FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Graph Reinforcement Learning for Improving Smart Grid Services

António Bernardo Linhares Oliveira

WORKING VERSION

Mestrado em Engenharia Informática e Computação

Supervisor: Prof. António Costa & Prof. Rosaldo Rossetti

January 28, 2024

Graph Reinforcement Learning for Improving Smart Grid Services

António Bernardo Linhares Oliveira

Mestrado em Engenharia Informática e Computação

January 28, 2024

Resumo

Este documento ilustra o formato a usar em dissertações na Faculdade de Engenharia da Universidade do Porto. São dados exemplos de margens, cabeçalhos, títulos, paginação, estilos de índices, etc. São ainda dados exemplos de formatação de citações, figuras e tabelas, equações, referências cruzadas, lista de referências e índices. Este documento não pretende exemplificar conteúdos a usar. É usado o *Loren Ipsum* para preencher a dissertação.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam vitae quam sed mauris auctor porttitor. Mauris porta sem vitae arcu sagittis facilisis. Proin sodales risus sit amet arcu. Quisque eu pede eu elit pulvinar porttitor. Maecenas dignissim tincidunt dui. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec non augue sit amet nulla gravida rutrum. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Nunc at nunc. Etiam egestas.

Donec malesuada pede eget nunc. Fusce porttitor felis eget mi mattis vestibulum. Pellentesque faucibus. Cras adipiscing dolor quis mi. Quisque sagittis, justo sed dapibus pharetra, lectus velit tincidunt eros, ac fermentum nulla velit vel sapien. Vestibulum sem mauris, hendrerit non, feugiat ac, varius ornare, lectus. Praesent urna tellus, euismod in, hendrerit sit amet, pretium vitae, nisi. Proin nisl sem, ultrices eget, faucibus a, feugiat non, purus. Etiam mi tortor, convallis quis, pharetra ut, consectetuer eu, orci. Vivamus aliquet. Aenean mollis fringilla erat. Vivamus mollis, purus at pellentesque faucibus, sapien lorem eleifend quam, mollis luctus mi purus in dui. Maecenas volutpat mauris eu lectus. Morbi vel risus et dolor bibendum malesuada. Donec feugiat tristique erat. Nam porta auctor mi. Nulla purus. Nam aliquam.

Abstract

Graph Reinforcement Learning is a topic that has earned significant attention and research in the last few years. Combining Deep Reinforcement Learning techniques with an underlying graph-based representation of data enables intelligent agents to learn, optimize and take actions in various scenarios where the data is network-oriented. Although a lot of work has been done on the topic in more recent years, research is still considered to be in an early stage.

Considering the current global challenges associated with sustainability and energy systems, there is an increasing need for advancements in energy-focused intelligent systems to modernize power distribution grids. In the present, renewable energy sources play a major role in reducing the reliance on fossil fuels, which changes the topology of energy distribution systems as consumers gain the capability to generate renewable power. Furthermore, with the improvements in Artificial Intelligence and Machine Learning, systems can be designed to adapt to the decentralization of energy production and efficiently manage energy monitoring and distribution. This translates into the transition to the *smart grid*. Graph Reinforcement Learning can improve these systems, enabling the development of decision-making agents that understand the energy distribution network topology and use it to learn optimal policies for its adjacent control problems.

This dissertation aims to advance the existing research on Graph Reinforcement Learning techniques by proposing a model with concrete improvements to the current ones by better integrating the capabilities of Deep Reinforcement Learning Agents with Graph Neural Networks, which encompass the state-of-the-art techniques regarding Machine Learning on Graphs.

In this context, theoretical and empirical research on these topics is performed, as well as a thorough review of the recent literature that studies and proposes Graph Reinforcement Learning models, gaining an overall perspective of the recent state-of-the-art techniques. Ultimately, this culminates in a proposed implementation of a deep reinforcement learning agent that uses and improves a graph neural network layer.

Lastly, the model is applied to a case study scenario inside the context of smart grid services to evaluate its capabilities and performance. It's expected to design a novel model that intelligently combines Graph Neural Networks with Reinforcement Learning and shows similar or better performance than the other models proposed until the present.

Keywords: Graph Reinforcement Learning, Graph Neural Networks, Deep Reinforcement Learning, Smart Grid

ACM Classification: Computing Methodologies → Machine Learning → Learning Paradigms → Reinforcement Learning

Acknowledgements

To my supervisors for all the guidance and teachings during the development of this work
To my parents who always supported me and expected me to become the best version of myself

António Oliveira

"Man is not worried by real problems
so much as by his imagined anxieties about real problems"

Epictetus

Contents

List of Figures

List of Tables

Listings

Abreviaturas e Símbolos

IT Information Technology

ANN Artificial Neural Network

MLP Multilayer Perceptron

CNN Convolutional Neural Network

RL Reinforcement Learning

MDP Markov Decision Process

DRL Deep Reinforcement Learning

GNN Graph Neural Network

GRL Graph Reinforcement Learning

SAC Soft Actor-Critic

DDPG Deep Deterministic Policy Gradient

DQN Deep Q-Network

PPO Proximal Policy Optimization

GCN Graph Convolutional Network

GAT Graph Attention Network

ADN Active Distribution Network

ESS Energy Storage System

Chapter 1

Introdução

1.1 Context

1.2 Motivation

1.3 Objectives

1.4 Report Structure

Chapter 2

Background Knowledge

2.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are a class of machine learning algorithms based on the neural process of biological learning. The simplest form of an ANN is a Multilayer Perceptron (MLP), also called a feedforward network, whose main objective is to approximate to function f that models relationships between input data x and output data y of considerate complexity [**?**, **?**]. It defines a mapping $y = f(x;\theta)$ and learns the best composition of parameters θ to approximate it to the unknown model. The MLP serves as a fundamental part of developing the other more complex types of neural networks [**?**].

