Autonomous University of Barcelona

Master thesis

Solving a Complex Optimization Problem for Product Delivery with Reinforcement Learning and Artificial Neural Networks

Autor:
Daniel Salgado Rojo

Director:
Antonio Lozano Bagén[1]

Master's degree in Modelling for Science and Engineering
Mathematics Department[1]

Wednesday 21st March, 2018

"Artificial Intelligence is the new electricity."

Andrew Ng

Data Scientist and co-founder of Coursera

Abstract

To do.

[1]

Acknowledgements

To do.

Nomenclature

(DEL TFG DE MATES)

To indicate that some magnitude X is a dimensional quantity we use the notation X^*, while for dimensionless quantities or the ones we do not care about their dimensions, simply by X.

The units we specify here are the ones that should be used in the model equations, and for convenience they will usually be the ones from the International System of Units.

Symbol	**Units**	**Description**
t^*	s	Time,
C	arbitrary	Mean field monomer concentration,
c^*	$\mathrm{mol\,m^{-3}}$	Monomer concentration,
V_T	$\mathrm{m^3}$	Total volume of solution,
M_p	$\mathrm{kg\,mol^{-1}}$	Molar mass of a NP,
ρ_p	$\mathrm{kg\,m^{-3}}$	Mass density of a NP,

Contents

Chapter 1

Introduction

Introducción/ co

- Motivation artificial intelligence? AlphaGO,...

1.1 Goal optimization problem

Imagine we are a petrol company which owns some number of petrol stations and we are paying a transport company to bring our product to the stations with their trucks in order to refuel them (see Figure 1.1).

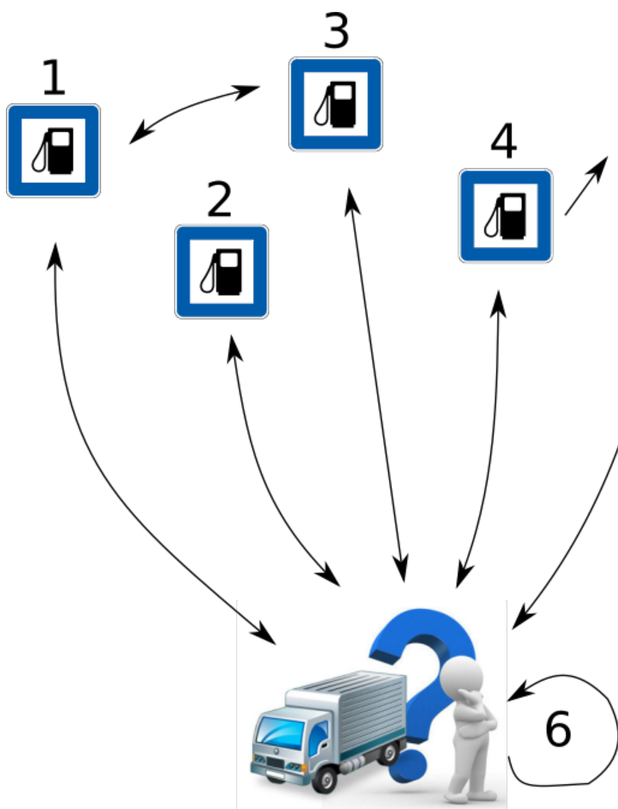


Figure 1.1: Product-delivery problem representation. (Truck image from: `https://www.pinterest.es/pin/521925044288845297/`)

The goal is to minimize the total amount of money (lets say per month or per year) that the petrol company has to pay to the transport company, ensuring that petrol is always available for costumers (among other constraints). Note that this problem can be generalized to the situation in which we have a company that is selling some product, and we have to pay to a transport company to distribute it from the loading dock so that it is available in all of our shops at any time.

1.2 Overview

Overview de las secciones para ubicar al lector.

