Northern Illinois University
Huskie Commons

Graduate Research Theses & Dissertations | Graduate Research & Artistry

2019

Sky Surveys Scheduling Using Reinforcement Learning

Andres Felipe Alba Hernandez
ahandresf@gmail.com

Follow this and additional works at: https://huskiecommons.lib.niu.edu/allgraduate-thesesdissertations

 Part of the Artificial Intelligence and Robotics Commons, Astrophysics and Astronomy Commons, and the Operational Research Commons

Recommended Citation

Alba Hernandez, Andres Felipe, "Sky Surveys Scheduling Using Reinforcement Learning" (2019). *Graduate Research Theses & Dissertations*. 6789.
https://huskiecommons.lib.niu.edu/allgraduate-thesesdissertations/6789

This Dissertation/Thesis is brought to you for free and open access by the Graduate Research & Artistry at Huskie Commons. It has been accepted for inclusion in Graduate Research Theses & Dissertations by an authorized administrator of Huskie Commons. For more information, please contact jschumacher@niu.edu.

ABSTRACT

SKY SURVEYS SCHEDULING USING REINFORCEMENT LEARNING

Andrés Felipe Alba Hernández, M.S.
Department of Electrical Engineering
Northern Illinois University, 2019
Mansour Tahernezhadi, Director

Modern cosmic sky surveys (e.g., CMB S4, DES, LSST) collect a complex diversity of astronomical objects. Each class of objects presents different requirements for observation time and sensitivity. For determining the best sequence of exposures for mapping the sky systematically, conventional scheduling methods do not optimize the use of survey time and resources. Dynamic sky survey scheduling is an NP-hard problem that has been therefore treated primarily with heuristic methods. We present an alternative scheduling method based on reinforcement learning (RL) that aims to optimize use of telescope resources for scheduling sky surveys.

We present an exploration of RL techniques and we implement a Q learning agent through tabular methods. We compare our implementation with standard methods like the greedy agent and standard frameworks, like Astroplan. We show that tabular-based methods are wholly insufficient for large-scale surveys with large numbers of targets and when long-range planning is required. Future work may be able to use the same environment model and explore approximation methods in order to scale to an input size on the orders of magnitude of an actual sky survey.

NORTHERN ILLINOIS UNIVERSITY
DE KALB, ILLINOIS

AUGUST 2019

SKY SURVEYS SCHEDULING USING REINFORCEMENT LEARNING

BY

ANDRÉS FELIPE ALBA HERNÁNDEZ
© 2019 Andrés Felipe Alba Hernández

A THESIS SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE

MASTER SCIENCE

DEPARTMENT OF ELECTRICAL ENGINEERING

Thesis Director:
Mansour Tahernezhadi

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my mentor, Dr. Brian Nord, at Fermi National Accelerator Laboratory. The door to Dr. Nord's office was always open whenever I ran into trouble or had a question about my research or writing, giving me feedback and supporting me, several times out of normal office hours. He consistently allowed this paper to be my own work but steered me in the right direction whenever he thought I needed it. He was always careful about what resources were needed into accomplish this project, and he did his best to provide the help needed. Dr. Nord has also been a mentor on my career, helping me to develop new skills and visualize a career path. He made me part of his team, listening to my ideas and concerns, including me in different projects with his research team, where I have had the opportunity to grow as a professional.

I would also like to thank Dr. Eric Neilsen because without his expertise in sky surveys, feedback, and collaboration, this work would not been possible. His support checking my code, teaching me, and digging into my bugs to improve performance was fundamental for this work.

I would also like to express my gratitude to my thesis advisor, Dr. Mansour Tahernezhadi of the Department of Electrical Engineering at Northern Illinois University. He recruited me to the Master of Science program and provided me graduate assistantships; without his support, reaching this point would not have been possible. Dr. Tahernezhadi always opened a spot in his busy schedule to follow my thesis project and help me to keep on track delivering this project. His collaboration in other matters like course selections, internship applications, and recommendation letters have been fundamental in building my current set of skills that have allowed me to reach this level of experience and academic formation.

I would also like to thank professor Dr. Benedito Fonseca from Northern Illinois University. Dr. Fonseca advised me in an independent study course in machine learning that allowed me to build the theoretical background to delve deep into this amazing field. His courses and several conversations in his office helped me to develop a set of skills in statistics, probability, and detection theory that are key to understanding how machines learn. Also, Dr. Fonseca was always there to guide me about this path of studying in a new country and balancing family life with work and studies.

I would like to thank Prof. Hassan Ferdowsi and Prof. Edward Miguel for their encouragement; it was a great experience to be a teaching assistant in your courses.

Last but not the least, I am thankful for all the support from friends and family. My wife, Vanessa, during this period sacrificed many things to help me succeed in my studies. To Laura, my sister-in-law, who was always there when I needed any support since I came to the United States. To my friends Jeny and Nando, who opened their house and received us as family. To my brother and my mom, who have always encouraged me in my goals. There are many other friends and people who helped me in this journey; sorry if I missed any names here. I have to say that it was great to follow this path and find support from many of you.

DEDICATION

To my lovely wife, Vanessa, who follows me in this journey and walks always by my side.

To my mom, Isabel, and my brother, Sergio, who have always believed in me.

In memory of my father, Luis, and my grandmother, Maria.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

To uncover the secrets of the sky, scientists need to collect exposures of a variety of objects such as stars, planets, asteroids, galaxies, clusters of galaxies, and streams and clumps of gas. Some structures cover large areas in the sky; others are so uncommon that we need to check over millions of objects to find one. Scientists are pushing to understand interesting things such as dark matter and dark energy and understand big structures of the universe. In order to collect the necessary data, collaborations around the world have been mapping the sky; the task of mapping the sky is called sky survey (e.g., CMB S4, DES, SDSS, LSST). Sky surveys utilize expensive facilities and instruments and the final data set is shared by the scientific community. Obtaining the best data set of exposures in a survey depends on deciding the right exposure at every given time.

The sky survey scheduling problem consist in deciding what exposure may be taken by the telescope depending on previous exposures, current conditions of the sky, and remaining time in the survey while accomplishing certain desired characteristics in the final dataset.

The problem of sky survey scheduling belongs to the category of NP-hard problems [1] [2]. In spite of the similarities with the classical observation scheduling problem [2], the sky survey scheduling problem has enough differences that it is not possible to model correctly with the same approaches like the ones presented by Luo [3] or Miller [1] [4] for the classical observation scheduling problem. First, some concepts of astronomy will be presented, then the problem of sky survey scheduling is going to be discussed in detail, and finally an approach for modeling the problem in the reinforcement learning paradigm will be described.

Astronomical observations consists in the observation of objects in the sky through a telescope. The observable objects in the celestial sphere are not always the same. Therefore, the object of interest needs to be visible in the telescope time assigned for the observation. The goal of the observation is defined by the team of researches who submit a proposal to the organization that manages the telescope.

The classical observation scheduling problem consists in how to assign observation time in a telescope to different research proposals; this problem is faced by facilities administrators. Several studies have done in that direction to optimize the facility utilization [1] [2] [3] [4] [5] [6].

A sky survey consists in mapping the sky for a period of time, usually in the range of years, collecting images (images can be in nonvisible wavelength) with certain features or characteristics. One big difference in sky surveys from traditional observation is the lack of an specific predefined set of exposures. The sky surveys allow to catalog objects, perform statistical analysis, detect transitory events like supernovas, or find moving objects such as asteroids and comets. Sky surveys are important for the study of large-scale structures in the universe, dark matter, and dark energy. Some sky surveys are: the Dark Energy Survey (DES) that was designed to probe the origin of the accelerating universe and understand the nature of dark energy [7], the Sloan Digital Sky Survey (SDSS) that created the most detailed three-dimensional map ever made [8], and the Large Synoptic Survey Telescope (LSST) that is a new project that may start operation in 2023 with the goals of understanding the mysteries of dark energy and dark matter, find hazardous asteroids, explore the remote solar system, study the transient optical sky, and understand the formation and structure of the Milky Way.

The traditional scheduling problem (non-survey) in astronomical observation is a relevant task for the operation of large telescopes and very expensive facilities. Van Rooyen, Miller, Luo, and Mora presented different approaches to provide a solution, optimizing the schedule

under specific constraints, usually given by the facility [2] [3] [4] [9]. However, most of the work done was focused on solving the problem of execution scheduling [2], how to execute a set of predefined exposures in a given period.

In the execution scheduling problem, each proposal contains a set of targets and exposures conditions. All proposals are evaluated by a scientific committee that assigns them a priority [3] [4] [5] [9]. The goal is scheduling targets under specific conditions per exposure while maximizing the scientific reward, given the assign priority. After the observation is executed, the proponents of each proposal receive a set of exposures (images) of their specific project [3]. A lot of work has been done in the astronomical execution scheduling problem driven by the importance of maximizing the scientific return from the operation of the facilities [3] [4] [9] [10]. However, in the context of sky surveys a different scheduling problem emerges because instead of assigning a time for given predefined field and exposure time, the goal is to select the right next field to take a exposure, depending on the previous exposures taken and current sky conditions. Moreover, the final dataset is shared by all the scientific community, so every project or team may receive the same set of exposures with the same conditions.

The sky survey scheduling problem is similar to a dynamic scheduling problem in astronomical observations that is an NP-hard problem [1], but the sky survey scheduling problem has the additional complexity of not having a fixed set of celestial objects to observe but instead a set of characteristics that define scientific return in the final set of exposures. In the case of sky surveys, there are not pre-selected fields with certain scientific priorities but instead different scientific groups may have a set of desired characteristics (constraints) in the final dataset. Some of these characteristics may drive to common goals (similarities in the final dataset) but others may drive competitive goals (dissimilarities in the desire final dataset). The characteristics of each exposure (image) in the final dataset determine the scientific return; an ideal dataset should maximize the usage of the exposures by the scien-

tific community [3]. In spite of the similarities with the classical astronomical scheduling problem, the sky survey scheduling problem has significant differences that justify separate analysis and new research approaches that fit the problem particularities.

