after several learning episodes. The purpose of using past experiences is to save the time lost by these episodes. For that, we give the agent a good reward if he is on a good path leading to his sub-goal. Thus we introduce the exploitation of optimal paths in the reward r(s,a) initialization step, such that, when the agent is in a state that belongs to an optimal path leading to the termination state of the option and chooses an action which follows this path, he will receive a positive reward, the reward is negative in the other cases except if the agent reaches his sub-goal. Actor-critic learning will follow the same steps described in option policy learning in our previous work FAOD, and we introduce the principle of intrinsic motivation to save even more time. Thus, learning for our agent endowed with a sense of direction can be summarized in the following steps:

1. Initialization of weights W(actors) and values V(critics) with 0,
   initialization of rewards $r(s,a) = +1$ if (s,a) belongs to an optimal path, otherwise $r(s,a) = -1$;

2. For each option OP discovered:

   a) For each learning episode:

      a Choose a random starting position s

      b While (termination state not reached and the number of steps $< nmax$)

      - Choose an action by softmax $P(a) = \frac{e^{W(s_t,a)/\tau}}{\sum_{a' \in A} e^{W(s_t,a')/\tau}}$
      - Execute the action a.
      - Observe the next state s' and the reward r.
      - Calculate the prediction error $\delta = r + \gamma V(s') - V(s_t)$.
      - Calculate actor and critic. $\begin{cases} V(s) \leftarrow V(s) + \alpha_V \delta \\ W(s,a) \leftarrow W(s,a) + \alpha_W \delta \end{cases}$
      - $s \leftarrow s'$

      c if (number of steps $== nmax$) then eliminate OP

Where $\alpha_V \in]0,1[$ and $\alpha_W \in]0,1[$ are the learning rates, and $nmax$ is the maximum number of iterations allowed in a learning episode, calculated based on the number of

states in the initiation set of each option. This number nmax makes it possible to eliminate the superfluous options which are the options that have been added to contain the global goal of the MDP, only one option among these really contains the global goal, for the others, the agent will search by following the steps but he will never find the goal, so the option will be eliminated.

The choice of actions by the softmax function and the updating of weights and values will follow the same strategy described in the FAOD method (see previous chapter). The reward received for each executed action defines the strategy of compromise between exploration and exploitation, at the same time as the intrinsic motivation already used in the exploration step, so that if the agent is in a state belonging to a optimal path and chooses the action that allows him to follow this path, the reward is (+1). On the other hand, if he is in a state that is not part of any optimal path towards the sub-goal, we proceed by exploration with a reward of (-1) which allows him to search again. Finally, so that the agent does not get stuck in a position and always looks for new states leading to the sub-goal, his reward is reduced by (0.5) each time he chooses an action allowing him to stay in the same state, as if trying to cross an impassable wall. After the execution of the algorithm, the policy of the option is given by the actions which check the maximum weight in each state of the option.

However learning will not be completely random, to benefit from the past experience, the agent uses the memorized paths during his discovery of the shortest paths. If the starting state of one learning episode is on a path to the terminal state, the agent takes this path and his reward is reinforced, otherwise the agent can end up in a state that is not part of any path going to the sub-goal, in this case he learns by applying intrinsic motivation on temporal difference (with actor-critic) learning, Vigorito and Barto [2010], Singh et al. [2004], Barto et al. [2004] ; for a better exploration until he finds a way to the goal or he finds the goal itself.

4.1.5 Hierarchical learning

Option policies already learned by the agent are task independent, so for each new task the agent can use the options that will always be available, and he will not need to explore and discover his environment each time. Hierarchical reinforcement learning is a reinforcement learning that uses abstract actions (options) instead of primitive actions. In this work, we use the same actor-critic architecture with the temporal difference algorithm, Sutton and Barto [2018], that at each training step chooses an option instead of choosing a primitive action. The options are not interruptible for maze problems but

they are for tic tac toe games as described in the experiments.

Like primitive actions in an RL, weights are associated with options in an HRL. At each extended step, the actor can choose an option, which will be executed by scrolling the policy of the option already learned, until termination if it is a maze problem, or until the The option is no longer available in the event of a non-deterministic problem (the case of tic tac toe games). A reward is then received by the agent and the prediction error is calculated. A positive error prediction indicates that the reward is better than expected, and a negative error prediction indicates that things are worse. This error is used to update the actor, i.e. the weight associated with the initiation state with the option, and the critic which is the value associated with this same state in the same way as we did in FAOD. The stages of the hierarchical learning of our agent with a sense of orientation are summarized as follows:

1. Initialization of weights W(actors) and values V(critics) with 0,
2. For each episode:
   - **a** Take the starting position $s = s_0$
   - **b** While (Goal state s_G not reached)
     - Choose an option by softmax function $P(o) = \frac{e^{W_h(s,o)/\tau}}{\sum_{o' \in O} e^{W_h(s,o')/\tau}}$
     - Execute the option o.
     - Observe the next state s' and the reward r.
     - Calculate the prediction error $\delta = r + \gamma V_h(s_{t+1}) - V_h(s_{init})$.
     - Calculate actor and critic. $\begin{cases} V_h(s) \leftarrow V_h(s) + \alpha_V \delta \\ W_h(s,o) \leftarrow W_h(s,o) + \alpha_W \delta \end{cases}$

Where $P(o)$ is the probability of selecting an option o at state s, O is the set of available options for the actual state s, and $W_h(s,o)$ is the weight specific to option o at state s.

V_h is the policy over options value function. The selected option started at state s_{init} and ended or stopped at state s_{t+1}, r is the problem depended reward, and t is the number of time-steps elapsed since the relevant option o was selected. $\alpha_V \in]0,1[$ and $\alpha_W \in]0,1[$ are the learning rates.

After the execution of the algorithm, the high level policy is given by options which check the maximum weight for each state encountered during hierarchical learning for the current problem.