Chapter 2

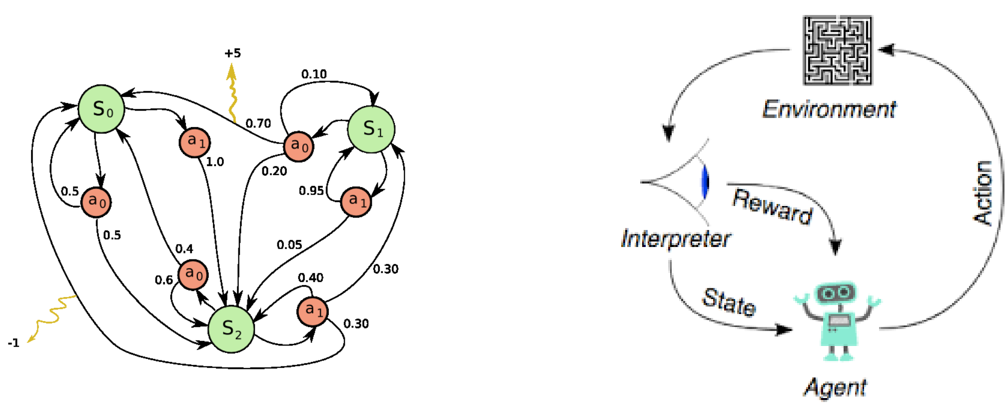
Markov Decision Processes and Reinforcement Learning

Markov Decision Processes (MDP) [2] are the fundamental mathematical formalism for *decision-theoretic planning* (DTP) [3], reinforcement learning (RL) [4, 5, 6] and other learning problems in stochastic domains [7].

In this chapter we introduce the basic concepts about Markov Decision Processes and Reinforcement Learning, focusing on the approaches that are adaptable to solve our goal optimization problem of product delivery. We mainly follow the first chapter in [7].

2.1 Mathematical formalism of Markov Decision Processes

In MDP models an *environment* is modelled as a set of *states* and *actions* that can be performed to control the system's state. The goal is to control the system by means of some *policy*, in such a way that some performance criteria is maximized (or minimized). MDP are commonly used to model problems such as stochastic planing problems, learning robot control and game playing.



(a)

(b)

Figure 2.1: (a) *"Example of a simple MDP with three states (green circles) and two actions (orange circles), with two rewards (orange arrows)"* [8]. (b) *"The typical framing of a Reinforcement Learning (RL) scenario: an agent takes actions in an environment, which is interpreted into a reward and a representation of the state, which are fed back into the agent"* [9].

RL is a general class of algorithms whose goal is to make an *agent* learn how to behave in an environment, where the only feedback consists of a scalar *reward* signal [7]. The goal of the agent is to perform actions that maximize rewards in the long run.

In Figure 2.1a we see the directed graph representation of a simple MDP with three states (green nodes) and some actions (red nodes). In a given state, only some actions can be performed, and each can lead to one or more states according to different probabilities. Transitions between states and the intermediate actions are represented by directed edges. In some state transitions, a positive or negative reward can be obtained (orange arrows) if we think of a MDP in the context of Reinforcement Learning (see Figure 2.1b).

2.1.1 Markov Decision Processes

The definition of MDPs we present at the end of this section consist of states, actions, transitions between states and a reward function. Although general MDPs may have infinite state and action spaces, we focus on discrete MDP problems with finite state and action spaces.

States

We denote $\mathcal{S} = \{s^1, ..., s^N\}$ the set of possible states that can define the environment in a particular instant, and $|\mathcal{S}| = N$. Each $s \in \mathcal{S}$ is going to be considered as a tuple of *features* or properties that uniquely determine the environment state in a certain situation. For instance, in *the tic-tac-toe* game (see Figure 2.2 on the right), a state could be a tuple of 9 positions corresponding to each cell of the game's board and it may contain either 0, 1 or 2 (or any other three symbols) depending on if the cell is empty or occupied by one of the players.