The time and data scales handled in astronomy sky surveys are very big; for example, LSST is expecting to collect around 200 petabytes of images and data products during a period of ten years [11]. Additionally, changes in the sky conditions due to weather or atmospheric perturbations, or a problem in facilities that constraint the use of the telescope, among other factors, may force modifications in the schedule of exposures. These scenarios require reestimating the correct schedule online and such a task is really hard unless the scheduler is a flexible tool that can be re-run to take decisions under new conditions or state of the survey. In this work the sky survey scheduling problem is modeled under reinforcement learning (RL) paradigm. First a simplified version of the problem with few targets (possible fields) and a short time frame is used; this version is solved using RL and tabular methods which may converge to the right answer even if they are highly computationally expensive, then future work will approach the same problem with approximation methods that are suitable for problems with a large number of states [12]. The tabular method solution may serve as a reference for evaluating the approximation methods solution and the implementation of the environment. The last step is to scale the approximation solution and check the performance in the sky survey scales where tabular methods will be hard to implement due to memory and computational power limitations.

CHAPTER 2

REINFORCEMENT LEARNING

Learning by interacting with our environment is a common way to think about the nature of learning. When we are young, we explore our environment by taking actions and evaluating the consequences of those decisions. As we get older, it is expected that our experiences make us better decision makers. Undoubtedly a major source of knowledge comes from the interaction with our environment and the consequence of our actions, then future decisions depend strongly on our previous experiences. A wise person is usually the one that who accumulates enough experiences that enable him to infer the consequence of actions under some given scenario; he knows how to take the right decision.

Reinforcement learning (RL) is an artificial intelligence approach where an agent learns from the interaction with an environment. RL is an incredible general paradigm that combines with deep neuronal networks to result in the technique very close to what is called artificial general intelligence. Reinforcement learning has accomplished outstanding results. In 2019 the company DeepMind, a subsidiary company of Google, presented AlphaStar, an agent that was able to play Starcraft II and defeat one of the top professional players. The designers of the agent used a multi-agent reinforcement learning process. This milestone is very promising because Starcraft II is a game with the following characteristic: imperfect information, requires long-term planning, real time, and large action and state space [13].

However, there are also big challenges in the field; one of the main problems is the sample inefficiency of Deep RL algorithm; another is that successful examples hide the difficulty and hard work of creating the environment, and finally this algorithm requires a reward function

that encourages the right behavior in the agent and sometimes the design of such a function is not easy.

2.1 Definition of Elements of Reinforcement Learning

Reinforcement learning is a different paradigm from all other machine learning paradigms. The two main elements of reinforcement learning are the agent and the environment (Figure 2.1). In this paradigm the agent performs actions, and each action changes the state of the environment and generates a reward. Then the environment returns the reward and its future state to the agent. An intelligent agent may maximize the total accumulated reward instead of the immediate reward, in other words a maximization of the expected return.

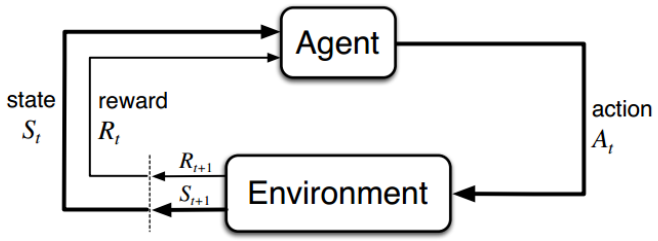

Figure 2.1: Agent and environment interaction. The agent perform actions, receiving rewards and the states from the environment; the agent may learn from this interaction.

Under certain conditions the process of moving among states, accumulating rewards and making decisions based on the future expected return can be modeled as a Markov decision process (MDP). The main condition is that the future depends only on the present and not in the past [14]. Several RL problems can be modeled as a MDP providing a mathematical framework to attack the problem; after having this framework the problem of maximizing the expected return is equivalent to solving the Bellman equations [12].

The reinforcement learning paradigm is composed of the following elements: agent, environment, model, value function, policy function, and state action value pair function.

The *agent* is the learner in the reinforcement learning paradigm. It decides and performs an action and receives the reward and the state from the environment. In the interaction described, the agent may learn from its previous actions and rewards received. For example, in a chess game the agent is the software program that decides the move for a given state of the board game.

The *environment* may return reward and the next state for every action received from the agent. For example, in a chess game the environment is the board game.

A *policy function* is a function that maps a state $s \in S$ with an action $a \in A$, defining the behavior of the agent in the environment.

$$\pi(s) : S \to A \tag{2.1}$$

The *value function* maps the states to real numbers that represent the quality of the state.

$$v(s) : S \to \mathbb{R} \tag{2.2}$$

The value function $v^{\pi}(s)$ describes how valuable it is for the agent to be in certain state while following the policy π. The value function provides a quantitative measure of how good it is to be in a state. The optimal value function (depending on π) is the one that has the highest value for all the states compared to other value functions [14]. In the interaction, agent and environment follow a trajectory of states, actions, and rewards $\{S_n, A_0, R_1, S_{n+1}, A_1, R_2, ...\}$; the obtained rewards can be written as rewards $\{R_1, R_2, R_3, ...\}$. Therefore, reaching one state S_{n+k} opens possibilities for certain trajectories; being in some states may be better than being in others. The equation below describes the *discounted reward*, which is a weighted

summation of future rewards. The parameter γ determines how much weight will be given for reward R_{t+k+1}, k+1 steps from the current state:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.3}$$

The value function can be mathematically defined as the expected return in a state s while following the policy π [12], with γ as the discount factor $0 < \gamma < 1$. The summation goes to infinite for the general case with unlimited states. If the number of states is limited, the summation series can be truncated.

$$v^{\pi}(s) = \mathbb{E}_{\pi}\Big[G_t | S_t = s\Big] = \mathbb{E}_{\pi}\Big[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\Big| S_t = s\Big] \tag{2.4}$$

The same equation can be written in two different ways presented below. For convenience with s' denoting the next state, it can be written as in 2.6.

$$v^{\pi}(s) = \mathbb{E}_{\pi}\Big[G_t | S_t = s\Big] = \mathbb{E}_{\pi}\Big[R_{t+1} + G_{t+1} | S_t = s\Big] \tag{2.5}$$

$$v^{\pi}(s) = \mathbb{E}_{\pi}\Big[R_{t+1} + v_{\pi}(s') | S_t = s\Big] \tag{2.6}$$

The state-action pair Q function maps the state-action pairs to real numbers that represent the quality of the state-action combination.

$$Q : S \times A \to \mathbb{R} \tag{2.7}$$

The state-action pair Q function is a measure of how good it is to perform an action $a \in A$ given a state $s \in S$ while following a policy $\pi(s)$. It can be mathematically defined as the

expected return of performing an action under a state s. The equation 2.8 describes the Q function state-action while following the policy π.

$$Q_\pi(s,a) = \mathbb{E}_\pi\Big[R_{t+1} + \gamma v(S_{t+1})\Big|S_t = s, A_t = a\Big] \tag{2.8}$$

How to find the optimal policy $\pi(s)$ that provides the best trajectory under a given initial state is discussed in this section. A reinforcement learning problem can be mathematically idealized as a Markov decision process (MDP). This involves delayed reward and the need to trade off immediate and delayed reward [12]. Using the definition of policy, state and state-value pair function previously formulated with the idealization of the problem as an MDP, the question of how to find the optimal policy $\pi(s)$ that provides the best trajectory under a given initial state can be approached mathematically. The mathematical framework for this task was introduced by Bellman in 1953. The Bellman optimality equations are presented [12] [14]. The value function for the optimal policy maps the state into a value of quality while following the best policy.

$$v_*^\pi(s) = \max_\pi v_\pi(s) \tag{2.9}$$

The state-action pair Q functions map the state-action combination into a value of the expected return while following the optimal policy π_* The equation 2.10 describes the Q function state-action while following the optimal policy.

$$Q_{\pi_*}(s,a) = \mathbb{E}_\pi\Big[R_{t+1} + \gamma v_*(S_{t+1})\Big|S_t = s, A_t = a\Big] \tag{2.10}$$

The optimal value function can be written without referencing any specific policy [12] with the equation in 2.6.

$$v_*(s) = \max_{a \in A(s)} Q_{\pi(s)_*}(s, a) \tag{2.11}$$

$$v_*(s) = \mathbb{E}\Big[R_{t+1} + \gamma v_*(S_{t+1}) \Big| S_t = s, A_t = a\Big] \tag{2.12}$$

2.2 Function Representation Methods

Previous sections explained how to describe mathematically reinforcement learning problems (Bellman equation). The policy function, value function, and state-action pair function were formulated. However, these equations need to be represented and solved numerically in a computer. As Professors Sutton and Barto mention in their book [12]. "There is a tension between breadth of applicability and mathematical tractability."

2.2.1 Tabular Representation

For a finite Markov decision process the policy function, state value function, and state-action pair function can be defined through tables. The tables can be implemented in different programming languages through arrays, lists, matrices, numpy arrays or similar data structures depending on the language selected.

Policy table: For n states $s \in S$ and m actions $a \in A$, then the policy $\pi(s)$ can be represented by table $\pi(s)$ array of n rows, where each row contains an index from zero to $m - 1$ that represents the action a performed given the state s.

Value table: For n states $s \in S$, the value function can be represented by a table $v(s)$ (array of n rows) that contains a real number that represents the quality of a state.