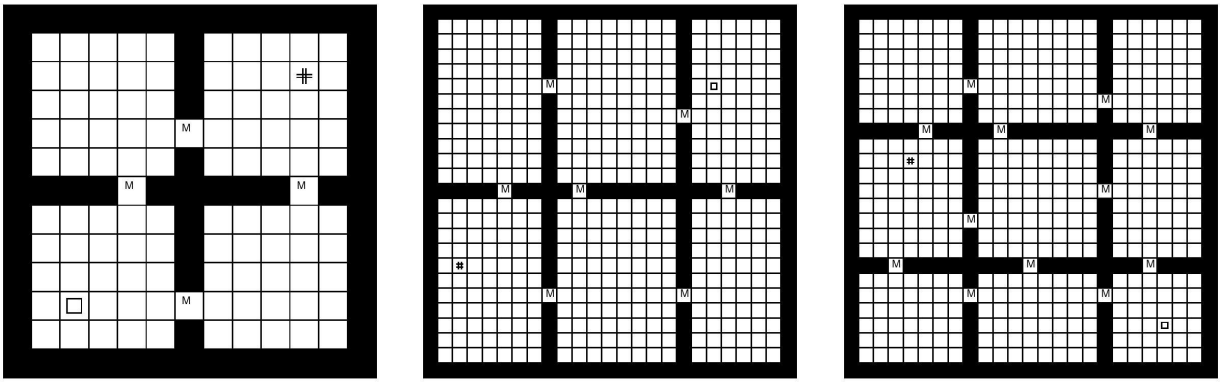
4.2 Experiments and results

To show the performance of our agent with a sense of direction in option discovering, we test it on two type of problems; multi-room maze problem and Tic Tac Toe game.

4.2.1 multi-room maze

For the first case we used the 4-room, 6-room, and 9-rooms grid-world, we performed 30 experiments to compare the results of our HRL method to the flat reinforcement learning. At each experience the start state and the goal state are fixed by the user interactively.

The algorithm starts by a partially random exploration of the environment as it is driven by an intrinsic motivation reward. Thus, the agent has to walk in random paths. The number and the length of these paths are calculated according to the estimated size of the environment. At each step of each path, the agent can choose one of the eight possible actions (north, south, east, west, north-east, north-west, south-east and south-west).



(a) (b) (c)

Figure 4.6: The sub-goals discovery : the sub-goals discovered by the agent with a sense of direction are marked by the letter M and they correspond to the doors which are the best subgoals made manually in 4-room maze (a), 6-room maze (b) and 9-room maze (c). At each experience the start state is marked by # and the goal state is marked by □

Next the agent memorizes the start and end positions of each path, and takes the shortest path between the two extremities, this shortest path is calculated geometrically, i.e. the path closest to the straight line. This is what we call our agent's sense of direction. All states of the shortest paths are then recorded, and the states that form the most repeated intersections between shortest paths form subgoals that the agent

must cross to reach its destination. The results of discovering subgoals were successfully exactly similar to manually made subgoal described in Sutton et al. [1999], Botvinick et al. [2009], as shown in figure 4.6.

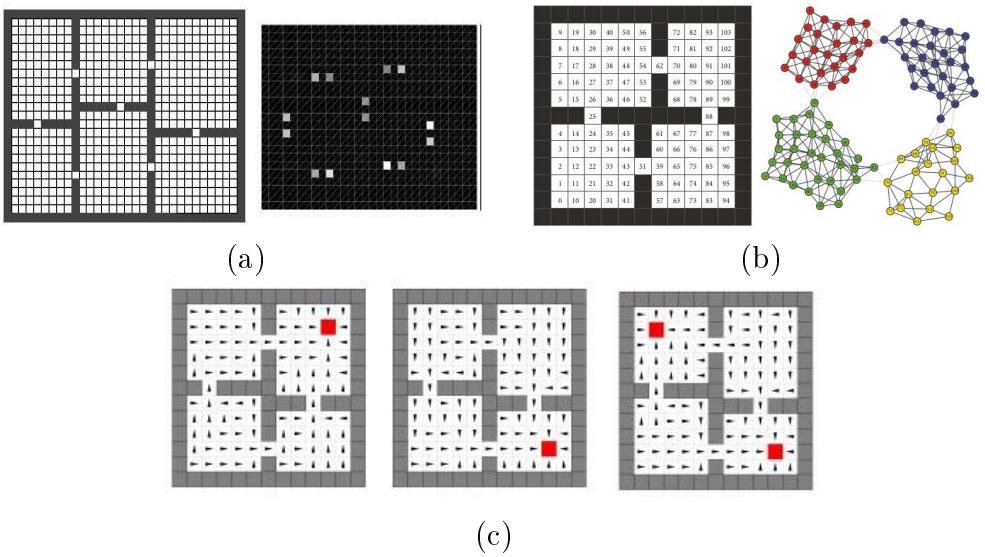


Figure 4.7: Sub-goal discovery in methods that use the adjacency matrix. (a) Sub-goal discovery in the approach of Simsek and colleagues. (b) Sub-goal discovery in the approach of Xu and colleagues. (c) Sub-goal discovery in the approach of Machado and colleagues.

From Figure 4.6 and Figure 4.7[1] we can make a comparison of the results of the option discovery quality between our method and other methods which proceed by dense exploration of the environment and which use an adjacency matrix. In figure 5-a, the results of the Şimşek and Barto [2009] approach gives as sub-goal the two boxes which border the door between two rooms which is slightly different from the ideal sub-goal defined in Sutton et al. [1999], while the definition of initiation sets is very different.

Figure 5-b shows the result of dividing the state space on the same type of problems, multi-room maze, by the methods of X. Xu and Li [2018]. This method divides the state space into communities, the figure indicates 4 communities connected with a node between each two communities, these nodes are the doors between the rooms and represent the sub-goals, which corresponds exactly to the sub-goals sought, only the method suffers from high temporal complexity.