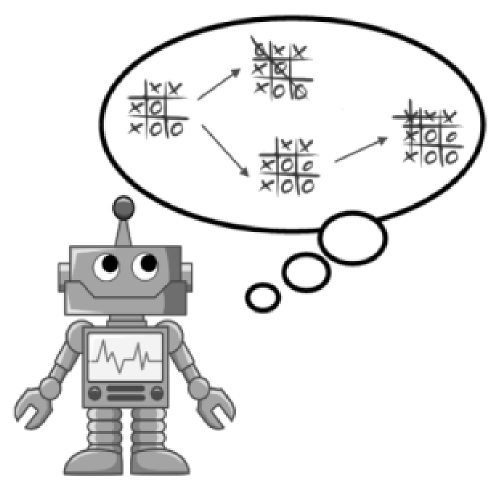


Figure 2.2: `https://mostafa-samir.github.io/Tic-Tac-Toe-AI/`

Actions

We denote $\mathcal{A} = \{a^1, ..., a^\Lambda\}$ the set of actions that can be applied to control the environment by changing its current state; $|\mathcal{A}| = \Lambda$. Usually not all actions are going to be applicable when the system is in a particular state $s \in \mathcal{S}$. Thus, we define $\mathcal{A}(s) \subseteq \mathcal{A}$ the set of actions that are applicable in state s.

The transition operator

If an action $a \in \mathcal{A}$ is applied in a state $s \in \mathcal{S}$, the system makes a transition from s to a new state $s' \in \mathcal{S}$ based on a probability distribution over the set of possible transitions [7]. We define the transition function T as

$$\begin{aligned} T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} &\longrightarrow [0, 1] \\ (s, a, s') &\longmapsto P(s'|s, a) \end{aligned}$$

Hence, $T(s, a, s')$ is the probability of a system in a state s to make a transition to a new state s' after being applied action a.

There are some conditions to be satisfied so that T defines a proper probability distribution over possible next states:

i) For all $s, s' \in \mathcal{S}, a \in \mathcal{A}$, $0 \leq T(s, a, s') \leq 1$,

ii) To model the fact that some actions are not applicable when being in some states, one sets $T(s, a, s') = 0$ for all triples (s, a, s') with $s', s \in \mathcal{S}$ so that $a \notin \mathcal{A}(s)$.

iii) For all $s \in \mathcal{S}, a \in A$, $\sum_{s' \in \mathcal{S}} T(s, a, s') = 1$.

For talking about the *order* in which actions occur on defines a discrete *global clock*, $t \in \{0, 1, 2, ...\}$, so that s_t, a_t denote the state and action at time t, respectively.

Definition 2.1. *The system being controlled is* Markovian *if the result of an action does not depend on the previous actions and visited states, but only depends on the current state, i.e. [7]*

$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, ...) = P(s_{t+1}|s_t, a_t) = T(s_t, a_t, s_{t+1}). \tag{2.1}$$

In our context one assumes that the system being controlled is Markovian.

The rewards function

In classical optimization problems one seeks a *cost function* and aims to either maximize or minimize it under some set of conditions. In the context of RL, one talks about *rewards*, and the goal is to make an agent learn how to maximize the rewards obtained. For the type of problems we are interested to solve, we define the reward function R as

$$\begin{aligned} R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} &\longrightarrow \mathbb{R} \\ (s, a, s') &\longmapsto R(s, a, s') \end{aligned}$$

Thus, R is a scalar feedback signal which can be interpreted as a *punishment*, if negative, or a *reward*, if positive.

For the *tic-tac-toe* example, one could assign either high positive or negative rewards, or zero rewards, to actions that (when being in a given state) lead to win or lose the game, or having a draw, respectively. In this situation, the goal of the agent is to reach positive valued states, which means wining the game. Sometimes the problem is more complex and it is common to assign non-zero reward to combinations of states and actions that are good or bad under some criteria. For instance, having two aligned pieces in *tic-tac-toe* is good, so we may assign a positive reward for these situations.

In conclusion, rewards are used to give a *direction* in which way the MDP system should be controlled [7].

The Markov Decision Process

With all this stuff, we can now give the definition of a *Markov decision process*.

Definition 2.2. *A* ***Markov decision process*** *(MDP) is a tuple* $(\mathcal{S}, \mathcal{A}, T, R)$ *in which* $\mathcal{S}$ *is a finite set of states,* $\mathcal{A}$ *a finite set of actions,* T *a transition probability function* $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ *and* R *a reward function,* $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$*. One says that the pair* T, R *define the* model *of the MDP.*

This definition of MDP allows us to model *episodic tasks* and *continuing tasks*:

- In episodic tasks, there is the notion of *episodes* of some length τ where the goal is to take the agent from some starting state to a *goal* state. In these cases, one can distinguish between *fixed horizon tasks* in which each episode consists of a fixed number of steps, or *indefinite horizon* tasks in which each episode can end but episodes can have arbitrary length [7]. An example for the first type would be *tic-tac-toe* and for the second *chess* board game.