State-action pair table: For n states $s \in S$ and m actions $a \in A$, then the state-action pair function can be represented by a table (two dimensional array or matrix) with dimensions $Q_{n \times m}$ that contain a real number that represents the quality of the state-action combination.

2.2.2 Parametric Representation

Although tabular methods described previously are a good approach to test the environment and gain some insights about the problem, the computational cost of tabular methods is very high [12] [14]. For the sky survey scheduling problem with the model described before, the computational power and the amount of memory needed it may grow quickly with the number of targets (possible fields). Approximation solution methods are necessary in cases where the number of states is very big, continuous, infinite or when the problem is partially observable [12].

Policy parametrization: The policy function maps the state space S with the action space A, $\pi(s) \rightarrow A$. In this approach the parameter $\boldsymbol{\theta}$ is introduced, then the policy function is written as $\hat{\pi}(s)(a|s, \boldsymbol{\theta})$.

Value function parametrization: A similar parametrization can be used for the value function introducing the parameter $\boldsymbol{w}$. Then the value function may be written as $\hat{v}(s, \boldsymbol{w})$.

Q function parametrization: The value-pair Q function can be also parametrized by introducing $\boldsymbol{w}$ as $\hat{Q}(s, a, \boldsymbol{w})$ with $a \in A$ and $s \in S$.

The parameters $\boldsymbol{\theta}$ and $\boldsymbol{w}$ can be found through different methodologies for function approximation such as stochastic gradient, linear methods, kernel-based function approximation, actor-critic methods, and neuronal networks [12].

2.3 Algorithms in Reinforcement Learning

A collection of algorithms useful for searching the optimal policies and optimal value functions are presented in this section. The problem was formulated as Markov decision process (MDP) and different techniques can be used to search the optimal trajectory. Some of them such as dynamic programming require knowledge of the model and visit all the states in order to find the optimal trajectory; others such as Q learning sample the state space and update the knowledge on the functions at every step. In the subsequence subsections a description of some of these algorithms is presented.

Reinforcement learning problems can be approached by the method of dynamic programming introduced by Bellman in the 1950s. In dynamic programming a complex problem is broken down into simpler subproblems in a recursive manner. Policy iteration is an example of this kind of algorithm used in dynamic programming. The policy iteration algorithm is presented below.

Algorithm 1 Policy Iteration

1: Initialization

Initialize with random values for the policy function $\pi(s)_{new}$ and value function $v(s)$.

2: Policy Evaluation

$\Delta v(s) \leftarrow 0$

loop for each $s \in S$:

$V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \epsilon$, for a small $\epsilon > 0$

3: Policy Improvement

$policy - stable \leftarrow True$

loop for each $s \in S$:

$\pi(s)_{old} \leftarrow \pi(s)_{new}$

$\pi(s)_{new} \leftarrow argmax_a \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$

If $(\pi(s)_{old} \neq \pi(s)_{new})$, then $policy - stable \leftarrow False$

If $policy - stable$, then stop and return $v \simeq v_*$ and $\pi \simeq \pi_*$; else go to 2

2.3.1 Greedy Algorithms

A greedy algorithm makes the locally optimal choice at each step [15]. Greedy algorithms may not find the optimal solution for some problems. An example is the traveling salesman problem; the greedy algorithm may find a less than optimal solution but it terminates in a reasonable amount of steps. In the case of convex optimization problems, the greedy algorithm may converge to the global maximum and find the optimal solution.

A greedy agent always exploits the known information to obtain the maximum reward and it does not explore new actions. The policy followed by a greedy agent is called greedy policy. For a given Q function $Q(s, a)$ the agent will always perform the action a^* that maximizes the reward $\max_{a^* \in A(s)} Q(s, a^*)$. The previous definition of the greedy agent does not define a rule for obtaining the $Q(s, a^*)$ function; the rule for estimating Q establishes how future states s' affect the action-state value.

Greedy one-step agent can be defined by finding the parameter $\gamma = 0$ in the $Q(s, a^*)$ function. This agent reduces the problem of finding the state-action value to the problem of finding the action that provides the maximum immediate reward in a given state. The Q function for this agent is define by the equation below:

$$\max_{a^* \in A(s)} Q(s, a^*) = \max_{a^* \in A(s)} \mathbb{E}\Big[R_{t+1} + \gamma \cdot v_*(S_{t+1}) \Big| S_t = s, A_t = a^*\Big] \tag{2.13}$$

$$\max_{a^* \in A(s)} Q(s, a^*) = \max_{a^* \in A(s)} \mathbb{E}\Big[R_{t+1} \Big| S_t = s, A_t = a^*\Big] \tag{2.14}$$

Algorithm 2 One-Step Greedy Agent

1: For a given state s find the action a^* using equation 2.14
2: Perform the action $a*$ and move from state s to s' and get the reward R_{t+1} (exploit)
3: Update $s \leftarrow s'$ and repeat 1 and 2 until you reach a terminal state.

2.3.2 Temporal Difference (TD) Learning

Temporal difference (TD) methods are classified as model-free learning algorithm because they do not require the model dynamics to be known in advance and they can be applied for non-episodic tasks as well [14]. For example, in policy iteration method described before the knowledge of the one-step dynamic of the environment is required. Temporal difference

methods estimate the unknown values each step by sampling a value and replacing it. The strategy is called TD prediction [14] and the TD rule for updating the value function is described below:

$$v(s) \leftarrow v(s) + \alpha \cdot (r + \gamma \cdot v(s') - v(s)) \tag{2.15}$$

The equation above updates the value of the previous state $v(s)$ after performing an action a that results in the reward r and the future state s'. The term α is the learning rate and γ is the discount factor. The second term of the equation can be seen as an error calculated as the difference between the actual reward $r + \gamma \cdot v(s')$ and the expected reward $v(s)$. Therefore, the goal of the algorithm is to reduce this difference after several iterations. With TD prediction the value function is estimated. With TD control the value function is optimized, in the following sections.

2.3.2.1 Q Learning: Off-Policy Learning TD Algorithm

Q learning method is called "off-policy" because the target policy $\pi(s)$ is infered from data collected following the behavior policy ϵ-greedy [12]. Q learning aims to sample the state-space and estimate the state-action value pair Q function $Q(s, a)$. This temporal difference algorithm updates the value pair $Q(s, a)$ after moving from the current state s to the next state s' due to the action a. The update rule is given by the equation 2.16. The operation $\max_{a^* \in A(s)} Q(s', a^*)$ finds the action a^* that maximizes the expected return at the next state s'. New information is obtained from performing the action a to obtain the reward r and the next state s'; with these two it is possible to calculate $r + \gamma \cdot \max_{a^* \in A(s)} Q(s', a^*)$, subtracting the previous value stored in the state-action value pair $Q(s, a)$ function the temporal difference error in the Q table can be calculated as $(r + \gamma \cdot \max_{a^* \in A(s)} Q(s', a^*) -$

$Q(s,a)$). This difference is multiplied by the learning rate α and added to the previous value of the state-action value pair Q function in order to update the function.

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot \Big(r + \gamma \cdot \max_{a^* \in A(s)} Q(s', a^*) - Q(s,a)\Big) \tag{2.16}$$

The equation 2.16 allows us to update $Q(s,a)$; however, it does not define how to sample the state space. The behavior policy ϵ-greedy determines the sampling of the state space and provides what action s to take in a given state s during the learning process. In Q learning the behavior policy ϵ-greedy is defined by two policies: the greedy policy and a random policy. Greedy policy always take an action s that returns the maximum state-action pair value in the Q table $\max_{a \in A(s)} Q(s,a)$; the random policy only chooses a random action $a \in A(s)$. The ϵ-greedy policy introduces the parameter ϵ (epsilon). The random policy is used with probability epsilon (exploration probability) and the greedy policy is used with probability 1-epsilon (exploitation probability) [14]. In other words, the algorithm sometimes exploits the maximum reward from known information in $Q(s,a)$ and in others it just explores a new actions. If an agent exploits all the time, it can get stuck into a local minimum while following always the same "best trajectory" from the known information. New possible trajectories $s_0, s_1, ...s_n$ can be discover only by sometimes exploring new actions that uncover unknown information. This is known in the literature as the trade-off between exploration and exploitation and it is necessary unless you have complete knowledge of the state space. The target target policy $\pi(s)$ is obtained using the $Q(s,a)$ and choosing always the best action (greedy policy); therefore, the target policy will be optimal only if the $Q(s,a)$ function is close enough to the real one. The Q learning algorithm is presented below in algorithm 3.

Algorithm 3 Q Learning Algorithm

1: Initialize the $Q(s, a)$ function to some arbitrary values.
2: Perform an action $a \in A(s)$ using the epsilon-greedy policy for an epsilon $\epsilon > 0$
3: Update the Q(s,a) using the equation describe in equation 2.16
4: Repeat step 2 and 3 until reach a terminal state s.

In the case of a non-episodic task the algorithm will never reach the terminal state but it can be still used without modification. In order to test the reward of Q learning, the Q function should be used in combination with the greedy policy. This process is detailed in algorithm 4.