Figure 5-c shows the results of the option and subgoals discovery for the method, of Machado et al. [2017] which performs a proto-value function calculation, although this method succeeds in creating different options, subgoals discovered are far from the

[1]These images were taken directly from their corresponding articles

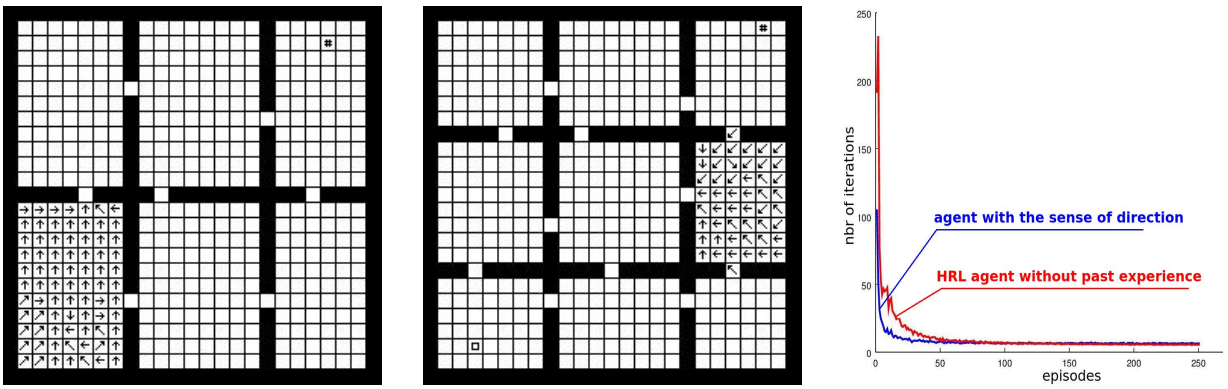
optimal subgoals defined by the research community, as well as the initiation sets are the same for all options, which makes the complexity of learning the policy of one option identical to the complexity of learning global RL policy.

From these results we conclude that we have succeeded in discovering the optimal sub-goals while consuming less time thanks to the senses of orientation of our agent.

The initiation sets of options are successfully constructed according to Algorithm 13, followed by the application of the principle of complementarity between the options gives good results, and the initiation sets are the same as those defined in Sutton et al. [1999]. We have managed to create an option for each room and door in the room, that is to say that for a room which contains two doors we will have two options whose initiation sets are almost the same and the sub-goals are different, unlike the methods of Simsek and Machado which did not succeed in finding the right initiation sets.

Then the options became ready for learning. We applied the actor-critic architecture with temporal difference learning, and with the use of the past experience i.e. the shortest paths discovered in the first phase. At each option learning episode, the agent checks his position, if it belongs to an already discovered shortest path leading to the sub-goal, he takes this path directly, this is how the agent uses his past experience. If the position does not belong to any shortest path, the agent performs actor-critic learning until arriving at the sub-goal or a shortest path.

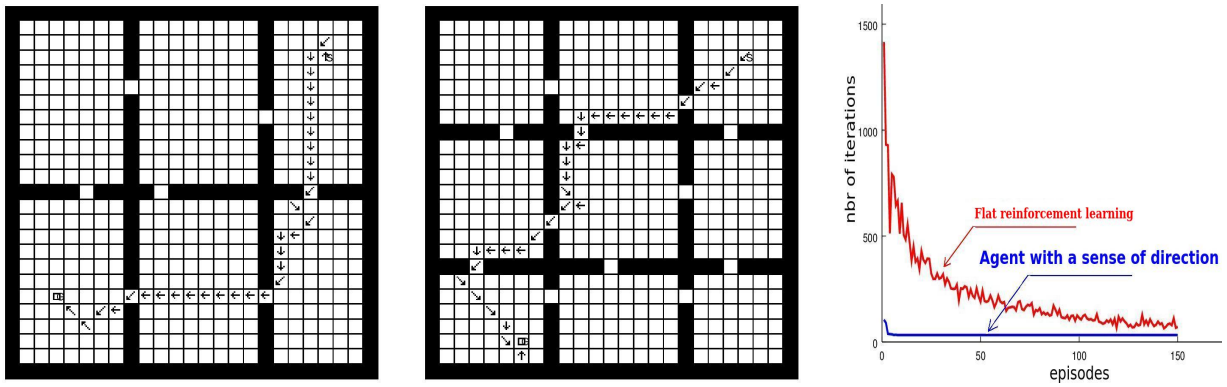
We have performed 166 experiences of option learning with 250 episodes for each option learning, and as result, the learning process converged rapidly compared to the same process of learning without past experience. The results of option learning are shown in figure 4.8.



(a) (b) (c)

Figure 4.8: Learning options policies. (a & b) Examples of two option policies in 6-room and 9-room maze. (c) Curve comparing between our wayfinding agent learning convergence, and an agent without past experience.

For the hierarchical top level reinforcement learning, we performed an actor-critic architecture learning with 150 episodes, and in all experiences the agent found his path to the goal using the options policies as abstract actions. The HRL converge rapidly

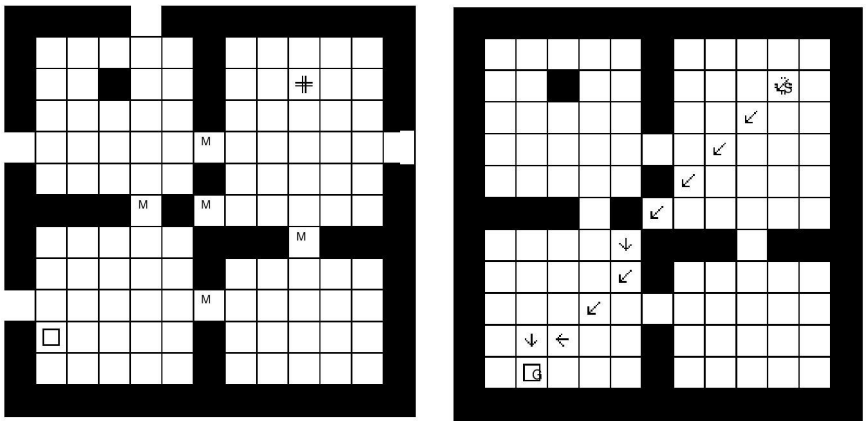


(a) (b) (c)

Figure 4.9: Learning top level policy. (a & b) Examples of final paths after hierarchical learning in 6-room and 9-room maze. (c) Curve comparing between our agent hierarchical learning convergence, and a flat reinforcement learning agent convergence.

as shown in the curve of the figure 4.9 where we compared HRL learning with the flat reinforcement learning which became very slow in complex problems (6-room and 9-room maze). We have carried out other tests on special cases of the problem. The first case is that of maze with windows on the external walls of the rooms, these windows should not be considered as sub-goals by learning systems, the second case is that where a shortcut is present in the maze, which is a window on the internal walls of the rooms,

thus in the best cases, shortcuts should be considered as subgoals, these tow problems were invoked in Botvinick et al. [2009]. For the first case, our agent was not distracted



(a) (b)

Figure 4.10: The experiments on special cases. (a) The windows on the exterior walls of the rooms were not considered as sub-goals. (b) The internal window was considered as a sub-goal and the agent succeeded in finding the shortcut.

by the presence of windows and his sens of direction did not mislead, and the windows on the external walls of the rooms were never considered as subgoals, on the other hand, the shortcut was considered as subgoal (see figure 4.10) which is a very satisfying result comparing to the manually formed options in Botvinick et al. [2009], and to our previous work FAOD Koudad [2021] where only flat reinforcement learning succeed to find the path passing by the shortcut.