2.1.1 Feedforward Neural Networks

The main building block of a MLP is the *Perceptron*, pictured in figure 2.1, a simple computational model initially designed as a binary classificator that mimics biological neurons' behaviour [**?**]. A neuron might have many inputs x and has a single output y. It contains a vector of *weights*

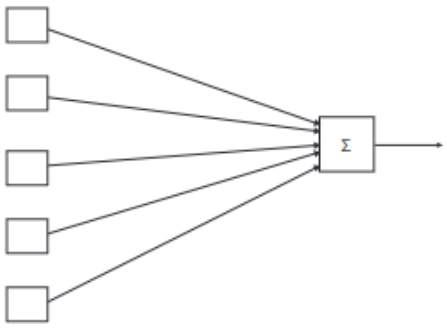

Figure 2.1: The Perceptron [**?**]

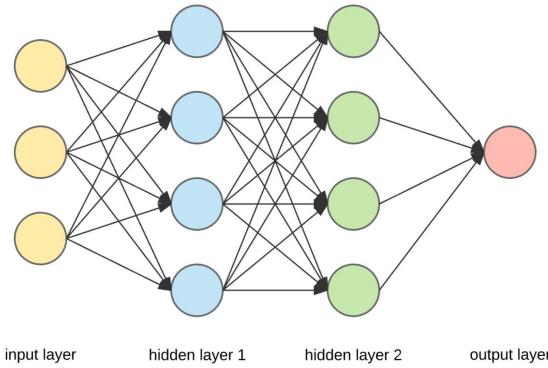

Figure 2.2: Feedforward Neural Network [**?**]

$w = (w_1 ... w_m)$, each associated with a single input, and a special weight b called the *bias*. In this context, a perceptron defines a computational operation formulated as equation 2.1 portrays [**?**].

$$f(x) = \begin{cases} 1 & \text{if } b + \mathbf{w} \cdot \mathbf{x} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

Functions that compute $b + \mathbf{w} \cdot \mathbf{x} > 0$ are called *linear units* and are identified with Σ [**?**, **?**]. An activation function g was introduced to enable the output of non-linear data. The default recommendation is the *Rectified Linear Unit* (*ReLU*), with the Logistic curve (sigmoid) also being very common [**?**].

Feedforward networks are composed of an input layer formed by the vector of input values, an arbitrary number of hidden layers and an output layer, which is the last layer of neurons [**?**]. The greater the amount of layers the higher the *depth* of the network [**?**, **?**].

On its own, the model amounts only to a complex function. Still, with real-world correspondence between input values and associated outputs, a feedforward network can be trained to approximate the unknown function of the environment. In more concrete terms, this involves updating all of the different weight and bias values of each neuron to achieve an output as close as possible to the real or desired value or minimize the total loss, which indicates how distant the network model is to the real function to approximate [**?**, **?**]. *Loss functions* are used to calculate this value.

2.2 Reinforcement Learning

RL consists of a field and a class of machine learning algorithms that study how to learn to take good sequences of actions to achieve a goal associated with a maximizing received numerical reward [**?**]. The main objective is to maximize the received cumulative reward by trying between the available actions and discovering which ones yield the most reward [**?**]. This sequential decision-making process becomes more complex when a delayed reward is considered, given that an action with immediate reward may not always reflect the delayed consequences of that decision [**?**]. It's also the learner's job to consider this during the learning process. These concepts of *delayed*

reward and *trial-and-error search* make up the most important characteristics of Reinforcement Learning [**?**]. The classic formalisation of this problem is the MDP through defining the agent-environment interaction process, explained in the following subsection 2.2.1.

A major challenge in this machine learning paradigm is the trade-off between *exploring* new unknown actions and *exploiting* the already known "good" actions [**?**]. To choose the sequence of actions that return the highest reward, the agent must choose actions it found effective in similar past situations or *exploit* what it learned from experience. Furthermore, given that the agent may not know the action-reward mappings initially, it has to *explore* possible actions that were not selected previously or may initially seem to yield a low reward to compute accurate reward estimates. The main problem is that neither exploitation nor exploration can be favoured exclusively without failing at the task [**?**]. Additionally, an agent's environment is uncertain, and changes in the environment's dynamics may also involve re-estimating action rewards.

In conclusion, RL techniques enable the implementation of sequential decision-making agents that seek to maximize a reward signal analogous to an explicit (complex) goal. The agents need to balance between actions that yield a reward on posterior time steps and actions that produce immediate rewards. In addition, these agents are also faced with the task of balancing the exploitation of information from past experiences and the exploration of new decision paths that could potentially return a higher reward down the road [**?**].

2.2.1 Markov Decision Process

Markov Decision Processs (MDPs) are a classical formalization of a sequential decision-making process, constituting the mathematical definition of the RL problem [**?**, **?**]. Beyond estimating potential rewards for the available actions, the problem defined by MDPs involves learning which actions are optimal in specific situations, i.e. learning a mapping between states of the environment and actions [**?**].

The central component of MDPs is the agent, which acts as a decision-maker and learns from interactions with the environment it's inserted. In a continuous process, the agent takes actions that affect the environment's state, which in turn presents new situations [**?**]. The environment also responds with the reward signals which the agent aims to maximize over time through its decision process.

Formally, the agent-environment interactions, as figure 2.3 entails, occur in a sequence of discrete time steps t, where at each step, the agent receives a representation of the state of the environment $S_t \in \mathcal{S}$ which is used to select an appropriate action $A_t \in \mathcal{A}(s)$, where $\mathcal{S}$ is the set of possible states called the *state space* and $\mathcal{A}(s)$ is the set of available actions for state s [**?**, **?**]. In the next step, the agent receives, as a consequence of its decision, a numerical reward signal $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ and is faced with a new state S_{t+1} [**?**]. Ultimately, the MDP agent follows a logical sequence that occurs as equation 2.2 states. The collection of a state S_t, action taken A_{t+1}, reward R_{t+1} received and next state S_{t+1} constitutes an *experience tuple* [**?**].

$$S_0, A_0, R_1, S_1, A_2, R_2, S_2, A_2, R_3, \ldots \tag{2.2}$$

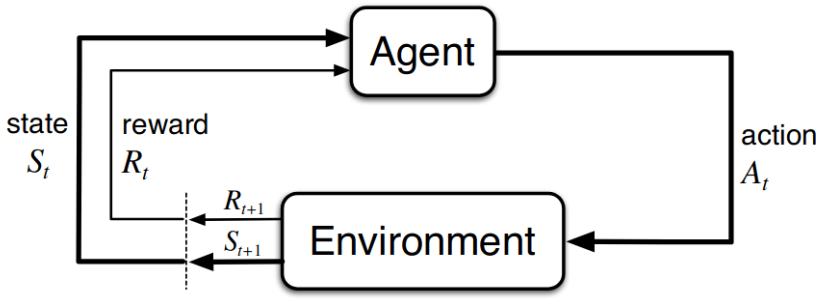

Figure 2.3: Interaction between agent and environment in an MDP [?]