Algorithm 4 Test Q Table Algorithm

1: Receive the Q table $Q(s, a)$, and initial state as arguments. Initialize accumulated reward $Rew \leftarrow 0$.
2: Find the greedy action a^* that maximize the Q table $\max_{a^* \in A(s)} Q(s, a^*)$.
3: Perform the action a^*, move from state s to s', and obtain the step reward r (exploit).
4: Update $s \leftarrow s'$, and accumulated reward $Rew \leftarrow (Rew + r)$ repeat 2 and 3 until reach a terminal state.
5: Return the accumulated reward Rew

2.3.2.2 SARSA: On-Policy Learning TD Algorithm

Another TD control algorithm is state-action-reward-state-action (SARSA), an on-policy algorithm. The update rule of SARSA is similar to the one of Q learning but instead of getting the max value of $\max_{a^* \in A(s)} Q(s', a^*)$ it uses only $Q(s', a')$, where a' is an action selected by a epsilon-greedy policy $\epsilon > 0$:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot (r + \gamma \cdot Q(s',a') - Q(s,a)) \tag{2.17}$$

Algorithm 5 SARSA Algorithm

1: Initialize the $Q(s,a)$ function to some arbitrary values.
2: Perform an action $a \in A(s)$ using the epsilon-greedy policy for an epsilon $\epsilon > 0$
3: Update Q(s,a) using the equation describe in equation 2.17
4: Repeat step 2 and 3 until reach a terminal state s

2.3.3 Algorithms Based on Parametrized Functions

Different approximation methods can be used for finding the optimal parametrized functions, such as stochastic gradient and semi-gradient methods, linear methods, least-squares temporal difference, or nonlinear function approximation as artificial neuronal networks [12]. Successful results have been found using non-linear methods; for example, Deep Q Network was used in order to play Atari games. Recurrent neuronal networks were used for building a stronger topology called Deep Recurrent Q Network (DRQN). This network was trained to play the video game Doom. Furthermore, the DRQN was also improved using a topology named DARQN because it included an attention layer on top of the convolutional layer of the DRQN [14]; this network showed better performance playing Doom. Later algorithms such as asynchronous actor critic agent presented by Google DeepMind outperformed DARQN and was able to compete playing Starcraft against human top players of the word [13].

CHAPTER 3

MODELING THE SKY SCHEDULING PROBLEM AS A REINFORCEMENT LEARNING PROBLEM

With the RL paradigm already defined it is important to specify each element for the specific case of sky survey scheduling problem in order to do the actual implementation. The agent's actions are the targets in the sky, in other words, possible fields to take an exposure. The agent takes an action then decides the coordinates of the next target for taking an exposure. Each target is selected at a certain time, so the sequence of time-target pairs rises the schedule of the survey. The goal of the agent is to select this time-target sequence in a given interval of time in such a way that the accumulated reward received from the environment is maximized.

3.1 Environment Model in Sky Surveys

Defining the environment is a little more complex; two elements may be defined: the reward and the state.

3.1.1 Reward Model

The definition of objective function determines the reward. It is a non-trivial task and it requires a detailed discussion among the scientific community [1] [3] [16]. In this work a

metric call *volume survey* is defined. The reward of an action is defined as the change in the volume survey. Defining the volume survey needs other definitions that are presented below.

- Zenith_seeing is the astronomical seeing at the zenith and it can be measured during observation time; for simplicity, it is assumed to be constant value in this work.

- The airmass is the path length for light rays from the celestial source to pass through the atmosphere. The shorter the path, the better exposure; the shortest is usually in the zenith point. The airmass is given by the formula 3.1 using the Hardie approximation [17] where $k_1 = 0.0018767$, $k_2 = 0.002875$, and $k_3 = 0.0008083$. With θ_z as the angle between the zenith and the celestial body:

$$\text{airmass} = \sec\theta_z - k_1 \cdot (\sec\theta_z - 1) - k_2 \cdot (\sec\theta_z - 1)^2 - k_3 \cdot (1 - \sec\theta_z)^3 \tag{3.1}$$

- The seeing is the amount of apparent blurring and twinkling of astronomical objects due to turbulent mixing in the atmosphere of Earth [18], the zenith_seeing the seeing at the zenith. The mathematical equation for calculating the image full width half maximum (FWHM) is presented below [19]:

$$\text{FWHM} = (\text{zenith\_seeing}) \cdot (\text{airmass})^{0.6} \tag{3.2}$$

- Effective exposure time t_{eff}: The quality of the exposure depends on the exposure time, the seeing, and airmass, among other factors. Some of these factors depend on the angle from the zenith to the target as was explained above. Two exposures of the same target under similar sky conditions and same exposure time may have different quality because the angle with the zenith at the moment of the exposure was different. If an exposure is taken in an arbitrary position in the sky with exposure time t_0 obtaining a

quality of exposure q_0, then the effective exposure time t_{eff} is exposure time needed, if the exposure is taken at the zenith in order to produce the same exposure quality q_0. The t_{eff} can be calculated with the following equation:

$$t_{eff} = \left(\frac{0.9}{FHWM} \right)^2 \tag{3.3}$$

- The volume survey increases as a function of the effective exposure time. The concept of volume survey emerged from the fact that taking longer exposures or several exposures of the same field increases the signal to noise ratio. Therefore, longer exposure time (or taking more exposures) allows us to observe objects with brightness lower than the minimum brightness observed with less time. In other words, as more or better exposures of a field are taken, objects of a given brightness at a greater distance can be observed. If objects are uniformly distributed in space, the number of objects detected at this brightness (or brighter) is proportional to the volume. So, to a first approximation, maximizing the number of objects detected is equivalent to maximize the volume survey.

With equations 3.4, 3.5, and 3.6 the volume survey can be calculated. The volume survey depends exclusively on the t_{eff} for our purposes; all other variables are just adjusting parameters that depend on the telescope, but for optimization purposes they behave as constants, scaling the results without changing the policy. The variable m_0 sets the zero point of the magnitude scale for a specific instrument. It maps the numbers of counts in pixels in an image to the brightness of objects on the sky. For a given instrument, m_0 is the magnitude of the faintest object that can be detected in an image with a t_{eff} of 1. The footprint_area was set at an angle of 3 in radians, this angle is approximately the solid angle

subtended by the field of view of DECam, the camera we are using for DES [7]. It may change as well with each facility. The constant abs_mag_limit is zero for this work.

$$\text{limiting\_magnitude} = m_0 - 1.25 \cdot log_{10}(t_{eff}) \tag{3.4}$$

$$\text{max\_distance} = 10^{(1+0.2 \cdot \text{abs\_mag\_limit} - \text{limiting\_magnitude})} \tag{3.5}$$

$$V_n[t_{eff}] = \frac{\text{max\_distance}}{3} \cdot \text{footprint\_area} \tag{3.6}$$

All the necessary elements that describe the reward have been defined. For V_n as the volume survey at step n and V_{n-1} as the volume survey at step $n-1$, the reward r_n at time n is defined by the following equation:

$$r_n = \triangle V_n[t_{eff}] = V_n - V_{n-1} \tag{3.7}$$

The change in the volume survey is used for calculating the reward as defined by equation 3.6, and equation 3.7. The Figure 3.1 shows the behavior of the reward while taking the same exposure continuously. Vertical axes are reward and the horizontal axes number of exposures for a target with the same θ_z (angle between the zenith and celestial body). Because θ_z is not changing, the airmass is not changing either. Notice slope on the obtained reward decreases with the number of exposures.

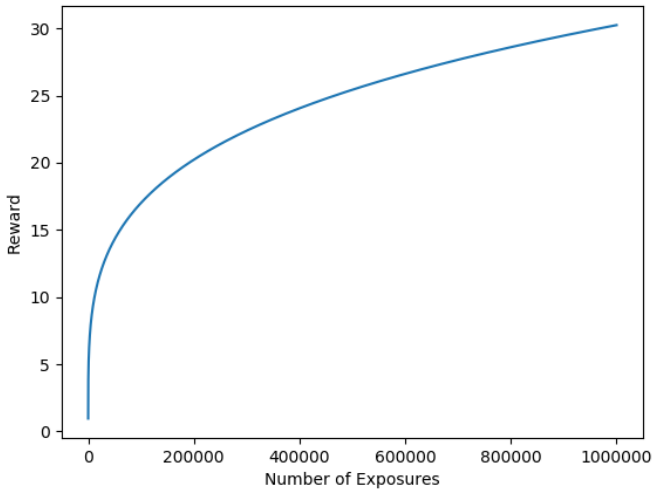

Figure 3.1: Reward as a function of number of exposures. This plot shows how the reward increases as a function of the number of exposures when the same exposure is taken several times. It can be observed that it is not linear even if the conditions of zenith angle, airmass, and seeing are the same.

3.1.2 State Model

The model of the state requires us to define three elements: time, state of t_{eff}, and current target.

- Time: The time is a floating number in modified Julian day (mjd) that is the integer assigned to a whole solar day in the Julian day count starting from noon, with Julian day number 0 assigned to the day starting at noon UT on Monday, January 1, 4713 BC.

- State of t_{eff} array: Every possible exposure in the survey period is going to be treated as a target. The set of all possible targets T conform to the action space of the agent. The accumulated t_{eff} (deep of field) for every target can be represented by a vector:

$$\vec{t}_{eff} = [t_{eff_0}, t_{eff_1}, ..., t_{eff_{m-1}}]'_{m\times 1} \tag{3.8}$$

  Each position j in the vector corresponds to a target and the value t_{eff_j} is the accumulated t_{eff} for the j target.

- Current target: The set of possible targets can be organized in a list; therefore, the current target is no more than the index in the list for the target of the current exposure, in other words, the place in the sky where the telescope is pointing.