4.2.2 Tic-tac-toe game

The Tic-tac-toe game, S. J. Russel [2010], Sutton and Barto [2018] is played by two players X and O on 3x3 grid, where at each step the player X have to mark 'x' on a free position he choose, then the turn of player O to mark 'o' in a free position too, the game ends in a maximum of 5 stages. The winner is the player that marks a row first, it can be a vertical, horizontal or diagonal row, if there is no winner at the 5th stage, the two players are tied.

For our experiments, our agent with a sense of direction played X and the opponent agent played O. At the first exploration our agent played randomly and memorized only the paths leading to a victory. The tasks are then decomposed, the agent has to reach the 3rd or the 4th state in the memorized paths, these states can be final steps if the agent win in 3 steps as shown in figure 4.11(a), or before final steps otherwise figure

4.11(b,c), these states formed terminal conditions for options, final steps were considered as terminal conditions too. Thus we had two types of options, phase one options, are the options that start at step 1 and end at step 3 or 4, and the options of phase two are the options that start from the terminate states of the options of Phase one, and end at the end of the game, usually single stage options. Because of the non deterministic character of the game, the number of the phase one options exceeded 500 cases.

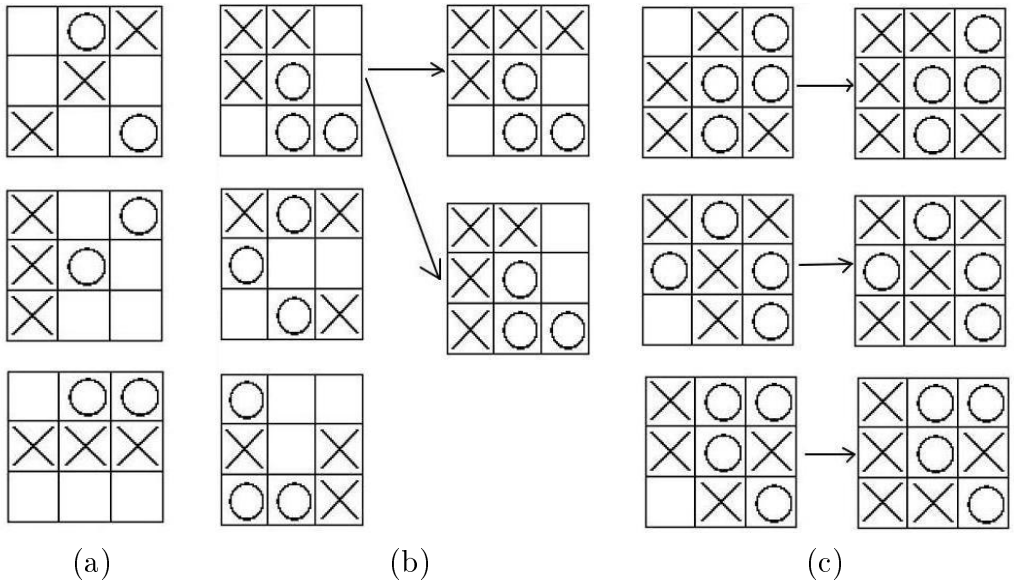


Figure 4.11: Examples of option terminal state in tic-tac-toe game. (a) Terminal state for option after 3 steps is terminal state for HRL at the same time. (b) Terminal state for the option after 3 steps and a second option is called to win. (c) Terminal state for the option after 4 steps and a second option is called to win.

Next, the agent learned the policies of theses options, we used the same actor-critic reinforcement learning architecture. The reward was fixed to +100 if our agent reach the terminal state of the option, -100 if he loses and $-(difference\,with\,subgoal)/10$ otherwise. For the top level reinforcement learning, we used the same architecture too, the reward was +1000 if our agent wins, -1000 if he loses, -10 if the game ended in a tie and -0.1 in intermediate stages.

Due to the non deterministic character of the game, the options were not in-interruptible, we could start with option o_i, and before reaching the terminal state, we jump to another option o_j for which we reach the terminal state. So high-level learning takes an option in the first phase, but it doesn't have to be the only one option. The first phase can end with the terminal condition of the first, second, third or fourth option. The agent takes only one option in the second phase.

We performed statistics on the results of option learning of the two phases, and on the top level hierarchical learning. Figure 4.12(a) shows that in the beginning of phase one learning, our agent X was on par with adversary agent O which is a random agent, and as the learning progresses, our agent manages to gain the upper hand, and learn more and more options and reach the final states of its options. In the second phase, indicated in figure 4.12(b), our agent succeeds quickly in mastering the situation in few epochs of learning, and gaining the upper hand and this is due to the small number of game steps in this phase. Regarding statistics on high-level hierarchical learning, we counted the number of times our agent won, the number of times he lost and the number of ties in each 100 learning episodes knowing that the number of episodes was 50000, and we perform the same processing to the results of learning by primitive action only. Our agent's results were good and stable from the beginning thanks to the options already learned and are clearly improving towards the end of the learning as shown in Figure 4.12(c). On the other hand, learning by primitive actions only, (flat reinforcement learning) took time before reaching some stabilization, as shown in figure 4.12(d).