In addition, when the set of possible actions, states and rewards ($\mathcal{A}$, $\mathcal{S}$ and $\mathcal{R}$) are finite, the MDP is said to be *finite* [?]. This results in S_t and R_t having well-defined discrete probability distributions in function of the preceding state and chosen action [?]. Therefore, the probability of receiving a particular reward and state given the previous state and selected action, which characterizes a finite MPD's dynamics, may be characterized by function p defined in equation 2.3

$$p(s',r|s,a) \doteq Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\} \tag{2.3}$$

For all $s,s' \in \mathcal{S}$, $r \in \mathcal{R}$ and $a \in \mathcal{A}(s)$, where $\doteq$ denotes a mathematical formal defintion. This encompasses the assumption that the probability of each possible state, S_t, and reward, R_t, pair is only dependent on the preceding state, S_{t-1}, and action taken, A_{t-1} [?]. Instead of observing this as a restriction on the decision process, it's more convenient to view it as a constraint on the state variable, considering that it must contain all the necessary information from experience to make a valuable decision in the immediate step. If this condition is satisfied, the state is declared to have the *markov property* [?].

From function p in equation 2.3, the state-transtion probabilities, also called the *transition function*, can be computed as described by equation 2.4 [?, ?].

$$p(s'|s,a) \doteq Pr\{S_t = s'|S_{t-1} = s, A_{t-1} = a\} = \sum_{r\in\mathcal{R}} p(s',r|s,a) \tag{2.4}$$

In addition, the expected rewards can be calculated for state-action pairs (equation 2.5) or state-action-next-action triples (equation 2.6) [?, ?].

$$r(s,a) \doteq \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} p(s',r|s,a) \tag{2.5}$$

$$r(s,a,s') \doteq \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r\in\mathcal{R}} r\frac{p(s',r|s,a)}{p(s'|s,a)} \tag{2.6}$$

2.2.2 Rewards and Returns

As stated in the previous subsections, the main goal of a RL agent defined by the numeric reward signal, $R_t \in \mathbb{R}$, it receives from the environment [?]. In this context, the agent's objective is to

maximize the total reward it receives, considering not only immediate but also the cumulative reward over time. In the ubiquitous work of [**?**], the *reward hypothesis* is stated as follows:

> That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward). [**?**]

This also entails that the process of reward maximization from the agent has to be closely tied to it achieving its defined goals in a practical sense. Otherwise, the agent will fail at fulfilling the desired objectives [**?**].

Formally, the goal of an RL agent can be defined by the maximization of the cumulative reward received of time called the *expected return*, Return_t [**?**].

$$\text{Return}_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T \tag{2.7}$$

T describes the final time step. This definition can be applied in domains with a natural notion of a terminal state or final time step. In these cases, the agent-environment interaction process can be broken into logically independent subsequences called *episodes* [**?**]. Each episode ends in a special state, called the terminal state, restarting a new sequence of states and actions completely independent from the previous episode [**?**]. In this context, episodes can be considered to end in the same terminal state, with different accumulated rewards for the different outcomes [**?**].

In contrast, there are situations where the decision-making process doesn't divide itself into logically identifiable episodes but goes on indefinitely. In this case, $T = \infty$ and according to equation 2.7, the expected return the agent aims to maximize would be infinite [**?**]. In this manner, another concept is added in the expected return definition called the *discount rate*, γ where $0 \leq \gamma \leq 1$, representing how strongly the agent should account for future rewards in the expected return calculations, as equation 2.8 [**?**].

$$\text{Return}_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.8}$$

From this equation, we can compute the expected discounted return on a given time step t in the function of the immediate reward signal received and the expected return for the next time step $t+1$, which eases the job of calculating expected returns for reward sequences [**?**]. This is entailed by equation 2.9.

$$\text{Return}_t = R_{t+1} + \gamma G_{t+1} \tag{2.9}$$

In this manner, a MDP can be defined by a tuple with a state space $\mathcal{S}$, an action space $\mathcal{A}$, a transition function p, a reward function r and a discount factor γ, as equation 2.10 portrays [**?**].

$$M = (\mathcal{S}, \mathcal{A}, p, r, \gamma) \tag{2.10}$$

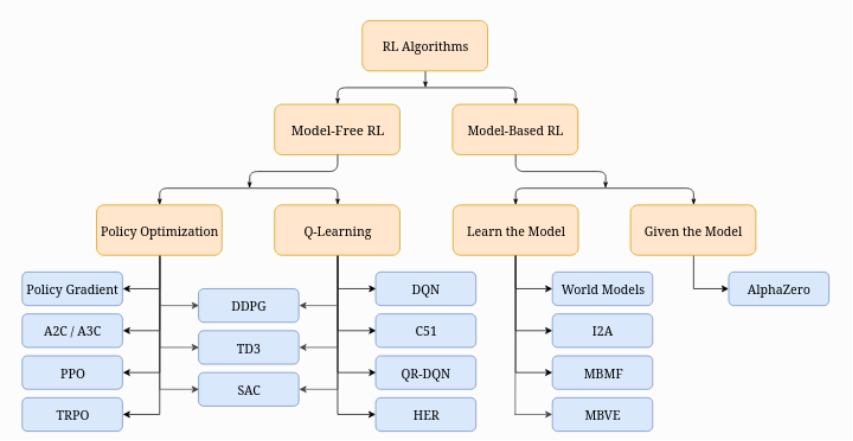

Figure 2.4: Taxonomy of algorithms in modern RL [?]