  In each episode the initial state of the system is initialized with some *initial state distribution* $I : \mathcal{S} \to [0, 1]$.

- In continuing or *infinite horizon* tasks the system does not end unless done it in purpose.

2.1.2 Policies

Given an MDP $(\mathcal{S}, \mathcal{A}, T, R)$, for us a (deterministic) policy is a function $\pi : \mathcal{S} \to \mathcal{A}$ that maps states to actions that are applicable in that state, i.e., $\pi(s) = a \in \mathcal{A}(s)$ for all $s \in \mathcal{S}$.

A policy π can be used to make evolve, i.e. to control, a MDP system in the following way:

- Starting from a initial state $s_0 \in \mathcal{S}$, the next action the agent will do is taken as $a_0 = \pi(s_0)$.
- After the action is performed by the agent, according to the transition probability function T and the reward function R, a transition is made from s_0 to some state s_1, with probability $T(s_0, a, s_1)$ and a obtained reward $r_0 = R(s_0, a_0, s_1)$.
- By iterating this process, one obtains a sequence $s_0, a_0, r_0, s_1, a_1, r_1, ...$ of state-action-reward triples which constitute a trajectory (or path) of the MDP.

If the task is episodic, the sequence of state-action-reward triples ends in a finite number of iterations τ, the system is restarted and the process starts again with a new sampled initial state. If the task is continuing, the sequence can be extended indefinitely ($\tau = \infty$) [7].

2.1.3 Optimality Criteria and Discounting

The core problem of MDPs is to find an optimal policy π^* to control the system optimally. Therefore, we need a model for optimality with regards to policies.

Although there are several models of optimality for a MDP, we focus here on the so called *discounted average reward* criteria, which is applicable in both finite (episodic) and infinite (continuing) horizon tasks.

The discounted sum of rewards received or *return* by the agent starting from state s_t is defined as

$$R_t = \sum_{k=0}^{\tau} \gamma^k r_{t+k} \tag{2.2}$$

where $\tau < \infty$ if the task is episodic, and $\tau = \infty$ if it is continuing, and $\gamma \in [0, 1]$ is a discount factor for future rewards ($\gamma < 1$ if the task is continuing).

The factor γ determines the importance of future rewards:

- For $\gamma = 0$ (and taking $\gamma^0 = 1$) the only term that survives in equation (2.2) is the one corresponding to the present reward. In this case the agent will be *myopic* (short-sighted) by only considering current rewards.
- On the contrary, for $\gamma \approx 1$ the discount factor will make the agent strive for a long-term high reward [10]. Put the examples of gamma 0.95 and 0.99, 13, 69
- For episodic tasks and the case $\gamma = 1$, R_t is the sum of rewards at each step of an episode.

The goal of the discounted average reward criteria in the context of MDP is to find a policy π^* that maximizes the expected return, $\mathbb{E}_\pi[R_t]$, over all episodes that start in state $s_t \in \mathcal{S}$. (???? policy depends on initial state?) This idea is formalized with the definition of the so-called value functions.

2.1.4 Value Functions

In order to link the criteria of optimality introduced in the precedent subsection to the policies, one considers the so-called *value functions*, which are a way of quantify "how good" it is for the agent being in a certain state or making a certain action when being in some particular state.