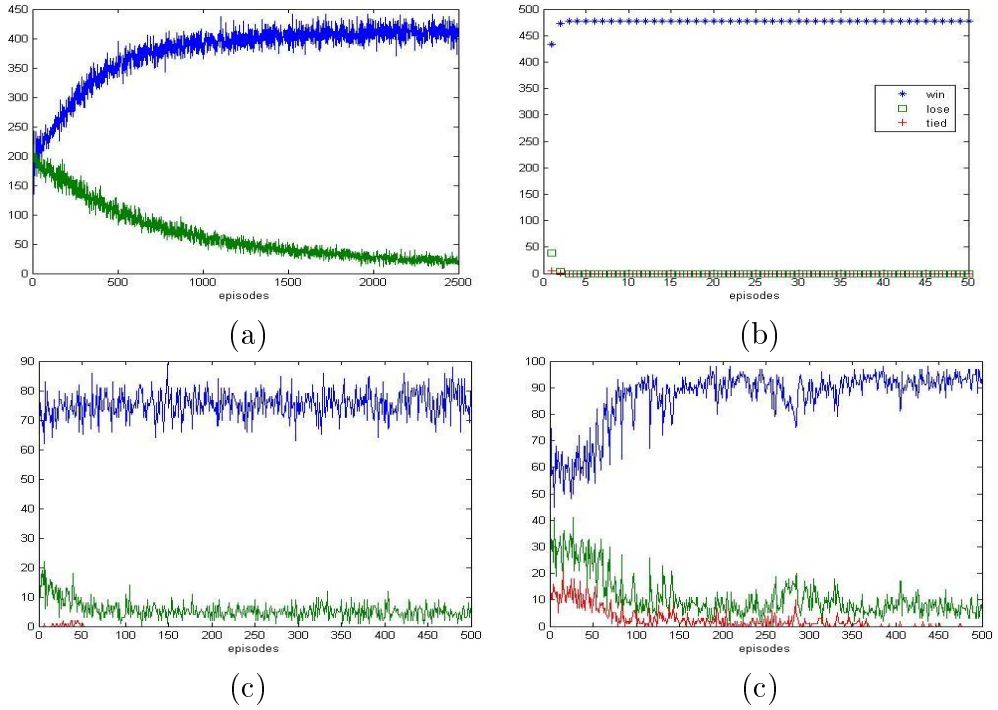


Figure 4.12: Learning outcomes. (a) The ascending curve indicates the number of options that reached their terminal state per episode during phase one of option learning, the descending curve indicates the number of failures. (b) The upper curve indicates the number of options that reached their terminal state in phase two of option learning and the lower curve indicates the number of failures. (c) The top curve shows the number of HRL learning successes in 100 episodes, the bottom curve shows the number of losses in 100 episodes, and the lowest curve indicates the number of ties. (d) The same calculations as in (c) but on flat reinforcement learning results.

To prove the effectiveness of our method, we tested it against the minimax algorithm S. J. Russel [2010]; which is a perfect algorithm for this game, and we compared its results with the flat reinforcement learning algorithm. The number of iterations for the preliminary exploration of the agent with a sense of orientation is 50000, and is the same as the number of iterations of the flat RL, we used the same actor-critic architecture for the flat RL, with the same learning parameters which are the learning rate $\alpha = 0.1$, and the discount factor $\gamma = 0.85$.

As a result, the flat RL agent always loses in front of the minimax algorithm, while our agent always ends with a zero score, which is very good since we cannot hope for better against the minimax algorithm, on the other hand, the response of our agent is immediate which puts it ahead unlike the minimax algorithm which is deterministic and has a high computational complexity S. J. Russel [2010] .

4.3 Conclusion and perspectives

In this work, we have proposed a new method for option discovery in hierarchical reinforcement learning, we have proposed an agent with a direction sense. First, our agent uses his intrinsic motivation for a guided exploration and memorizes a significant extremities of paths he done in this exploration. The agent doesn't need an adjacency matrix which need an integral exploration of the environment, he uses only his direction sense to find the shortest paths. The most visited states by these shortest paths are considered as subgoals or terminal states of options that will be discovered using the shortest paths too. Policies will be learned in very reduced time since the agent takes advantage of his past experience to find his paths to the sub-goals, therefore, his low-level learning is not entirely based on a random walk, because he already knows most of the shortest paths to the sub goals.

Our approach was tested on maze problems and Tic-tac-toe game, at the results our agent gave very satisfactory behaviors in the general and special cases of all the test experiences. The agent endowed with a direction sense succeeds in finding the shortcuts, and not to be distracted by windows which leads to nowhere, he succeeded in discovering the entire space of options without the need for a adjacency matrix or a complete exploration of the environment, in addition to rapid learning using his past experience.

For future work, we plan to explore the trail of deep learning and neural networks as in Tani and Nolfi [1999], to improve the capabilities of our agent in a much more complex environment, to make our agent even more flexible and faster in the phase of using his sense of direction to discover the shortest paths as well as the sub-goals or bypassing obstacles.

Conclusions and future work

The objective of the work presented in this thesis is to present new methods for the automatic discovery of options for HRL, hierarchical reinforcement learning, which appeared as a solution to the problem of dimensionality from which suffers classical RL. The solution given by the HRL consists in dividing a complex problem, often modeled in the form of an MDP, into sub-problems called options or abstract actions. Each treated as a separate RL problem, along with the top-level option that coordinates those at the lower level, hence the name Hierarchical RL.

We started by presenting the MDPs which form a suitable framework for the treatment of problems of the RL type and planning. Subsequently, we described the most popular dynamic programming algorithms; i.e. policy iteration and value iteration, and RL algorithms which are Monte Carlo algorithms, temporal difference algorithms and recent policy gradient methods. With regard to HRL we presented the SMDP framework and the options framework.

We continued with a study of existing methods for the automatic discovery of options, and we proposed a classification of these methods into three classes, the greedy methods which are the methods which discover the components of the options in an iterative way and perform the learning of policies Simultaneously and proportionally, two-step methods which discover the components (termination states and initiation states) of the options in the first step and then learn the policies, and finally the hybrid methods which aim to discover termination states first, then discover initiation states by training policies in parallel.

We have proposed two methods for the automatic discovery of options. The FAOD method: (Fast Automatic Option Discovery) which consists in making a rapid and efficient exploration of the environment to find the sub-goals, considered as salient changes in sensorimotor flow, detected during navigation. This method can be classified under the two-step methods, where we first discover the initiation states and the termination states, then we perform option policy learning and option learning. For both cases we used an actor-critic learning which is a temporal difference learning. We have tested and

validated this method on maze-like problems. Our agent managed to discover the right sub-goals and the best options with a very fast strategy.

The second contribution consists of an agent endowed with a waifinding sense for the automatic discovery of options. In this method we have exploited several notions of reinforcement learning and navigation, starting with the intrinsic motivation with which our agent manages its exploration of paths to avoid to be entirely random. The agent then seeks for each random path one corresponding optimal path, without resorting to graph theory, but by exploiting the second notion which is the sense of direction. To find the sub-goals, our agent calculates the local maxima of the states visited by the optimal paths. The other notion used in this work is the use of past experience in the learning phase, instead of doing a direct RL, we start by using the optimal paths found previously. The procedure of RL by the actor-critic method is only used if the agent is in states that are not part of the optimal paths leading to the sub-goal of the option. We tested this method on maze-like problems, and we adapted, tested, and validated it on the Tic-Tac-Too game.