2.2.3 Policies and Value Functions

RL techniques typically involve the estimation of what is understood as *value functions*, functions that estimate the expected return based on the current state value or state-action pair. This characterizes how good is for an agent to be in a specific state or to take an action in a specific state, respectively, using the expected return to characterize the overall *goodness* of these scenarios [?, ?]. These functions are tied to a specific way of determining the action in a given state. Formally, this is defined as a *policy* π, that defines the probability $\pi(a|s)$ of taking action a in state s [?]. In this context, the *state-value function*, $v_\pi(s)$ and *action-value functions* $q_\pi(s,a)$ for policy π can be defined by equations 2.11 and 2.14, respectively [?].

$$v_\pi(s) \doteq \mathbb{E}_\pi[\text{Return}_t|S_t = s], \forall s \in \mathcal{S} \tag{2.11}$$

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[\text{Return}_t|S_t = s, A_t = a] \tag{2.12}$$

The utility of such functions rely on the possibility of estimating them with regard to past experience of the agent [?]. A fundamental property of value functions is that it can, as was the case with the expected return (equation 2.9), satisfy recursive relationships with the next immediate value as equation 2.13 entails [?]. This equation is called the *Bellman equation for* v_π, which characterizes the relationship between the value of current and subsequent states.

$$v_\pi \doteq \sum_{a \in \mathcal{A}(s)} \pi(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{2.13}$$

$$q_\pi(s,a) \doteq \sum \mathbb{E}_\pi[\text{Return}_t|S_t = s, A_t = a] \tag{2.14}$$

2.2.4 Types of RL

Regarding RL algorithms, they can be divided into model-free and model-based techniques [?]. These categories are distinguished by whether an agent uses a provided or learned *model* of the set

of transition and reward functions, another optional element of RL techniques [**?**, **?**]. In the positive case, the method is said to be model-based, otherwise, it's model-free. Having an accurate model of the environment allows the RL agent to focus on planning ahead by calculating future scenarios and creating policies based on the results of the planning process. An example of a famous system of this kind is AlphaZero [**?**]. However, in most cases, agents can't access a ground-truth model of the environment, leaving only the scenario where an agent learns a model purely from experience. This creates several challenges, the most prominent of which relies on the fact that the model, in most times, doesn't fully capture the environment's transition dynamics, equipping it with bias in relation to the actual dynamics. With this, learning how to generalise the model to real-world environments so that the bias is not over-exploited becomes a very complex task [**?**]. Model-free algorithms can be also further divided into Q-learning and Policy Aproximmation techniques.

Furthermore, algorithms can also be subdivided into on-policy and off-policy methods. [**?**] On-policy algorithms evaluate and improve a single policy used to determine the agent's behaviour [**?**]. The methods under the policy optimization category such as A2C and A3C [**?**] or the Proximal Policy Optimization (PPO) [**?**] almost always fall into this label. In contrast, off-policy algorithms learn how to improve a different target policy based on the results that arise from the policy used to determine the system behaviour initially [**?**]. Such approaches include Q-Learning algorithms such as Deep Q-Networks (DQNs) [**?**, **?**]. Deep Deterministic Policy Gradient (DDPG) [**?**] combines policy optimization with q-learning, consisting of an off-policy method that learns both a q-function and a policy. DDPG constitutes the adaption of q-learning methods to continuous action spaces [**?**]. Another example of an off-policy algorithm is the Soft Actor-Critic (SAC) [**?**] method, which bridges stochastic policy optimization with the DDPG approach and has entropy regularization as one of its central features, which translates into training a policy that maximizes the expected return and entropy, a measure of randomness in the policy [**?**].

Lastly, with the advent of deep learning becoming one of the most ubiquitous techniques in machine learning, RL algorithms have evolved beyond the traditional tabular methods [**?**]. Traditional RL has evolved to Deep Reinforcement Learning (DRL), which studies how to use deep neural networks in RL problems to leverage their generalization abilities for solving more complex problems.

2.3 Graph Representation Learning

Several objects and problems can be naturally expressed in the real world using graphs, such as social networks, power grids, transportation networks, recommendation systems or drug discovery. The usefulness of such representations is tied to how they instinctively represent the complex relationships between objects. However, graph data is often very sparse and complex, and their sophisticated structure is difficult to deal with [**?**, **?**].

Furthermore, the performance of machine learning models strongly relies not only on their design but also on good representations of the underlying information [**?**]. Ineffective representations, on the one hand, can lack important graph features and, on the other, can carry vast amounts

of redundant information, affecting the algorithms' performance in leveraging the data for different analytical tasks [**?**, **?**].

In this context, **Graph Representation Learning** studies how to learn the underlying features of graphs to extract a minimal but sufficient representation of the graph attributes and structure [**?**, **?**, **?**]. Currently, the improvements in deep learning allow representation learning techniques consisting of the composition of multiple non-linear transformations that yield more abstract and, ultimately, more useful representations of graph data [**?**].

2.4 Graph Neural Networks

In the present, deep learning and ANN have become one of the most prominent approaches in Artificial Intelligence research [**?**]. Approaches such as recurrent neural networks and convolutional networks have achieved remarkable results on Euclidean data, such as images or sequence data, such as text and signals [**?**]. Furthermore, techniques regarding deep learning applied to graphs have also experienced rising popularity among the research community, more specifically **Graph Neural Networks (GNNs)** that became the most successful learning models for graph-related tasks across many application domains [**?**, **?**].

The main objective of GNNs is to update node representations with representations from their neighbourhood iteratively [**?**]. Starting at the first representation $H^0 = X$, each layer encompasses two important functions:

- **Aggregate**, in each node, the information from their neighbours
- **Combine** the aggregated information with the current node representations

The general framework of GNNs, outlined in [**?**], can be defined mathematically as follows:

Initialization: $H^0 = X$
For $k = 1, 2, \ldots, K$

$$a_v^k = \text{AGGREGATE}^k\{H_u^{k-1} : u \in N(v)\}$$
$$H_v^k = \text{COMBINE}^k\{H_u^{k-1}, a_v^k\}$$

Where $N(v)$ is the set of neighbours for the v-th node. The node representations H^K in the last layer can be treated as the final representations, which sequentially can be used for other downstream tasks [**?**].

2.4.1 Graph Convolutional Network

A **Graph Convolutional Network (GCN)** [**?**] is a popular architecture of GNNs praised by its simplicity and effectiveness in a variety of tasks [**?**, **?**]. In this model, the node representations in each layer are updated according to the following convolutional operation:

$$H^{k+1} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^k W^k) \tag{2.15}$$

$= A + I$ - Adjacency Matrix with self-connections
$I \in \mathbb{R}^{N \times N}$ - Identity Matrix
$\tilde{D}$ - Diagonal Matrix, with $\tilde{D}_{ii} = \sum_j i_j$
σ - Activation Function
$W^k \in \mathbb{R}^{F \times F'}$ - Laywise linear transformation matrix (F and F' are the dimensions of node representations in the k-th and $(k+1)$ layer, respectively)