Definition 2.3. *The* ***value of a state*** s ***under policy*** π*, denoted* $V^\pi(s)$ *is the expected return when starting in state* s *and following* π *thereafter [7]. If we consider the discounted average reward criteria, we have the following expression:*

$$V^\pi(s) = \mathbb{E}_\pi\left[R_t | s_t = s\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\tau} \gamma^k r_{t+k} | s_t = s\right] \tag{2.3}$$

where $\tau < \infty$ *if the task is episodic, and* $\tau = \infty$ *if it is continuing.*

Definition 2.4. *The* ***state-action value of*** s***,***a ***under policy*** π*, denoted* $Q^\pi(s, a)$ *is the expected return when starting in state* s*, taking action* a *and thereafter following* π *[7]. For the discounted average reward criteria we have the following expression:*

$$Q^\pi(s, a) = \mathbb{E}_\pi\left[R_t | s_t = s, a_t = a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\tau} \gamma^k r_{t+k} | s_t = s, a_t = a\right] \tag{2.4}$$

where $\tau < \infty$ *if the task is episodic, and* $\tau = \infty$ *if it is continuing.*

The value functions are known to satisfy the so-called *Bellman equations* [11] that we do not introduce in this work (see for example [7, 11, **?**, **?**]).

The main goal of a given MDP is to find the policy that receives the most reward. This is equivalent to find the policy that maximizes the value function for each possible state.

Definition 2.5. *A policy* π^* *is said to be* ***optimal*** *if it is such that* $V^{\pi^*}(s) \geq V^\pi(s)$ *for all* $s \in \mathcal{S}$ *and all policies* π*.* $V^* := V^{\pi^*}$ *is called the optimal value function. Similarly one defines the optimal Q-value,* Q^**.*

Assuming we know the optimal Q-values for each sate-action pair (i.e., $Q^*(s,a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}$), or an algorithm able to estimate them; one can select an optimal action using the greedy Q-policy π_Q defined as

$$\pi_Q(s) = \arg \max_{a \in \mathcal{A}(s)} Q^*(s,a), \qquad \forall s \in \mathcal{S}. \tag{2.5}$$

2.2 Reinforcement Learning

Once we have defined MDPs, policies, optimality criteria and value functions, the next step is to consider the question of how to compute optimal policies. Although there are several approaches, there are usually distinguished two types of algorithms: *model-based* and *model-free*.

Model-based algorithms assume that a model of the MDP is given, i.e. that a transition and a reward function pair (T, R) is known. This class of algorithms is also known as Dynamic Programming (DP).

On the contrary, model-free algorithms, which are under the general name of Reinforcement Learning, do not rely on the availability of a perfect model. “Instead, they rely on *interaction* with the environment, i.e. a *simulation* of the policy thereby generating *samples* of state transitions and rewards” [7]. These samples are then used to estimate the Q-values for each visited state-action pair $(s,a), s \in \mathcal{S}, a \in \mathcal{A}(s)$, by means of some iterative algorithm. Since a model of the MDP is not known a priori, the agent has to learn how to behave by experience. Thus, the agent has to *explore* the MDP to learn about its environment and how to behave in order to maximize the rewards obtained. “This naturally induces a *exploration-exploitation* trade-off which has to be balanced to obtain an optimal policy” [7].

As we are going to see during the following chapters, for the product delivery problem we introduced in section 1.1, we will have a model for the rewards function but we do not know anything about the transition function of our system modelled as an MDP. For this reason, we focus on model-free algorithms. In particular, we consider the Q-learning algorithm to estimate the optimal Q values.

2.2.1 The Q-learning algorithm (WORKING)

The k-th episode, $E_k = (s_0^k, a_0^k, r_0^k, s_1^k, a_1^k, r_1^k, ..., s_{\tau-1}^k, a_{\tau-1}^k, r_{\tau-1}^k, s_\tau^k)$.

Algorithm 1 Q-learning (train) algorithm

procedure Q-LEARNING(γ, $\alpha(a, s)$, τ , `n_episodes`)
    Q $\leftarrow 0$ ▷ Initialise $Q(s, a)$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$.
    for $k \in \{1, ..., \texttt{n\_episodes}\}$ **do**
        $s_0^k \leftarrow \texttt{Random\_choice}(s \in \mathcal{S})$ ▷ The system is initialized to some initial state randomly
        for $t \in \{0, ..., \tau - 1\}$ **do**
            Choose $a_t^k \in \mathcal{A}(s_t^k)$ based on the current Q function and an exploration strategy.
            Perform action a_t^k.
            Observe the new state s_{t+1}^k and the received reward r_t^k.
            Update Q with the following rule:

$$Q(s_t^k, a_t^k) \leftarrow Q(s_t^k, a_t^k) + \alpha_t(s_t^k, a_t^k) \left[r_t^k + \gamma \max_{a \in \mathcal{A}(s_{t+1}^k)} Q(s_{t+1}^k, a) - Q(s_t^k, a_t^k) \right]$$

        end for
    end for
end procedure

Theorem 2.1. *Given a finite MDP $(\mathcal{S}, \mathcal{A}, T, R)$ such that $\mathcal{S}$, $\mathcal{A}$ are finite, $\gamma \in (0, 1)$ and R is deterministic and bounded, the Q-learning algorithm given by the update rule*

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t(s_t, a_t) \left[r_t + \gamma \max_{a \in \mathcal{A}(s_{t+1})} Q(s_{t+1}, a) - Q(s_t, a_t) \right] \tag{2.6}$$

converges with probability 1 to the optimal Q-value function as long as

$$\sum_{t=0}^{\infty} \alpha_t(s, a) = +\infty, \qquad \sum_{t=0}^{\infty} \alpha_t(s, a)^2 < +\infty, \qquad 0 \leq \alpha_t(s, a) < 1 \qquad \textit{for all } (s, a) \in \mathcal{S} \times \mathcal{A} \tag{2.7}$$

Condition (2.7) *requires that all state-action pairs are visited infinitely often.*

Proof. Proof: `http://users.isr.ist.utl.pt/~mtjspaan/readingGroup/ProofQlearning.pdf` ☐

MORE ABOT Q: `https://pdfs.semanticscholar.org/4bf4/495b9b643b3714a45df08f6942811d28d405.pdf`

“ *The convergence of Q-Iearning holds using any exploration policy, and only requires that each state action pair is executed infinitely often. The greedy policy with respect to the Q-values tries to exploit continuously, however, since it does not explore properly, it might result in poor return. At the other extreme random policy continuously explores, but its actual return may be very poor. An interesting compromise between the two extremes is the E-greedy policy, which is widely used in practice [10]. This policy executes the greedy policy with probability 1 - E and the random policy with probability E. This balance between exploration and exploitation both guarantees convergence and often good performance.* ”

Example 2.1.

2.3 Deep Reinforcement Learning (?)

Solo la idea. Concretar en el capítulo correspondiente.

Chapter 3

Problem approach

3.1 Problem description and constraints

A single product is to be delivered to several shops from a depot using several trucks (see Figure 3.1). We call *unload* the fact that a truck delivers (unloads) some quantity of product to a shop; unloads can be either *simple*, if a truck only delivers to one shop and then returns to the depot, or *shared* if a truck delivers to more than one shop before returning to the depot. Trucks can only be loaded in the loading dock or depot.

Moreover, some initial restrictions and assumptions are considered:

1. Trucks leave the depot fully loaded (lets say at 8 in the morning) and return completely empty (lets say at the end of the day, i.e., 23:59:59).
2. Trucks perform at most one *unload* (simple or shared) every day (if necessary?).
3. A truck can not visit the same shop to perform a *unload* more than one time during the same day.
4. Trucks can perform a shared unload of at most V shops.
5. The price that the company owning the shops has to pay to the transport company is given by some cost function J which may depend on the distance travelled by the trucks from the depot to the corresponding shops, the quantity of product delivered and other contributions imposed by the transport company (for example, impose an additional cost if that day is a holiday, an additional cost depending on the number of shops visited for *shared* unloads, etc.).

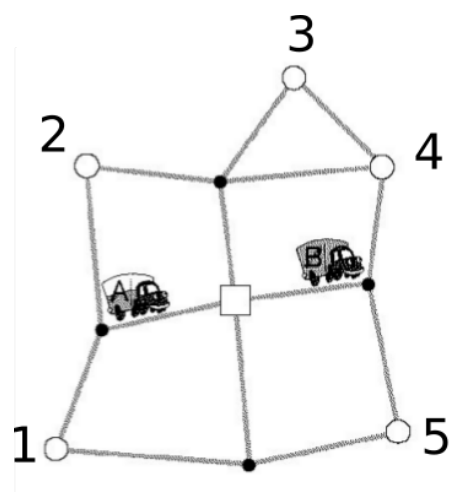


Figure 3.1: Product-delivery domain: the square in the centre is the depot (load doc) and white circles are the shops.. Adapted from [12].