We have tested and applied our two methods on discrete and proportionally simple problems, our first solution is very fast and efficient, but only applies to exploration and navigation problems. The second approach is more easily adaptable to other types of discrete problems. For future work, we want more adaptability of our option discovery methods to more complicated environments and can be moved to continuous problems, for this we propose to make use of modern deep learning techniques, as follows:

- Use recurrent neural networks as in Nolfi and Tani [1999] for a mixture of experts learning competitively to discover options.
- Adapt graphical or spatial problems to CNNs for the discovery of termination states and option initiation states.
- Use unsupervised learning or self-adaptation methods for an automatic division of problems into sub-problems.
- Exploit deep learning techniques on actor-critic methods Lillicrap et al. [2015], Wierstra et al. [2010] for learning option policies for continuous problems.

Bibliography

R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 2018.

R. S. Sutton, Precup .D, and Singh .S. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, 1999.

Richard S Sutton and Andrew G Barto. Toward a modern theory of adaptive networks: expectation and prediction. *Psychological review*, 88(2):135, 1981.

R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

Andy McGovern, Dona Precup, Balaraman Ravindran, Sathinger Singh, and Richard S Sutton. Hierarchical optimal control of mdps. In *Proceedings of the Tenth Yale Workshop on Adaptive and Learning Systems*, pages 186–191. 1998.

Dona Precup and Richard S Sutton. Multi-time models for temporally abstract planning. In *Advances in Neural Information Processing Systems, 10*, pages 1050–1056. Cambridge, MA: MIT Press, 1998.

Matthew M. Botvinick, Yael Niv, and Andrew C. Barto. Hierarchically organized behavior and its neural foundations: a reinforcement learning perspective. *Cognition*, 113(3):262–280, 2009.

Olga Kozlova. *Hierarchical and Factored Reinforcement Learning*. PhD thesis, Université Pierre et Marie Curie-Paris 6, 2010.

Thomas Degris. *Apprentissage par renforcement dans les processus de décision Markoviens factorisés*. PhD thesis, Université Paris VI, 2007.

Matthieu Geist. *Optimisation des chaînes de production dans l'industrie sidérurgique : une approche statistique de l'apprentissage par renforcement.* PhD thesis, Université Paul Verlaine, Metz, 2009.

Peter Dayan and Yael Niv. Reinforcement learning: The good, the bad and the ugly. *Current Opinion in Neurobiology*, 18(2):185–196, 2008. ISSN 0959-4388. doi: https://doi.org/10.1016/j.conb.2008.08.003.

E.L. Thorndike. *Animal intelligence:Experimental studies.* New York:MacMillan, 1911.

D Michie. Trial and error. *Science Survey*, 2:129–145, 1961.

A Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research Development*, 3(3):210–229, 1959.

A. K Lopf H. A comparison of natural and artificial intelligence. *SIGART newslette*, 53: 11–13, 1975.

Robert A Rescorla, Allan R Wagner, et al. A theory of pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. *Classical conditioning II: Current research and theory*, 2:64–99, 1972.

Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.

Aurélia Léon. *Apprentissage séquentiel budgétisé pour la classification extrême et la découverte de hiérarchie en apprentissage par renforcement.* PhD thesis, Sorbonne Université, 2019.

Christopher Watkins. Learning from delayed rewards. 01 1989.

Christopher Watkins and Peter Dayan. Technical note: Q-learning. *Machine Learning*, 8:279–292, 05 1992. doi: 10.1007/BF00992698.

Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13(1-2):41–77, 2003.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* New York: Wiley & Sons, 1994.

Ronald A. Howard. *Dynamic Probabilistic Systems: Semi-Markov and Decision Processes*. New York: Wiley & Sons, 1971.

Andrew Barto, Singh Satinder, and Chentanez Nuttapong. Intrinsically motivated learning of hierarchical collections of skills. *Proceedings of the 3rd International Conference on Development and Learning*, 01 2004.

Christopher M Vigorito and Andrew G Barto. Intrinsically motivated hierarchical skill learning in structured environments. *IEEE Transactions on Autonomous Mental Development*, 2(2):132–143, 2010.

Harlow Margaret. K Harlow Harry. F and Meyer Donald. R. Learning motivated by a manipulation drive. *Journal of Experimental Psychology*, 40:228–234, 1950. doi: 10.1037/h0056906.

Christopher M. Vigorito. *Intrinsically Motivated Exploration in Hierarchical Reinforcement Learning*. PhD thesis, University of Massachusetts - Amherst, 2016.

Pierre-Yves Oudeyer and Frederic Kaplan. What is intrinsic motivation? a typology of computational approaches. *Frontiers in Neurorobotics*, 1, 2009. ISSN 1662-5218. doi: 10.3389/neuro.12.006.2007. URL `https://www.frontiersin.org/article/10.3389/neuro.12.006.2007`.

Jürgen Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. 1991.

Özgür Şimşek and Andrew G. Barto. An intrinsic reward mechanism for efficient exploration. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, page 833–840. Association for Computing Machinery, 2006. doi: 10.1145/1143844.1143949.

Gary L Allen. Spatial abilities, cognitive maps, and wayfinding - bases for individual differences in spatial cognition and behavior. / edited by reginald g. golledge. In *Wayfinding Behavior, Cognitive Mapping and Other Spatial Processes*. Johns Hopkins University Press, Baltimore, Md., 1999.

Reginald G. Golledge. Human wayfinding and cognitive maps. / edited by reginald g. golledge. In *Wayfinding Behavior, Cognitive Mapping and Other Spatial Processes*. Johns Hopkins University Press, Baltimore, Md., 1999.

Raubal Martin. *Agent-Based Simulation of Human Wayfinding: A Perceptual Model for Unfamiliar Buildings*. PhD thesis, Vienna University of Technology, Faculty of Science and Informatics, 2001.

Jean. Piaget and Barbel. Inhelder. *The Child's Conception of Space*. Norton, New York, 1967.

R.G. Golledge and R.J. Stimson. *Spatial Behavior: a geographic perspective*. New York: Guilford Press, 1997. ISBN 1-57230-050-7.