$W^k \in \mathbb{R}^{F \times F'}$ is a layerwise linear transformation matrix that is trained during optimization [**?**]. The previous equation 2.15 can be dissected further to understand the *AGGREGATE* and *COMBINE* function definitions in a GCN [**?**]. For a node i, the representation updating equation can be reformulated as:

$$H_i^k = \sigma(\sum_{j \in \{N(i) \cup i\}} \frac{\tilde{A}_{ij}}{\sqrt{\tilde{D}_{ii}\tilde{D}_{jj}}} H_j^{k-1} W^k) \tag{2.16}$$

$$H_i^k = \sigma(\sum_{j \in N(i)} \frac{A_{ij}}{\sqrt{\tilde{D}_{ii}\tilde{D}_{jj}}} H_j^{k-1} W^k) + \frac{1}{\tilde{D}_i} H^{k-1} W^k) \tag{2.17}$$

In the second equation, the *AGGREGATE* function can be observed as the weighted average of the neighbour node representations [**?**]. The weight of neighbour j is defined by the weight of the edge (i, j), more concretely, A_{ij} normalized by the degrees of the two nodes [**?**]. The *COMBINE* function consists of the summation of the aggregated information and the node representation itself, where the representation is normalized by its own degree [**?**].

Spectral Graph Convolutions

Regarding the connection between GCNs an spectral filters defined on graphs, spectral convolutions can be defined as the multiplication of a node-wise signal $x \in \mathbb{R}^N$ with a convolutional filter $g_\theta = diag(\theta)$ in the *Fourier domain* [**?**, **?**], formally:

$$g_\theta \star \mathrm{x} = U_{g_\theta} U^T \mathrm{x} \tag{2.18}$$

$\theta \in \mathbb{R}^N$ - Filter parameter
U - Matrix of eigenvectors of the normalized graph Laplacian Matrix $L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$

The eigendecomposition of the Laplacian matrix can also be defined by $L = U\Lambda U^T$ with Λ serving as the diagonal matrix of eigenvalues and $U^T \mathrm{x}$ is the graph Fourier transform of the input signal x [**?**]. In a practical context, g_θ is the function of eigenvalues of the normalized graph Laplacian matrix L, that is $g^\theta(\Lambda)$ [**?**, **?**]. Computing this is a problem of quadratic complexity

to the number of nodes N, something that can be circumvented by approximating $g_\theta(\Lambda)$ with a truncated expansion of Chebyshev polynomials $T_k(x)$ up to K-th order [**?**, **?**]:

$$g_{\theta'}(\Lambda) = \sum_{k=0}^{K} \theta'_k T_k(\tilde{\Lambda}) \tag{2.19}$$

$\tilde{\Lambda} = \frac{2}{\lambda_{\max}}\Lambda - \mathrm{I}$
$\lambda_{\max}$ - Largest eigenvalue of L
$\theta' \in \mathbb{R}^N$ - Vector of Chebyshev coefficients
$T_k(x)$ - Chebyshev polynomials
$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$ with $T_0(x) = 1$ and $T_1(x) = x$

By combining this with the previous equation, the first can be reformulated as:

$$g_\theta \star \mathrm{x} = \sum_{k=0}^{K} \theta'_k T_k(\tilde{L})\mathrm{x} \tag{2.20}$$

$\tilde{L} = \frac{2}{\lambda_{\max}}L - I$

From this equation, it can be observed that each node depends only on the information inside the K-th order neighbourhood and with this reformulation, the computation of the equation is reduced to $O(|\xi|)$, linear to the number of edges ξ in the original graph G.

To build a neural network with graph convolutions, it's sufficient to stack multiple layers defined according to the previous equation, each followed by a nonlinear transformation. However, the authors of GCN [**?**] proposed limiting the convolution number to $K = 1$ at each layer instead of limiting it to the explicit parametrization by the Chebyshev polynomials. This way, each level only defines a linear function over the Laplacian Matrix L, maintaining the possibility of handling complex convolution filter functions on graphs by stacking multiple layers [**?**, **?**]. This means the model can alleviate the overfitting of local neighbourhood structures for graphs whose node degree distribution has a high variance [**?**, **?**].

At each layer, it can further considered $\lambda_{\max} \approx 2$, which the neural network parameters could accommodate during training [**?**]. With these simplifications, the equation is transformed into:

$$g_{\theta'} \star \mathrm{x} \approx \theta'_0\mathrm{x} + \theta'_1\mathrm{x}(L - I_N)\mathrm{x} = \theta'_0\mathrm{x} - \theta'_1 D^{-\frac{1}{2}}AD^{-\frac{1}{2}} \tag{2.21}$$

θ'_0 and θ'_1 - Free parameters that can be shared over the entire graph

The number of parameters can, in practice, be further reduced, minimising overfitting and minimising the number of operations per layer as well [**?**] as equation 2.22 entails.

$$g_\theta \star \mathrm{x} \approx \theta(I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}})\mathrm{x} \tag{2.22}$$

$\theta = \theta'_0 = -\theta'_1$

One potential problem is the $I_N + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ matrix whose eigenvalues fall in the $[0,2]$ interval. In a deep GCN, the repeated utilization of the above function often leads to an exploding or vanishing gradient, translating into numerical instabilities [**?**, **?**]. In this context, the matrix can be further renormalized by converting $I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ into $\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ [**?**, **?**]. In this case, only the scenario where there is one feature channel and one filter is considered which can be then generalized to an input signal with C channels $X \in \mathbb{R}^{N\times C}$ and F filters (or hidden units) [**?**, **?**]:

$$H = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}XW \tag{2.23}$$

$W \in \mathbb{R}^{C\times F}$ - Matrix of filter parameters
H - Convolved Signal Matrix

2.4.2 Graph Attention Network

Graph Attention Network (GAT) [**?**] is another type of GNNs that focuses on leveraging an attention mechanism to learn the importance of a node's neighbours. In contrast, the GCN uses edge weight as importance, which may not always represent the true strength between two nodes [**?**, **?**].