Lets consider a particular shop and imagine that it stores the product in a cilindric container (see Figure 3.2). There are considered several levels of stock:

1. **Minimum capacity or zero level**: when there is no product in stock at all.

2. **Minimum level**: below this level it is considered as having a break of stock (high risk). Thus, we consider that below this level is like having no product in stock.

3. **Danger level**: from this level to the minimum level it is considered that we are taking a moderate risk, so that the shop should receive a *unload* of product soon. The smaller the value of stock below this level, the higher the risk of having a break of stock.

4. **Maximum level**: the desired maximum stock. This level can be overpassed in some quantity but never surpassing the maximum capacity level. The higher the value of stock above this level, the higher the risk of having a break of stock.

5. **Maximum capacity level**: when it is physically impossible to put more product in the container (i.e., however much we compress the products inside the container, we cannot fit more product).

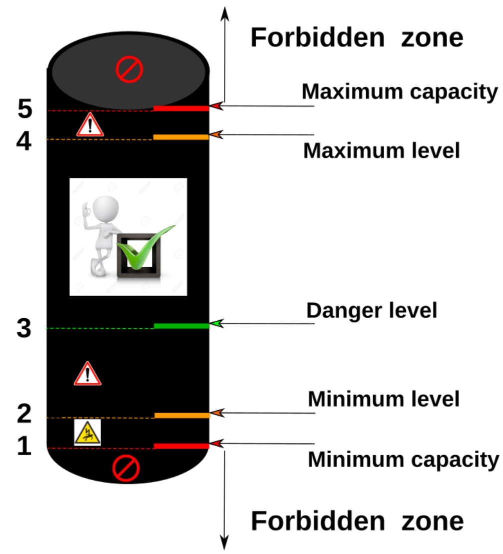


Figure 3.2: Stock levels for an abstract storage container of a given shop.

The ideal and desired state of stock is one between the danger level (3.) and the maximum level (4.).

3.2 Mathematical model (WORKING)

To formalize the model we the use Markov Decision Processes (MDP) nomenclature introduced in chapter 2.

Our system of shops and trucks can be modelled as a weighted and directed graph $\mathcal{G}$, a set of Trucks $\mathcal{K}$ and a set of Tanks $\mathcal{N}$ (the abstract storage containers of each shop, as in Figure 3.2). $\mathcal{G}$ is determined by an adjacency matrix $A_\mathcal{G}$ and a matrix of weights $W_\mathcal{G}$, which are defined as follows:

$$(\mathcal{A}_\mathcal{G})_{ij} = \begin{cases} 1 & \text{if there is a path from shop } i \text{ to shop } j \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

$$(W_\mathcal{G})_{ij} = \begin{cases} w_{ij} \in \mathbb{R}^+ & \text{if there is a path from shop } i \text{ to shop } j \\ \infty & \text{otherwise} \end{cases} \tag{3.2}$$

Hence, weights are used to quantify the cost of going from a shop to another, and this cost is infinity when it is not possible to go from some shop to some other. Note that a component of $W_{\mathcal{G}}$ is ∞ if and only if the corresponding component of matrix $A_{\mathcal{G}}$ is zero.

As an example, the following matrix is the adjacency matrix of the system of 5 shops from Figure 1.1. The depot is considered to be the 6th node.

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

We consider n tanks and k trucks. Thus, $|\mathcal{K}| = k$ and $|\mathcal{N}| = n$.

Each $K_i \in \mathcal{K}$ have the following intrinsic properties:

- An integer id, i,
- A maximum load (maximum capacity) L_i,
- The current load, l_i,
- Its current position: characterized by the tank's id where the truck is, we call it p_i. If $i = 0$, the truck is in the charge station.
- A discrete partition of the interval $[0, L_i]$ with m_i parts: $f_\alpha = [\alpha \cdot L_i/m_i, (\alpha + 1) \cdot L_i/m_i]$, $\alpha = 0, ..., m_i - 1$.