Then the state $s \in S$ is given by $(mjd, S_{t_{eff}}, target_j)$, where $S_{t_{eff}}$ is the state of the vector $\vec{t}_{eff}$. Under this abstraction the variable mdj is continous; the vector $\vec{t}_{eff}$ contains t_{eff} values that are are continous; therefore, the number of possible states $s \in S$ is infinite as well as the number of $S_{t_{eff}}$. In order to use tabular methods it is necessary to have a limited number of states. By defining a number of mjd_{levels} and a number of $t_{eff_{levels}}$ the number of $s \in S$ is limited and discrete. Therefore, for a quantized mjd and a quantized $vect_{eff}$ vector, the state $s \in S$ is given by $s = (mjd, S_{t_{eff}}, target_j)$. If the number of $targets \in T$ (represented by a list) is given by card(T) where card(x) refers to cardinality of the set x, then the number of states is given by:

$$\text{card}(S_q) = (mjd_{levels}) \cdot (t_{eff_{levels}})^{\text{card}(T)} \cdot (\text{card}(T)) \tag{3.9}$$

In the case of sky survey problem using the tabular approach, either the value function or the value table can be represented by multidimensional array with shape-equal Shape_of_states, defined below:

$$t_{eff_{states}} = (t_{eff_{levels}})^{\text{card}(T)} \tag{3.10}$$

$$\text{Shape\_of\_states} = \text{tuple}(mjd_{levels}, t_{eff_{states}}, \text{card}(T)) \tag{3.11}$$

For the value function, each position in the array contains the value of being in that state. On the other hand, for the policy function, each position in the array contains the index of the next exposure to be taken, in other words, what action may be taken under an specific state. Depending on the quantization, $t_{eff_{levels}}$, and mdj_{levels}, the multidimensional arrays needed to store the value table and the policy can consume huge amounts of memory. Also the number of states given by the equation 3.9 may be very large, and the amount of time required to visit all the states using a methods such as policy iteration [12] [14] may grow exponentially. Similar to the representation of the policy and value function as tables described before, also the state-action pair Q function can be represented by a multidimensional array but the cardinality of this array is bigger. So, considering that you need to store the value of each action for a given state, the cardinality of the $Q(s, a)$ table is given by the equation 3.12:

$$\text{card}(Q(s, a)) = (mjd_{levels}) \cdot (t_{eff_{levels}})^{\text{card}(T)} \cdot (\text{card}(T))^2 \tag{3.12}$$

CHAPTER 4

IMPLEMENTATION DESCRIPTION

The key elements in reinforcement learning are the agent and the environment. The environment was designed as a class. Agents can instantiate objects of the environment class to use them during the learning. An additional script named get_preSet_env.py instantiates and returns an object of the environment using some pre-set parameters; this was useful for testing all the agents with the same parameters. For convenience, the reward function was separated from the environment and located in a module called reward_function.py. This modularity allows modifications in the reward function without changing the environment class. Other relevant tasks such as quantize observations are implemented also in modules.

4.1 Environment

The environment was implemented in the OpenAI Gym style [20]. Objects from a class called ObservingEnv() are used to implement the environment; several methods and members were defined to perform tasks such as calculate the t_{eff}, store the t_{eff} for every target, determine the observable targets for an mjd, or return the reward after an action is performed. The most relevant methods of an ObservingEnv() object are briefly described below:

- Step: Performs the actual step, changing the state of the environment as a consequence of the action received in the arguments. This function returns four data structures: observation, reward, done, and info. Observation is a Python dictionary that describes the future state. It contains a time object and the t_{eff} array. Reward is an integer that

gives the value of the reward obtained by performing the action. Done is a Boolean flag that indicates if there is still remaining time in the survey. Info is a dictionary that contains two Boolean flags: invalid action and time overflow.

- Reset: As its name indicates this method cleans all the variables and returns the environment to initial conditions defined in the object constructor.
- Render: This method shows the state of the environment. This means printing t_{eff} value for all the targets and the time in the survey.

4.2 Agents

Different agents such as one-step greedy agent, policy iteration agent, and Q learning agent were implemented by following the algorithm descriptions in 2.3. These agents use objects of the class ObservingEnv(). However, the states are continous and the number of possible states is infinite. Each observation received by the agent needs to be quantized. This step is fundamental in tabular methods because the Bellman equations are only applicable for finite and discrete states.

4.3 Modules and Diagnostic Scripts

For reducing implementation complexity and improving code maintenance, most of the methods were aggregated in modules. Methods defined at agent and environment are the only ones specifically related with the algorithm behavior or the abstract definition of environment. Functionalities such as store information in hd5 files, quantize or unquantize

observations, and test and plot the final policy are aggregated in different modules under the directory sky_sched_modules.

- Reward function: In this module the definitions volume_reward and calculate_reward are established to implement the model described in 3.1.1.
- Agent utils: Methods to change bases of a number, store hd5 files, quantize and unquantize values.
- Quantization utils: Group specific methods to quantize and unquantize observations of the environment; this module uses methods from the agent utils module.
- Test final policy: Aggregate all the methods that plot and test the reward of the final policy of the different agents.
- Target generator: This module contains methods that generate different sets of targets for testing.
- Get pre-set environment: A special module get_preSet_env.py is used to instantiate an environment and return an object of the environment. This module is an exception because it is the only script inside the module directory that has dependencies on scripts outside the module directory. This module was extensively use during the agent testing.

The initial testing shows that some routines were slow and impacted the performance of the code. Additional testing shows that these routines are not inherent to agent algorithms but instead the delay was related with calls to objects from the library "Astroplan." In order to speed up, a module developed in Cython implements functionalities provided by the Astroplan library. Cython is a compile language that generates a CPython extension module. This module can be imported in Python, increasing the performance. The process

is as follows: from the .pyx that is Cython sintax a .c file is generated. That file is compiled to generates a .so file in the case of Linux. This file is the one used when the module is called by another script. Three scripts are located in this module directory: vect.pyx, conversion.py and constraints.py, briefly described below.

- vect.pyx: This module is the actual core of the Cython modules; it defines how to calculate the angle separation between two objects, airmass, t_{eff}, and others. It also provides a vector implementation that allows us to apply the same operation to several elements in an array, doing quick operations over all the targets, for example, or over all the t_{eff} array. This vectorization is the key element to speed up the operations.

- conversion.py: This module basically transforms Astroplan objects into a different type of objects that can be operated faster. For example, objects such as the transitioner provide the functionality of determining the slew time, change of filters, and others to calculate the transition time from observing one target in the sky to another one.

- constraints.py: This determines if a target accomplishes all the defined constraints, such as: airmass, separation with the moon, and observable at a given time, among others.

Astroplan is an open-source Python package to help astronomers plan observations [21]. Astroplan is imported by some modules of the final implementation. Furthermore, some of the Astroplan definitions inspire methods and members used in the final implementation, specifically the implementation of observation blocks, targets, constraints, and observability. A separate module containing specific calls to Astroplan was created. Two scripts are located in this module: a script that uses Astroplan scheduler to create a schedule that may be compared against agent ones and another script that obtains the actual reward while following the Astroplan schedule with the given reward function.

A separated module called diagnostic scripts was created for evaluating the complexity of the implementation, evaluate the memory and computational limits, and stressing the reward function for evaluate its behaviour.

4.4 Evaluation of Implementation Complexity

The main problem of tabular methods is that computational and memory power requirements grow fast with size of the input. This is a common problem on NP-hard problems [3][1][12]. For the specific RL model described in Chapter 2, a brief exploration on the complexity for tabular methods is described below.

Three multidimensional arrays are necessary for the tabular representation. One array represents the $Q(s, a)$ state-action table, another array represents the policy function $\pi(s)$, and another represents the effective exposure time array $\vec{t}_{eff}$. The variables that affect the size of these multidimensional arrays are t_{eff} levels, time levels, and number of targets. In order to show the effect of the variables into the table sizes, we will fix two out of the three variables and explore the dependence due to the remaining variable.

In Figure 4.1 we explore the change in the multidimensional arrays size as a function of the number of time levels. The number of time levels does not affect the t_{eff} array, so it is not included in Figure 4.1.

In Figure 4.2 we explore the change in the multidimensional arrays size as a function of the number of t_{eff} levels.

In Figure 4.3 we explore the change in the multidimensional arrays size as a function of the number of targets.

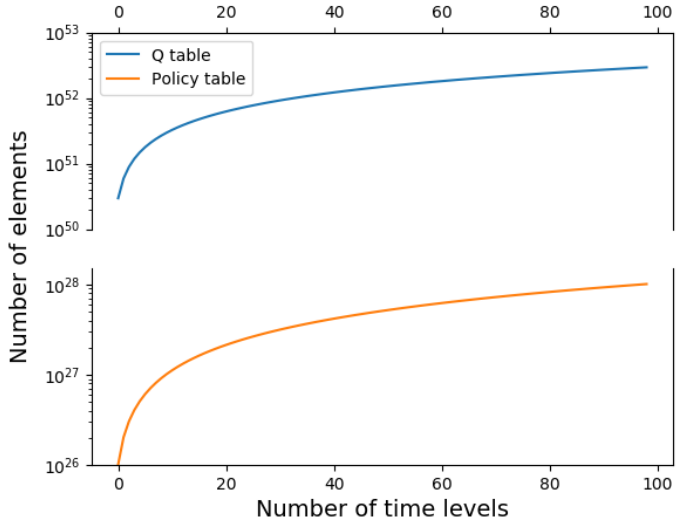

Figure 4.1: Size of tables as a function of number of time levels. The number of elements in $Q(s,a)$ (blue) and $\pi(s)$ (orange) grows at the same rate. However, the $Q(s,a)$ table contains one more dimension and therefore the number of elements is larger. The number of elements in the $\vec{t}_{eff}$ is not affected by the time levels and therefore is not presented in the plot.