The Graph Attention Layer defines the process of transferring the hidden node representations at layer $k-1$ to the next node presentations at k. To ensure that sufficient expressive power is attained to allow the transformation of the lower-level node representations to higher-level ones, a linear transformation $W \in \mathbb{R}^{F\times F'}$ is applied to every node, followed by the self-attention mechanism, which measures the attention coefficients for any pair of nodes through a shared attentional mechanism $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \to \mathbb{R}$ [**?**, **?**]. In this context, relationship strength e_{ij} between two nodes i and j can be calculated by:

$$e_{ij} = a(WH_i^{k-1}, WH_j^{k-1}) \tag{2.24}$$

$H_i^{k-1} \in \mathbb{R}^{N\times F'}$ - Column-wise vector representation of node i at layer $k-1$ (N is the number of nodes and F the number of features per node)
$W \in \mathbb{R}^{F\times F'}$ - Shared linear transformation
$a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \to \mathbb{R}$ - Attentional Mechanism
e_{ij} - Relationship Strength between nodes i and j

Theoretically, each node can attend to every other node on the graph, although it would ignore the graph's topological information in the process. A more reasonable solution is to only attend nodes in the neighbourhood [**?**, **?**]. In practice, only first-order node neighbours are used, including the node itself, and to make the attention coefficients comparable across the various nodes, they are normalized with a *softmax* function:

$$\alpha_{ij} = \text{softmax}_j(\{e_{ij}\}) = \frac{exp(e_{ij})}{\sum_{l\in N(i)} exp(e_{il})}$$

Fundamentally, α_{ij} defines a multinomial distribution over the neighbours of node i, which can also be interpreted as a transition probability from node i to each node in its neighbourhood [**?**]. In the original work [**?**], the attention mechanism is defined as a single-layer Feedforward Neural Network that includes a linear transformation with weigh vector $W_2 \in \mathbb{R}^{1\times 2F'}$ and a LeakyReLU nonlinear activation function with a negative input slope $\alpha = 0.2$ [**?**, **?**]. More formally, the attention coefficients are calculated as follows:

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(W_2[WH_i^{k-1}||WH_j^{k-1}]))}{\sum_{l\in N(i)}\exp(\text{LeakyReLU}(W_2[WH_i^{k-1}||WH_l^{k-1}]))} \tag{2.25}$$

$||$ - Vector concatenation operation

The novel node representation is a linear composition of the neighbouring representations with weights determined by the attention coefficients [**?**, **?**], formally:

$$H_i^k = \sigma(\sum_{j\in N(i)} \alpha_{ij} W H_j^{k-1}) \tag{2.26}$$

Multi-head Attention

Multi-head attention can be used instead of self-attention, determining a different similarity function over the nodes. An independent node representation can be obtained for each attention head according to the equation bellow [**?**, **?**]. The final representation is a concatenation of the node representations learned by different heads, formally:

$$H_i^k = \Big\|_{t=1}^{T} \sigma(\sum_{j\in N(i)} \alpha_{ij}^t W^t H_j^{k-1})$$

T - Number of attention heads
α_{ij}^t - attention coefficient computed from the t-th attention head
W^t - Linear transformation matrix of the t-th attention head

Lastly, the author also mentions that other pooling techniques can be used in the final layer for combining the node representations from different heads, for example, the average node representations from different attention heads [**?**, **?**].

$$H_i^k = \sigma(\frac{1}{T}\sum_{t=1}^{T}\sum_{j\in N(i)} \alpha_{ij}^t W^t H_j^{k-1}) \tag{2.27}$$

2.5 Smart Grid Services

Given the global ecological emergency and the increasing energetic crisis, there is a necessity for advancements in energy distribution and transmission systems now more than ever. To fulfil the need for energy sustainability, traditional centralized distribution grids must be adapted to, on the one hand, accommodate the rise of distributed renewable energy sources in corporate and domestic

consumers and, on the other, to make more efficient and reliable distribution of energy resources [**?**, **?**].

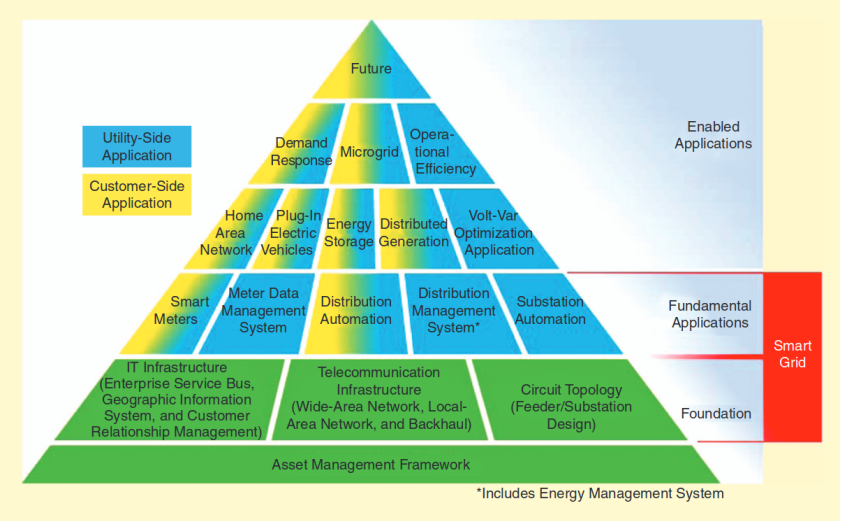

Figure 2.5: Smart Grid Pyramid [**?**]

The *Smart Grid* or the *Smart Power Grid* conceptualizes this modernization of the electricity network by leveraging the technological advancements in information technology and communication science to create intelligent systems that manage and monitor the distributed generation of energy [**?**, **?**]. Figure 2.5 describes the smart grid pyramid with asset management at its base. On this base, the foundation of the smart grid is laid out by the circuit topology, IT systems and telecommunications infrastructure, the basic ingredients for the emergence of the fundamental applications such as smart meters and distribution automation [**?**]. In turn, these serve as building blocks for creating more intelligent systems that leverage upper-layer applications, enabling the true smart grid capabilities [**?**].

Chapter 3

Literature Review

Graph Reinforcement Learning (GRL) or Reinforcement Learning on Graphs is a relatively new area in the broader field of machine learning. GRL techniques have shown significant progress in solving problems with underlying graph-based representations such as power grid management [**?**, **?**], smart transport [**?**, **?**] or task offloading [**?**, **?**]. In this work, the main focus lies on studying the development of GRL techniques and subsequent application to smart grid services such as dynamic economic energy dispatch systems [**?**, **?**], residential electricity behavior identification and energy management [**?**], or Volt-VAR regulation [**?**].

Research on this topic has significantly increased in the last few years with the improvements of DRL techniques and the developments in GNNs in the mid-2010s [**?**, **?**, **?**, **?**]. GNNs became the state-of-the-art for solving numerous data mining tasks involving graph-structured data, excelling at classification, link prediction and representation learning [**?**, **?**]. This advancement brought more sophisticated RL applications on graphs and the surge of a new field studying how to combine the improvements of graph mining and reinforcement learning techniques.