Andrew.I Frank. Spatial reasoning:theoretical considerations and practical applications. In *EGIS'92, Third European Conference and Exhibition on Geographical Information Systems*, 1992.

Benjamin Kuipers. The "map in the head" metaphor. *Environment and Behavior*, 14 (2):202–220, 1982. doi: 10.1177/0013916584142005.

E. C TOLMAN. Cognitive maps in rats and men. *Psychological review*, 55(4):189–208, 1948. doi: 10.1037/h0061626.

Kevin Lynch. *The Image of the City*. The MIT Press, 1964. ISBN 9780262620017.

Jerry Weisman. Evaluating architectural legibility: Way-finding in the built environment. *Environment and Behavior*, 13(2):189–204, 1981. doi: 10.1177/0013916581132004.

Benjamin Kuipers. Modeling spatial knowledge. *Cognitive Science*, 2(2):129–153, 1978. ISSN 0364-0213. doi: https://doi.org/10.1016/S0364-0213(78)80003-2.

David Leiser and Avishai Zilbershatz. The traveller: A computational model of spatial network learning. *Environment and Behavior*, 21(4):435–463, 1989. doi: 10.1177/0013916589214004.

Drew McDermott and Ernest Davis. Planning routes through uncertain territory. *Artificial Intelligence*, 22(2):107–156, 1984. ISSN 0004-3702. doi: https://doi.org/10.1016/0004-3702(84)90045-6.

Gordon I. McCalla, Larry Reid, and Peter F. Schneider. Plan creation, plan execution and knowledge acquisition in a dynamic microworld. *International Journal of Man-Machine Studies*, 16(1):89–112, 1982. ISSN 0020-7373. doi: https://doi.org/10.1016/S0020-7373(82)80073-4.

Susan L. Epstein. Spatial representation for pragmatic navigation. In Stephen C. Hirtle and Andrew U. Frank, editors, *Spatial Information Theory A Theoretical Basis for GIS*, pages 373–388, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

Sucharita Gopal, Roberta L. Klatzky, and Terence R. Smith. Navigator: A psychologically based model of environmental learning through navigation. *Journal of Environmental Psychology*, 9(4):309–331, 1989. ISSN 0272-4944. doi: https://doi.org/10.1016/S0272-4944(89)80012-X.

S Gopal and T R Smith. Human way-finding in an urban environment: A performance analysis of a computational process model. *Environment and Planning A: Economy and Space*, 22(2):169–191, 1990. doi: 10.1068/a220169.

Michael O'Neill. A biologically based model of spatial cognition and wayfinding. *Journal of Environmental Psychology*, 11(4):299–320, 1991. ISSN 0272-4944. doi: https://doi.org/10.1016/S0272-4944(05)80104-5.

Bernhard Hengst. Discovering hierarchy in reinforcement learning with hexq. In *ICML*, volume 2, pages 243–250, 2002.

Sander G van Dijk and Daniel Polani. Grounding subgoals in information transitions. In *Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2011 IEEE Symposium on*, pages 105–111. IEEE, 2011.

Marlos C. Machado, Marc G. Bellemare, and Michael H. Bowling. A laplacian framework for option discovery in reinforcement learning. *CoRR*, abs/1703.00956, 2017.

M. Yang X. Xu and G. Li. Constructing temporally extended actions through incremental community detection. *Hindawi,Computational Intelligence and Neuroscience*, 2018(2085721), 2018.

Aurelia Leon and Ludovic Denoyer. Budgeted hierarchical reinforcement learning. pages 1–8, 07 2018. doi: 10.1109/IJCNN.2018.8489459.

Emma Brunskill and Lihong Li. Pac-inspired option discovery in lifelong reinforcement learning. In *International Conference on Machine Learning*, pages 316–324, 2014.

Özgür Şimşek and Andrew G. Barto. Skill characterization based on betweenness. In *Advances in neural information processing systems*, pages 1497–1504. Cambridge, MA: MIT Press, 2009.

Aravind .S Lakshminarayanan, Ramnandan Krishnamurthy, Peeyush Kumar, and Balaraman Ravindran. Option discovery in hierarchical reinforcement learning using spatio-temporal clustering. *arXiv preprint arXiv:1605.05359*, 2016.

Z. Koudad. Faod: Fast automatic option discovery in hierarchical reinforcement learning. *International Journal on Artificial Intelligence Tools*, 30(02):2150006, 2021.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *EMNLP*, 2014.

Daan Wierstra, Alexander Förster, Jan Peters, and Jürgen Schmidhuber. Recurrent policy gradients. *Logic Journal of the IGPL*, 18:620–634, 10 2010. doi: 10.1093/jigpal/jzp049.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937. PMLR, 2016. URL `https://proceedings.mlr.press/v48/mniha16.html`.

Sahil Sharma, Aravind S. Lakshminarayanan, and Balaraman Ravindran. Learning to repeat: Fine grained action repetition for deep reinforcementlearning. *CoRR*, abs/1702.06054, 2017. URL `http://arxiv.org/abs/1702.06054`.

M. J. Kearns and S. P. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2-3):209–232, 2002.

Ronen I. Brafman and Moshe Tennenholtz. R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *J. Mach. Learn. Res.*, 3:213–231, mar 2003. doi: 10.1162/153244303765208377.

Marc Pickett and Andrew G. Barto. Policyblocks: An algorithm for creating useful macro-actions in reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 506–513. Morgan Kaufmann, 2002.

Sridhar Mahadevan. Proto-value functions: Developmental reinforcement learning. In *22 nd International Conference on Machine Learning,(ICML)*, pages 553–560, 01 2005.

Andrew G Barto, George Konidaris, and Christopher Vigorito. Behavioral hierarchy: exploration and representation. In *Computational and Robotic Models of the Hierarchical Organization of Behavior*, pages 13–46. Springer, 2013.

Craig Boutilier, Richard Dearden, and Moises Goldszmidt. Exploiting structure in policy construction. pages 1104–1113, 01 1995.

Degris .T, Sigaud .O, and Wuillemin .P.H. Learning the structure of factored markov decision processes in reinforcement learning problems. In *Proceedings of the 23rd International Conference on Machine Learning (ICML'06), Pittsburgh, PA*, pages 257–264, 01 2006.

Matthew Botvinick and Ari Weinstein. Model-based hierarchical reinforcement learning and human action control. *Phil. Trans. R. Soc. B*, 369(1655):20130480, 2014.