  This leads to a discrete set of possible 'deliveries' that a truck can do when it delivers product to some tank: $\{\lambda_0^{(i)}, ..., \lambda_{m_i-1}^{(i)}\}$, where $\lambda_j^{(i)} = (j + 1) \cdot L_i/m_i$, $j = 0, ..., m_i - 1$.

Each $N_i \in \mathcal{N}$ have the following intrinsic properties:

- An integer id, i,
- A maximum load (maximum capacity) C_i,
- The current load, c_i,
- A discrete partition of the interval $[0, C_i]$ with d_i parts: $h_\beta = [\beta \cdot C_i/d_i, (\beta + 1) \cdot C_i/d_i]$, $\beta = 0, ..., d_i - 1$.

3.2.1 States: simple approach

Real state of the system:

$$s = ((p_1, ..., p_k), (l_1, ..., l_k), (c_1, ..., c_n)) \tag{3.3}$$

Discrete state of the system:

$$s = ((p_1, ..., p_k), (f_{\alpha_1}, ..., f_{\alpha_k}), (h_{\beta_1}, ..., h_{\beta_n})) \tag{3.4}$$

with $0 \leq \alpha_1, ..., \alpha_n < m_i$, $0 \leq \beta_1, ..., \beta_n < d_i$; and f_{α_j}, h_{β_j} are such that $l_j \in f_{\alpha_j}$ and $c_j \in h_{\alpha_j}$, where j varies between 1 and k and between 1 and n respectively.

For the discrete case, the total number of possible states is:

$$|\mathcal{S}| = (n+1)^k \times \prod_{i=1}^{k} m_i \times \prod_{i=1}^{n} d_i \tag{3.5}$$

3.2.2 States: complex approach

Con lo de las previsiones de consumo

3.2.3 Actions

An action consists of a set of new positions for the trucks and a quantity of product to be delivered in the tanks that are in the new truck's position:

$$a = ((p'_1, ..., p'_k), (\lambda^{(1)}, ..., \lambda^{(k)})) \tag{3.6}$$

The dimension of the actions space $\mathcal{A}$ satisfies:

$$|\mathcal{A}| \leq (n+1)^k \times \prod_{i=1}^{k} m_i \tag{3.7}$$

3.2.4 Rewards function

Chapter 4

Application of classical Reinforcement Learning

4.1 A toy model

4.1.1 Simple state's approach

4.1.2 Complex state's approach

4.2 Q-learning simulations

4.2.1 Simple state's approach

4.2.2 Complex state's approach

Chapter 5

Application of Deep Reinforcement Learning

MAIN PAPER: [13]

5.1 Wolpertinger architecture (?)

5.2 (CAPÍTULO 5 AÚN POR DEFINIR)

Chapter 6

Results ??

Chapter 7

Conclusions and Future Work

Bibliography

[1] Daniel Salgado Rojo. Rl-project. `https://github.com/dsalgador/RL-Project`, 2018.

[2] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

[3] Craig Boutilier, Thomas L. Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *CoRR*, abs/1105.5460, 2011.

[4] J. N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, May 1997.

[5] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pages 1057–1063, Cambridge, MA, USA, 1999. MIT Press.

[6] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *CoRR*, cs.AI/9605103, 1996.

[7] Martijn van Otterlo and Marco Wiering. *Reinforcement Learning and Markov Decision Processes*, pages 3–42. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[8] Wikipedia contributors. Markov decision process — wikipedia, the free encyclopedia, 2018. [Online; accessed 18-March-2018].

[9] Wikipedia contributors. Reinforcement learning — wikipedia, the free encyclopedia, 2018. [Online; accessed 18-March-2018].

[10] Wikipedia contributors. Q-learning — wikipedia, the free encyclopedia, 2018. [Online; accessed 20-March-2018].

[11] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[12] Scott Proper and Prasad Tadepalli. Scaling model-based average-reward reinforcement learning for product delivery. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006*, pages 735–742, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[13] Gabriel Dulac-Arnold, Richard Evans, Peter Sunehag, and Ben Coppin. Reinforcement learning in large discrete action spaces. *CoRR*, abs/1512.07679, 2015.