In this context, this literature review is divided into the two main approaches in GRL, which compromise the popular GCN architecture that has been widely researched or leveraging the rising and promising GAT architecture. Lastly, other relevant approaches that use different architectures are also listed.

3.0.1 Plain GCN-Based GRL Techniques

A common approach in Graph Reinforcement Learning model implementation is the use of graph convolutions with the GCNs architecture for leveraging graph-based structures to extract and aggregate the essential features of data in hand and improve the performance of RL agents in those environments. The techniques listed in this subsection constitute approaches that integrate a GCN with RL algorithms.

[**?**] implements a GRL system to improve the decision quality of economic dispatch under high penetration of distributed energy generations. To accomplish this, a SAC system is employed with the main objective of finding the optimal action policy for minimizing generation cost with the appropriate reliability concerns. This problem is represented by an undirected graph with

nodes describing the power grid elements with their respective attributes and edges describing the underlying energy connections between those units. To extract the structural features of the graph, this work implements a full connected layer to perform feature transformation with a two-layer GCN followed by three full connected layers for the non-linear mapping of state-to-action policy in both actor and critic modules. [**?**] develops a similar approach, with both concluding that it significantly reduces learning time for achieving better training results in comparison to plain SAC and showing significant improvement on economy and flexibility of the system on more complex and sparse state graphs. The use of GCNs enables the system to adapt to changes in the state space by leveraging the network's generalization ability.

In [**?**] a three-layer GCN is used to extract node feature and graph topology information and is integrated into a Rainbow-based [**?**] DRL algorithm for electric vehicle charging guidance. In this article, the testing results show promising performance in reducing operation costs for electric vehicle users, portraying a model with good generalization ability in untrained scenarios.

Another interesting implementation of this approach is [**?**], which studies and compares different solutions for optimizing autonomous exploration under uncertain environments. It analyzes combinations of a single agent Deep Q-Network (DQN) and Advantageous Actor-Critic (A2C) with Graph Convolutional Networks, Gated Graph Recurrent Networks and Graph U-Nets. The paper reports that the GCN-DQN was the model that achieved the highest reward during policy training, followed by the GGNN-A2C model, although in the end, it concludes that the second showed improved scalability in relation to the first model. In [**?**] a DQN with a GCN is also used for residential electricity behaviour identification and energy management.

Table 3.1: GCN-Based GRL Techniques

Reference	**DRL Algorithm**	**Application Domain**
[**?**], [**?**]	GCN-SAC	Dynamic economic energy dispatch
[**?**]	GCN-SAC	Multi-access Edge Computing
[**?**]	GCN-A3C	Automatic Virtual Network Embeddings
[**?**]	GCN-DDPG	Automatic transistor sizing
[**?**]	GCN-DQN	Autonomous Exploration under uncertainty
[**?**]	GCN-DQN	Residential electricity behavior identification and energy management
[**?**]	GCN - Modified Rainbow	Electrical Vehicle Charging Guidance
[**?**]	GCN-MDP	Interpret GNNs at model-level
[**?**]	GCN-DQ	Task Offloading in Edge Computing

3.0.2 Attention-based GRL Techniques

Another effective approach in extracting relevant topology and graph features relies on using attention mechanisms to weigh different nodes' contributions dynamically. While this encompasses techniques that use the GAT architecture, which is a GNN design with the attention mechanism at its core, various scholars propose GCN approaches integrated with attention mechanisms such as [**?**] and [**?**]. [**?**] proposes a DDPG-based algorithm improved with a GAT block with three graph Attention Layers for extracting and learning the topology information for achieve real-time optimal scheduling for Active Distribution Networks (ADNs). This paper compares the obtained test results against a GCN-DDPG model and shows increased performance over the GCN method in reducing cost and power loss. Beyond this, the work demonstrates that the GAT's attention mechanism enables the algorithm to focus on more important nodes and improve the signal-to-noise ratio compared to its GCN counterpart. [**?**] and propose a multi-agent approach to the same domain but more focused on voltage regulation with a multi-agent SAC instead of a single-agent DDPG algorithm.

In [**?**], another model for the electric vehicle charging guidance is proposed, consisting of a bi-level approach of a Rainbow-based algorithm with a GAT block. The upper level focuses on the decision-making process regarding charging, while the lower level handles routing. The proposed model proved to be more effective than a shortest distance path-based [**?**] and a DRL-based [**?**] approach. It suggests that in a future direction, developing GNNs directly embedded into the RL framework might further improve the model's robustness and scalability. [**?**] develops a similar approach with a Double-prioritized DQN for the same application domain. In [**?**] and [**?**], the sequential distribution system restoration problem is addressed with a multi-agent RL algorithm equipped with a GCN with an attention mechanism. In the first case, multi-head attention is used as the convolution kernel for the GCN with a DQN algorithm. In the second, self-attention is used for improving the centralized training of the used multi-agent actor-critic algorithm, more concretely, by embedding it in the critic networks. At the same time, the GCN is integrated into the actor networks for extracting the graph features. Both solutions proved more efficient than traditional RL techniques, with the first highlighting its solution generalizability and the second showing increased scalability facing the non-GRL techniques.

Table 3.2: Attention-Based GRL Techniques

Reference	**DRL Algorithm**	**Application Domain**
[**?**]	GAT-DDPG	Optimal Scheduling for ADNs
[**?**]	GAT-MASAC	Multi-agent Voltage Regulation
[**?**]	GAT-Modified Rainbow	Electric Vehicle Charging Guidance
[**?**]	GAT-DQN	Electric Vehicle Charging Guidance
[**?**]	GCN-DQN	Multi-agent Sequential Distribution System Restoration
[**?**]	GCN-MAAC	Multi-agent Service Restoration

3.0.3 Other Approaches

This subsection includes GRL approaches that combine of other GNNs architectures with RL algorithms. In [**?**], a GraphSAGE network is used with a Deep Dueling Double Q-Network (D3QN) for emergency control of Undervoltage load shedding for power systems with various topologies. The author presents promising results for the GraphSAGE-D3QN model compared to a GCN-D3QN, achieving higher cumulative reward and faster voltage recovery speed, although it required longer decision times. The proposed model performed excellently in the application domain and successfully generalised the learned knowledge to new topology variation scenarios.