Jun Tani and Stefano Nolfi. Learning to perceive the world as articulated: an approach for hierarchical learning in sensory-motor systems. *Neural Networks*, 12(7-8):1131–1141, 1999.

Alec Solway, Carlos Diuk, Natalia Córdova, Debbie Yee, Andrew G Barto, Yael Niv, and Matthew M Botvinick. Optimal behavioral hierarchy. *PLoS computational biology*, 10 (8):e1003779, 2014.

Stefano Nolfi and Jun Tani. Extracting regularities in space and time through a cascade of prediction networks: The case of a mobile robot navigating in a structured environment. *Connect. Sci.*, 11:125–148, 1999.

Satinder Singh, Andrew G. Barto, and Nuttapong Chentanez. Intrinsically motivated reinforcement learning. In *Proceedings of the 17th International Conference on Neural Information Processing Systems*, pages 1281–1288. MIT Press, 2004.

D . Ernest S. J. Russel, P. Norvig. *Artificial Intelligence: a Modern Approach*. Upper Saddle River, NJ: Prentice Hall, 3rd edition, 2010.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Manfred Otto Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015.

Abstract

The hierarchical reinforcement learning framework breaks down the reinforcement learning problem into subtasks or extended actions called options in order to facilitate its resolution. Different models have been proposed where options were manually predefined or semi-automatically discovered. However, the automatic discovery of options has become a real challenge for research in hierarchical reinforcement learning.
In this thesis we propose two automatic option discovery method for hierarchical reinforcement learning. The first method that we call FAOD (Fast Automatic Option Discovery). In this contribution, we took inspiration from robot learning methods to categorize the sensorimotor flow during navigation. Thus, FAOD agent moves along the walls to discover the rooms' contour, closed spaces, doors and bottleneck regions to define terminate states and initiation sets for options.
In the second contribution our learning agent uses his sense of direction to discover the shortest paths and shortcuts after an exploration based on intrinsic motivation, without resorting to the algorithms of the graph theory, these discoveries subsequently serve to discover the termination conditions and the initiation states of the options. For the learning of options policies, the agent uses his experience of exploration as well as learning by temporal difference strategy. We tested and validated this approach on different maze problems and on the tic-tac-toe game.

Keywords

Hierarchical reinforcement learning; Reinforcement learning; Option Discovery; Markov decision process; Actor-criticlearning, Wayfinding; Intrinsic motivation.

ملخص

يقسم التعلم التعزيزي الهرمي مشكلة التعلم المعزز إلى مهام فرعية أو إجراءات موسعة تسمى خيارات لتسهيل حلها. تم اقتراح نماذج مختلفة حيث تم تحديد الخيارات مسبقًا يدويًا أو اكتشافها بشكل شبه تلقائي. ومع ذلك ، أصبح اكتشاف الخيار التلقائي تحديًا حقيقيًا لبحوث التعلم التعزيزي الهرمي.

في هذه الأطروحة ، نقترح طريقتين تلقائيًا لاكتشاف الخيارات للتعلم التعزيزي الهرمي. الطريقة الأولى تسمى \FAOD (الاكتشاف التلقائي السريع للخيارات). في هذه المساهمة ، استلهمنا من طرق تعلم الروبوت لتصنيف تدفق المستشعرات أثناء التنقل. وبالتالي ، يتحرك وكيل \FAOD على طول الجدران لاكتشاف الخطوط العريضة للغرف ، والمساحات المغلقة ، والمداخل ، ومناطق الاختناق لتحديد حالات الإنهاء ومجموعات البدء للخيارات.

في المساهمة الثانية ، يستخدم وكيل التعلم إحساسه بالاتجاه لاكتشاف أقصر المسارات والاختصارات بعد الاستكشاف بناءً على الدافع الداخلي ، دون اللجوء إلى خوارزميات نظرية الرسم البياني ، ثم تُستخدم هذه الاكتشافات لاكتشاف شروط الإنهاء وحالات بدء الخيار. لتعلم سياسة الخيارات ، يستخدم الوكيل تجربته في الاستكشاف بالإضافة إلى إستراتيجية تعلم فرق التوقيت. اختبرنا هذا النهج وتحققنا من صحته في مشاكل المتاهة المختلفة وفي لعبة تيك تاك تو.

الكلمات الدالة

التعلم التعزيز الهرمي. تعزيز التعلم ؛ اكتشاف الخيارات عملية اتخاذ القرار ماركوف ؛ تعلم الناقد الفاعل ، الشعور بالتوجه ؛ الدوافع الذاتية.

Résumé

L'apprentissage par renforcement hiérarchique décompose le problème d'apprentissage par renforcement en sous-tâches ou actions étendues appelées options afin de faciliter sa résolution. Différents modèles ont été proposés où les options étaient prédéfinies manuellement ou découvertes semi-automatiquement. Cependant, la découverte automatique d'options est devenue un véritable défi pour la recherche en apprentissage par renforcement hiérarchique.
Dans cette thèse, nous proposons deux méthodes de découverte automatique d'options pour l'apprentissage par renforcement hiérarchique. La première méthode s'appelle FAOD (Fast Automatic Option Discovery). Dans cette contribution, nous nous sommes inspirés des méthodes d'apprentissage des robots pour catégoriser le flux sensorimoteur lors de la navigation. Ainsi, l'agent FAOD se déplace le long des murs pour découvrir le contour des pièces, les espaces fermés, les portes et les régions de goulot d'étranglement pour définir les états de terminaison et les ensembles d'initiation pour les options.
Dans la deuxième contribution notre agent apprenant utilise son sens de l'orientation pour découvrir les chemins les plus courts et les raccourcis après une exploration basée sur la motivation intrinsèque, sans recourir aux algorithmes de la théorie des graphes, ces découvertes servent par la suite à découvrir les conditions de terminaison et les états d'initiation des options. Pour l'apprentissage des politiques d'options, l'agent utilise son expérience d'exploration ainsi que la stratégie d'apprentissage par différence temporelle. Nous avons testé et validé cette approche sur différents problèmes de labyrinthe et sur le jeu de tic-tac-toe.

Mots clés

Apprentissage par renforcement hiérarchique ; Apprentissage par renforcement; Découverte d'options ; Processus décisionnel de Markov ; Apprentissage acteur-critique, sens d'orientation; Motivation intrinsèque.