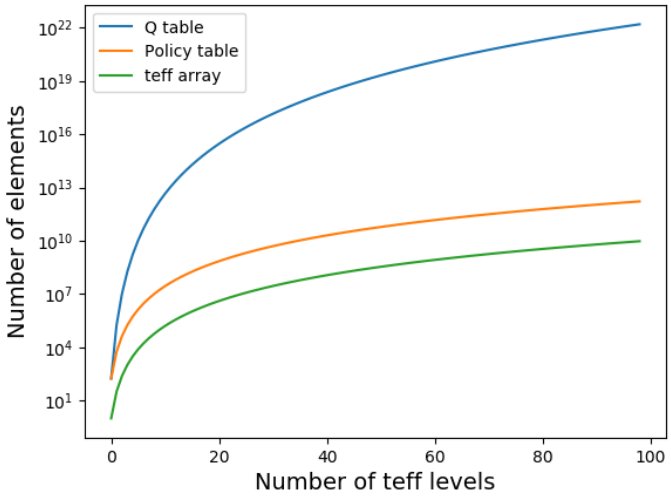

Figure 4.2: Size of tables as a function of number of t_{eff} levels. The number of elements in the $Q(s,a)$ (blue) grows faster than the number of elements in $\pi(s)$ (orange) and $\vec{t_{eff}}$ (green). The $\pi(s)$ table and the $\vec{t}_{eff}$ grow at the same rate but the number of elements in $\pi(s)$ is larger because it has more dimensions than $\vec{t}_{eff}$.

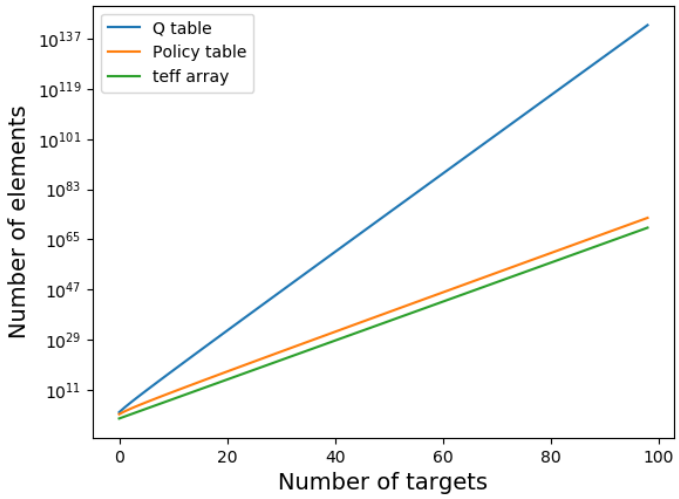

Figure 4.3: Size of tables as a function of number of the number of targets. The number of elements in the $Q(s, a)$ (blue) grows faster than the number of elements in $\pi(s)$ (orange) and $\vec{t_{eff}}$ (green). The $\pi(s)$ table and the $\vec{t_{eff}}$ grow at the same rate but the number of elements in $\pi(s)$ is larger because it has more dimensions than $\vec{t_{eff}}$. The size of the three arrays grows quickly as the number of targets increase.

The three multidimensional arrays grow faster with the number of targets than with the number of t_{eff} levels or the number of time levels. An actual survey may have around 8000 targets, so using tabular methods is not feasible because it would require huge amounts of memory to keep the multidimensional arrays.

A test was designed to experimentally explore limits due to the complexity using tabular methods. The parameters were two time discrete levels, two t_{eff}, and n number of targets. The Q table was generated by increasing the number targets n. In a machine with the following characteristics, Intel(R) Core(TM) i7-3537U CPU @ 2.00GHz and with 2GB of RAM, the maximum number of targets reached was $n = 18$; the limit was due to memory. In a significantly more powerful machine 4 CPU and 24GB of memory, the maximum number of targets reach was $n = 22$. Other options like distributing tables among machines or creating a data base were explored but those option would increase the processing time. We do not

consider there is a way to approach this problem through tabular representations; instead, we believe approximations methods with parametric representation may be a feasible solution for the complexity problem.

CHAPTER 5

TEST RESULTS AND ANALYSIS

In order to evaluate different algorithms, a set of tests was performed. In this chapter the reader can find details of each test, results, and some analysis and comparisons. The tabular representation approach was used during the test, constraining the number of targets due to the increase in the size of the tables, but this is not the only reason for using few targets while testing. Also, there is the difficulty of determining the best schedule by hand; therefore, the use of toy problems with a small number of targets allows us to manually check if the agent behavior is correct. Two toy problem configurations are presented in this chapter, one with two targets and another with three targets. The two-target configuration problem is solved with greedy agent and Q learning agent; the three-target configuration is solved with greedy agent, Q learning agent, and the traditional framework Astroplan.

Some tests were designed using Astroplan [21]. These tests work as checks against results obtained by the one-step greedy agent and the Q learning agent. In Astroplan the user may define Astroplan blocks (slots of times for exposures) and Astroplan targets (desire observations). If more Astroplan targets than available Astroplan blocks (slots of time for exposure) are provided, the software raises an error. If fewer Astroplan targets than available Astroplan blocks are provided, then it just does not schedule anything in the remaining Astroplan blocks. Therefore, the exact number of exposures per each target must be provided to Astroplan before the test.

Different parameters such as the location, date, and instrumentation impact the schedule. Below we describe in Table 5.1 some of the parameters used during tests. These parameters

are embedded into the environment class and are loaded through variables that can be tuned for different conditions.

Table 5.1: General test parameters

Parameter	**Values**
Telescope Location	Cerro Tololo
Date	2018-09-22
Exposure time in seconds	120
Read out time in seconds	30
Number of exposures per target	4
Instrument reconfiguration time in seconds	15
Slew Rate in degrees per second	2.2

The exposure time is the time in seconds of every exposure taken by the telescope when the agent selects a target; the read out time is the time spent by the CCD detector for reading the data of the exposure. Number of exposures correspond to how many exposures are taken when the agent selects a target. Slew rate is the angular velocity of the telescope while moving to point a new area in the sky.

The number of states and parameters that determine the number of states are presented in Table 5.2. In general, time t and t_{eff} can take real positive values such as $t \in \mathbb{R}^+$ and $t_{eff} \in \mathbb{R}^+$. Therefore, the number of possible states is infinite. In order to use tabular methods, every state $s \in S$ needs to be quantized as explained in Chapter 2. Parameters for quantization of each test are presented in Table 5.2.

Table 5.2: Quantization parameters in the test

Parameters	**First Test**	**Second Test**
Time levels	36	36
Targets	2	3
t_{eff} levels	5	5
$\vec{t}_{eff}$ array states	25	125
Number of states	1800	13500

The table presents the number of levels used for quantizing time, t_{eff}, and the number of targets.The $\vec{t}_{eff}$ array states is equal to the number of possible combinations that can have the $\vec{t}_{eff}$ given the quantization of t_{eff} and the number of targets. The last row in the table shows the number of possible states due to the quantization levels and the number of targets.

Q learning has some parameters that need to be manually tuned, such as learning rate, discount factor, and epsilon (exploration probability). The Q learning algorithm parameters used are described in Table 5.3.

Table 5.3: Hyperparameters for test

Parameter	**Values**
Learning rate	$\alpha = 0.4$
Discount factor	$\gamma = 0.99$
Exploration probability	$\epsilon = 0.005$

The learning rate α scales the temporal difference error in the updates of Q learning and it defines the step of the learning in the direction of the estimated error. The discount factor γ defines how much weight will be given to future expected rewards; γ is a value between 0 and 1. When $\gamma = 0$, only the one-step reward is considered and the future expected rewards are disregarded. The exploration probability ϵ defines that with probability ϵ a random action is taken to explore new actions in the action space for a given state.

5.1 First Test: Two Targets

A simple test with two possible targets was created to evaluate the behavior of agents, right ascension (RA) and declination (DEC), for each target presented below:

- Target A: RA=95° and DEC=0°
- Target B: RA=285° and DEC=0°

The coordinates of the targets were selected in such a way that when the observation start target B will be at the zenith point, target A will be below the horizon about to raise. At the end of the observation, target B will be setting in the horizon while target A will be very close to the zenith, as can be observed in Figure 5.1.

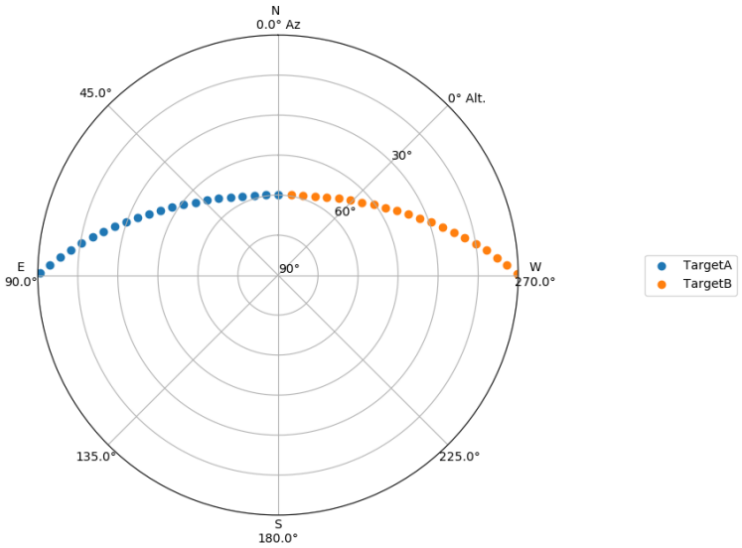

Figure 5.1: Trajectory of targets during observation. This plot describes the movement of the targets in the sky during the span of the survey. Target B (orange) is at the zenith at the beginning of the observation while target A (blue) will be at the horizon in the east at the beginning. Both targets move from east to west during the observation.