[**?**] focused on solving the Job shop scheduling problem through a priority dispatching rule with a Graph Isomorphism Network [**?**] and an actor-critic PPO algorithm where the GIN is shared between actor and critic networks. The method showed superior performance against other traditional manually designed priority dispatching rule baselines, outperforming them by a large margin.

3.1 Conclusion

In conclusion, GRL is very promising field, where several different applications and techniques were already studied. GNNs architectures such as GCN have been extensively applied with DRL algorithms for enabling feature extracting from graph-based state representations [**?**, **?**]. Architectures such as the GraphSAGE and other attention-based have also been successfully applied with very promising results [**?**, **?**] in comparison with GCNs. However, less research regarding their integration with DRL algorithms was discovered. This suggests that a possible improvement and research direction in the development of GRL techniques might be connected with exploring the use of different GNNs architectures and using the rising attention-based techniques.

Chapter 4

Problem Statement

In the preceeding chapter, the literature regarding GRL was reviewed, uncovering the recently used approaches in the matter at hand and the promising algorithms and technologies in the field. This chapter aims to drive deeper into the problems related to the topic of this dissertation.

4.1 Graph Reinforcement Learning

4.1.1 Problem

As previously stated troughout most chapters, the main topic of study in this work is Graph Reinforcement Learning (GRL) algorithms. GRL can be defined as a merge between concepts of Reinforcement Learning and Graph Theory. In this context, the formal definition of GRL problem extends the MDP with the added constraint of the environment being sensored as a graph3. Going deeper into this matter, this raises the additional question of how to efficiently represent the received observations by taking computational efficiency as well as data completeness into account, an issue studied by Graph Representation Learning. In this context, we consider that the observable state of the environment can be represented as a graph $G = (V, E)$, where V is the set of nodes and $E \in V \times V$ the set of edges of the environment graph. Each node $v \in V$ At a given time step t, the agent receives an state $s_t = G_t(V_t, E_t)$ of the graph in that instant and computes the optimal action a_t that maximizes the expected return $Return_t$, as defined by equation 2.9.

4.2 Dynamic Economic Dispatch

In the context of the application domain of this dissertation, which is Smart Grid Services, the methods to be studied as solutions of the GRL problem will be applied to the Dynamic Economic Dispatch Problem. We consider this problem under high penetration of distributed generations, *i.e.* with a significant number of distributed generators including renewable energy sources, namely photovoltaic and wind energy generation, and Energy Storage Systems (ESSs).

$$\min F = \sum_{t=1}^{T} F_G(t) + F_{RES}(t) + F_{ESS}(t) \tag{4.1}$$

Chapter 5

Proposed Solution

5.1 Requirements

5.2 Architecture

5.3 Methodology

5.4 Evaluative Methods

5.5 Work Plan

Chapter 6

Conclusions

6.1 Expected Contributions

6.2 SWOT Anaysis

6.3 SMART Analysis

References

[1] Apache. Batik *SVG* Toolkit Architecture, 2005. Available at http://xml.apache.org/batik/architecture.html#coreComponents.

[2] IBM. Program with SVG, 2005. Available at http://www-128.ibm.com/developerworks/xml/library/x-matters40/.

[3] Lipsum. Lorem ipsum, 2008. Available at http://www.lipsum.com/, last accessed in May 2008.

[4] Filipe Marinho, Paulo Viegas, and J. Correia Lopes. SVG na visualização de sinópticos. In José Carlos Ramalho, J. Correia Lopes, and Alberto Simões, editors, *XATA2006, XML: Aplicações e Tecnologias Associadas (Portalege, 9 e 10 de Fevereiro de 2006)*, pages 99–112. Universidade do Minho, 2006.

[5] Manuel A. Matos. Normas para apresentação de dissertações, bases essenciais. Technical report, Faculdade de Engenharia da Universidade do Porto, 1993.

[6] Estelle M. Philips and Derek S. Pugh. *How to Get a PhD*. Open University Press, Fourth edition, 2005.

[7] Ming Tham. Writing research theses or dissertations, 2001. University of Newcastle Upon Tyne. Available at http://lorien.ncl.ac.uk/ming/dept/Tips/writing/thesis/thesis-intro.htm.

[8] W3C World Wide Web Consortium. W3C — About SVG, 2005. http://www.w3.org/TR/SVG/intro.html/.

[9] W3C World Wide Web Consortium. W3C SVG Specification, 2005. Available at http://www.w3.org/TR/SVG11/.

[10] Debora J. Zukowski, Apratim Purakayastha, Ajay Mohindra, and Murthy Devarakonda. Metis: A thin-client application framework. *Proceedings of the Third Conference on Object-Oriented Technologies and Systems*, pages 103–114, 1997.

Appendix A

Lorem Ipsum

Depois das conclusões e antes das referências bibliográficas, apresenta-se neste anexo numerado o texto usado para preencher a dissertação.

A.1 O que é o *Lorem Ipsum*?

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum [3].

A.2 De onde Vem o Lorem?

Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source. Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC. This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet...", comes from a line in section 1.10.32.

The standard chunk of Lorem Ipsum used since the 1500s is reproduced below for those interested. Sections 1.10.32 and 1.10.33 from "de Finibus Bonorum et Malorum" by Cicero are also reproduced in their exact original form, accompanied by English versions from the 1914 translation by H. Rackham.

A.3 Porque se usa o Lorem?

It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout. The point of using Lorem Ipsum is that it has a more-or-less normal distribution of letters, as opposed to using "Content here, content here", making it look like readable English. Many desktop publishing packages and web page editors now use Lorem Ipsum as their default model text, and a search for "lorem ipsum" will uncover many web sites still in their infancy. Various versions have evolved over the years, sometimes by accident, sometimes on purpose (injected humour and the like).

A.4 Onde se Podem Encontrar Exemplos?

There are many variations of passages of Lorem Ipsum available, but the majority have suffered alteration in some form, by injected humour, or randomised words which don't look even slightly believable. If you are going to use a passage of Lorem Ipsum, you need to be sure there isn't anything embarrassing hidden in the middle of text. All the Lorem Ipsum generators on the Internet tend to repeat predefined chunks as necessary, making this the first true generator on the Internet. It uses a dictionary of over 200 Latin words, combined with a handful of model sentence structures, to generate Lorem Ipsum which looks reasonable. The generated Lorem Ipsum is therefore always free from repetition, injected humour, or non-characteristic words etc.