With only two targets at given locations described above, the greedy agent can converge to the optimal solution. This is because the problem is convex and the local optimal solution

is equal to the global optimal solution. Therefore, the trajectory given by one step greedy agent is the optimal trajectory and can be used for measuring the success of the other two agents. This test case can also be checked manually because the maximum reward is obtained by the target that is closer to the zenith. The problem was setting in that way for having a baseline and checking correctness of agents results. Therefore, the trajectory given by one step greedy agent is the optimal trajectory and can be used for measuring the success of the other two agents. The optimal trajectory is as follows: Start with 21 exposures of target B and afterwards 12 exposures of target A. Two additional exposures are taken by the greedy agent but they received a negative reward due to violation of airmass constraints; in other words, after 33 exposures the optimal trajectory is complete. The total accumulated reward of the agent was 49.64 after 35 actions (including the two last invalid actions). The final status of t_{eff} per target is presented in Table 5.4.

Table 5.4: Accumulated t_{eff} per target, two targets

name	A	B
t_{eff}	8.23	14.32

The table shows how much t_{eff} was accumulated per each target after the greedy agent selected the actions.

After performing learning for 1500 episodes, the total reward obtained was 45.28 (Figure 5.2. The optimal trajectory reward was 49.64. Therefore, the Q learning was able to obtain 91.22%.

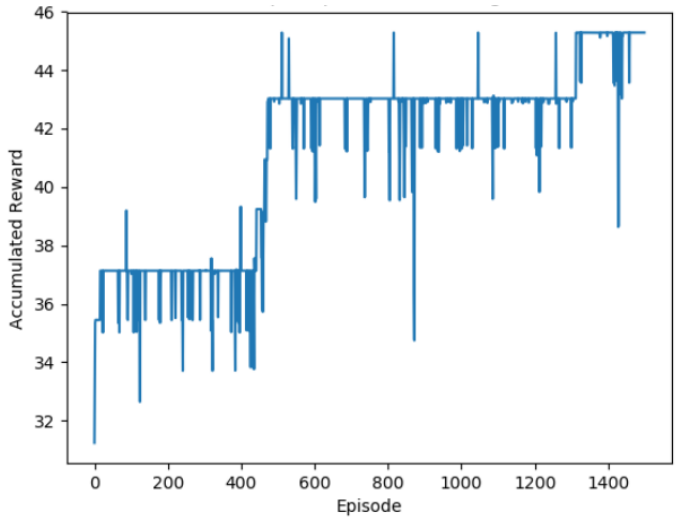

Figure 5.2: Reward per episode, Q learning, two targets, 1500 episodes. Each episode is a whole survey from beginning to end, choosing actions and accumulating rewards. So in this case the Q learning performed 1500 surveys of the given two-target problem.

This first test shows that the greedy agent is able to find the optimal local maximum and because of the location of the targets the global maximum was actually the same. The Q learning agent was able to find a trajectory close to the optimal trajectory after 1500 episodes. This test is important because the optimal trajectory can be found manually. The results provide insights that both algorithms were coded correctly and show promising results to scale into more targets and more difficult schedules.

5.2 Second Test: Three Targets

A standard test of three celestial targets was designed, and the performance of the three scheduling algorithms was compared. The targets A, B, and C were created with right ascension(RA) and declination (DEC) listed below:

- Target A: RA=95° and DEC=0°

- Target B: RA=10° and DEC=0°
- Target C: RA=285° and DEC=0°

For visualization purposes, a plot of the three targets at around the middle of the observation time is shown in Figure 5.3.

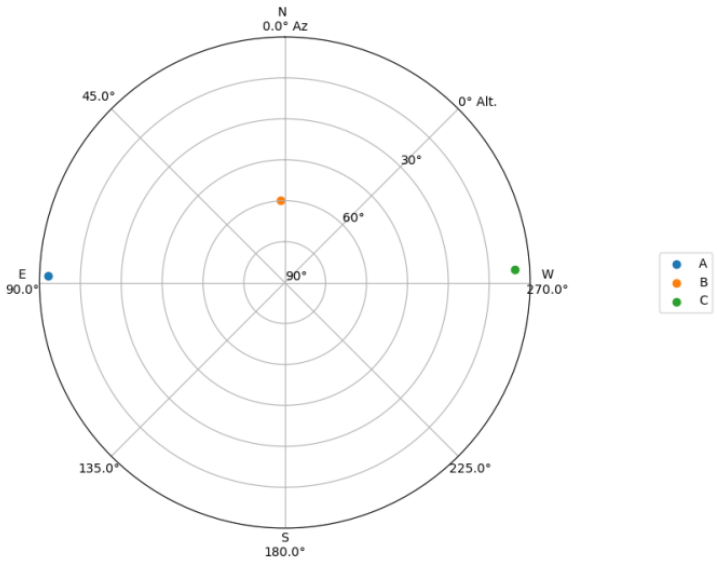

Figure 5.3: Position of targets at the middle of observation. With three targets in the sky, we show the position of each of those targets at the middle of the time span of the survey. At the middle of the survey one of the targets is in the zenith while the other two are in the horizon. All targets move from east to west.

A,B, and C are used as input to the one-step greedy agent defined in Chapter 2. The trajectory of the optimal solution is the following: 21 exposures of C, 34 exposures of B and 0 exposures of A, leaving some remaining time. During the remaining time, A is still up in the sky with enough time to take two exposures but violating the airmass constraint; therefore, these two exposures may not be considered in the optimal trajectory. The greedy agent has no other option than to pick a target when there is remaining time, so it just selected target A in the last two exposures, receiving a negative reward. The total accumulated reward of the algorithm was 96.73 and the t_{eff} by target is presented in Table 5.5.

Table 5.5: Accumulated t_{eff} per target

name	A	B	C
t_{eff}	1.465155	24.72246	11.837601

The table shows how much t_{eff} was accumulated per each targets after the greedy agent selected the actions.

The exact number of exposure per each target A, B, and C must be provided. Using the information from the greedy algorithm, the Astroplan targets were created as follows: 0 Astroplan targets of class A, 33 Astroplan targets of class B, and 16 Astroplan targets of class C. This approach requires previous knowledge of the number of exposures per celestial object in the optimal trajectory, being a big disadvantage of Astroplan. The restrictions are also a motivation to explore other approaches more suitable with sky survey scheduling. The Astroplan scheduler scheduled 45 exposures; however, it did not reach the optimal trajectory. In the optimal trajectory, 57 exposures may be scheduled. In this case Astroplan selected the following trajectory: 8 exposures of target C, 33 exposures of target B, and 2 exposures of target A, obtaining a total reward of 73.06 for a 75.53%.

Q learning algorithm can discover how many exposures should be taken on each one; the only inputs that Q learning needs are targets, three in this case. The execution time of this algorithm is longer than Astroplan but it does not require any previous knowledge of the optimal solution. Q learning agent obtained a solution space close to the one given by the greedy algorithm, and after 20000 episodes or iterations it outperformed greedy agent. The solution space of the Q learning agent was obtained by using the last Q table and greedy policy all the time as explained in the algorithm 4.

For 1500 episodes the final trajectory resulted in 67 actions, 19 of which actions were invalid actions. Therefore, the algorithm took 48 exposures. The total accumulated reward

in the last iteration was 68.48; the reward obtained by the greedy algorithm was 96.73. The learning process is shown in Figure 5.4. Due to the exploration-exploitation trade-off, we do not expect to converge to the best schedule inside the learning process but instead get an answer that is close enough.

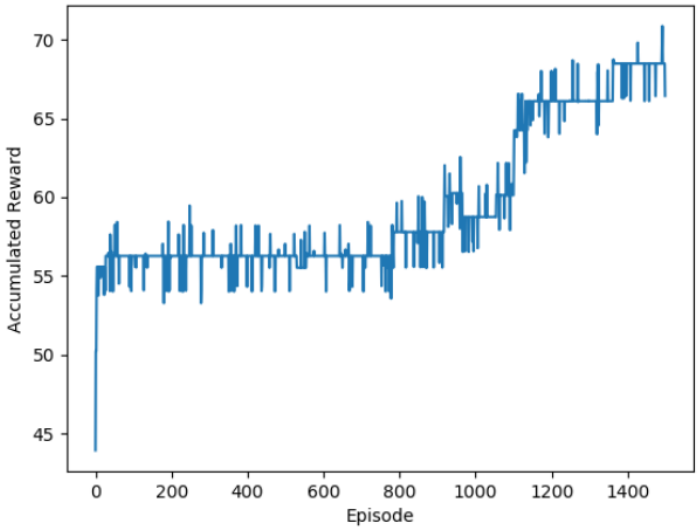

Figure 5.4: Reward per episode, Q learning, three targets, 1500 episodes.

A second test was performed increasing the number of episodes to 5000. With this condition, the accumulated reward improved from 68.48 to 87.28, in other words, from 70.79% to 90.79% of the score obtained by the greedy agent. The plot of the learning process for 5000 episodes is presented in Figure 5.5.

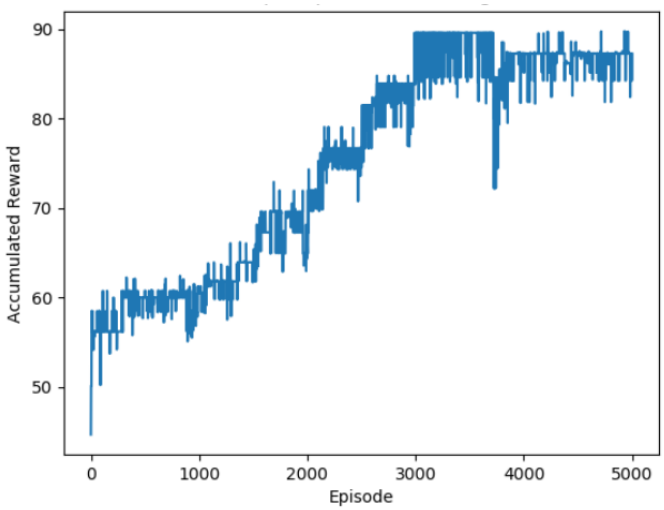

Figure 5.5: Reward per episode, Q learning, three targets, 5000 episodes.

Increasing the number of steps to 20000 results in an increment in the final reward to 97.24; this reward is 3% higher than the result obtained by the greedy algorithm. This trajectory in particular contains 56 actions, with only one invalid action, and the learning process can be observed in Figure 5.6. Increasing the number of episodes by other 20000 for a total of 40000 episodes does not increase the final reward of the Q learning agent. The learning process for 40000 episodes can be seen in Figure 5.7.

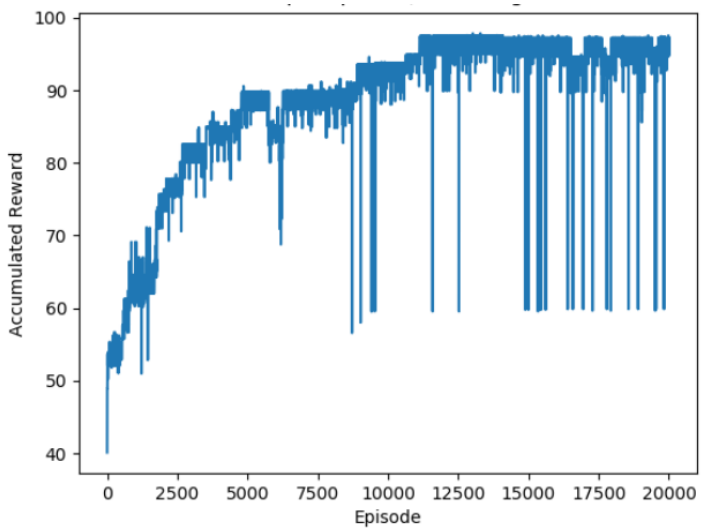

Figure 5.6: Reward per episode, Q learning, three targets, 20000 episodes.

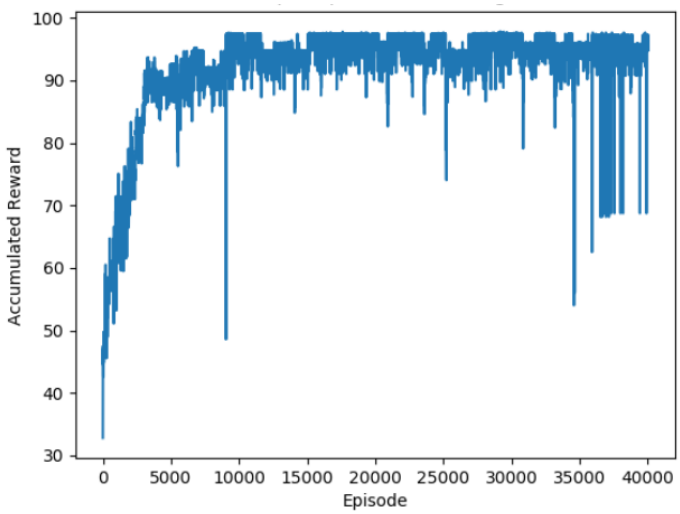

Figure 5.7: Reward per episode, Q learning, three targets, 40000 episodes.

A summary of the results can be observed in Table 5.6; this table compares the results of the Q learning agents against the greedy agent. The reward percentage row shows that after 20000 episodes Q learning outperformed the other agents. The Table 5.7 shows the final total reward accumulated by each agent; in this case we use the best result obtained by Q learning to plot in this table.

Table 5.6: Performance of Q learning with respect to the number of episodes

Description	**Results 1**	**Results 2**	**Results 3**	**Results 4**
Number of episodes	1500	5000	20000	40000
Number of actions	67	59	56	56
Number of valid actions	43	51	55	55
Number of invalid actions	24	8	1	1
Accumulated Reward	68.48	87.28	97.25	97.25
Q learning-reward/Greedy-reward	70.79%	90.79%	103%	103%
Algorithm run time in min	64	253	807	1485

In this table we summarize the Q learning algorithm performance as we increment the number of episodes. The number of valid actions are the ones when the agent selected a target that was visible in a given moment; the number of invalid actions are the ones when the agent selected a target that was not visible at that moment. With more episodes the algorithm is able to increase the accumulated reward. After 20000 episodes it performed better than the greedy agent. Having the greedy agent accumulated reward as a baseline, we compared in percentage of how much reward was obtained by the Q learning agent with respect to the greedy agent. The last row shows how with more episodes the running time of the algorithm increases incrementally.

Table 5.7: The accumulated reward per algorithm

Agent	Greedy	Astroplan	Q Learning
Accumulated Reward	96.73	73.06	97.24

The accumulated reward per algorithm shows that Q learning obtained the best results; in this comparison we considered the accumulated reward obtained by Q learning after 20000 episodes.

The result in Table 5.6 shows that looking for a local optimal solution is not always driven to the optimal global solution in non-convex problems. For this reason the Q learning agent outperformed the one-step greedy agent. The Q learning agent has a the disadvantage compared with the greedy agent: it requires more computational time to reach the solution. This time increases with the number of episodes, as is shown in Table 5.6.

5.3 Conclusions

We developed a model for the sky survey problem under the reinforcement learning (RL) paradigm. We presented a model of the states and the reward function based on the time effective t_{eff}. The toy problems and tests show that it is possible to use this model to make an agent learn the right schedule. This model can be used by applying different RL techniques. The state model can be extended to include other features as well as the reward function and can be modified to encourage a different behavior in the agents. This work sets the foundations for modeling sky survey problems under this paradigm.

We implemented a greedy agent and a Q learning agent and tested two toy problems. Also we used Astroplan for the three-target problem as a standard framework. The results show that if enough episodes (iterations) are allowed, Q learning can outperform greedy agents and standard frameworks. With this work we can also say that the greedy algorithm in combination with a control set of targets can be used for showing the proximity of the result policy with the optimal policy. The comparison between the greedy algorithm and the new algorithms is important to keep a reference with a method that converges in convex problems. Using Astroplan in a sky survey is not feasible because it requires information about the optimal policy. However, it is useful for comparison purposes. In this test, the

results of a conventional tool like Astroplan do not diverge from the ones obtained by the Q learning agent and the greedy agent.

The parameter epsilon (exploration) in Q learning has a big impact in algorithm convergence. From the testing was found that epsilon should be lower than 0.01 in order to reach convergence. The best result was obtained using epsilon=0.005. The initial values of the Q table also have an important impact on algorithm convergence of the Q learning algorithm. After some experimentation, the best results were obtained initializing the Q table with random values between 0 and 1. Negative values or higher values slow down the convergence in a significant way.

With the analysis of complexity in section 4.4 and the run times obtained in Table 5.6, we can observe that tabular methods are computationally expensive and are wholly insufficient for large-scale surveys with large numbers of targets.

Future work may use more sophisticated agents such as Asynchronous Actor Critic Network or Double Q Learning Network for scaling to the actual sky surveys. However, the environment can be used instantiating an object of the class environment without significant changes. Also, future approaches can build on top of our model, either extending the state model or modifying the reward function. The object-oriented and modular architecture of our code is convenient for future extensions.

REFERENCES

[1] A. I. Gomez de Castro1 and J.Yañez. Optimization of telescope scheduling algorithmic research and scientific policy. *Proc. SPIE 10704, Observatory Operations: Strategies, Processes, and Systems VII*, 403:537–367, 2003.

[2] Mauricio Solar Matias Mora. A survey on the dynamic scheduling problem in astronomical observations. In *Third IFIP TC12 International Conference on Artificial Intelligence (AI)*, Artificial Intelligence in Theory and Practice III, pages 111–120, September 2010.

[3] Ce Yu Jian Xiao Jizhou Sun1 Ming Zhu Yi Zhong Qi Luo, Laiping Zhao. Cost-efficient scheduling of fast observations. *Springer Science+Business*, 45:107–126, 2018.

[4] Glenn Miller and Mark Johnston. Long range science scheduling for hubble space telescope. *ELSEVIER, Telematics and Informatics*, 8:313–323, 1991.

[5] Allen R. Farris. Scheduling simulation in alma. In *Optimizing Scientific Return for Astronomy through Information Technologies*, volume 5493 of *SPIE Astronomical Telescopes + Instrumentation*, 2004.

[6] C. Bignell M. Clark J. Condon M. McCarty P. Marganian A. Shelton J. Braatz J. Harnett R. J. Maddalena M. Mello K. O'Neil, D. Balser and E. Sessoms. The gbt dynamic scheduling system: A new scheduling paradigm. In *Astronomical Data Analysis Software and Systems XVIII*, volume 411 of *ASP Conference Series*, 2009.

[7] Dark Energy Survey, December 2019. https://www.darkenergysurvey.org.

[8] Sloan Digital Sky Survey Collaboration. Sloan digital sky survey, December 2018. https://www.sdss.org/.

[9] Peter Martinez Ruby Van Rooyen, Deneys S. Maartens. Autonomous scheduling in astronomy. *Proc. SPIE 10704, Observatory Operations: Strategies, Processes, and Systems VII*, 10704, 2018.

[10] D. C. Morris T. Giblin J. Neff A. Klotz B. Gendre, N. B. Orange and P. Thierr. From a computer controlled telescope to a robotic observatory: the history of the virt. *arXiv Astrophysics e-print astro-ph*, July 2018.

[11] The LSST Corporation. Large synoptic survey telescope, December 2018. https://www.lsst.org/about.

[12] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge Massachusetts, second edition, 2018.

[13] Sergey Bartunov Petko Georgiev Alexander Sasha Vezhnevets Michelle Yeo Alireza Makhzani Heinrich Küttler John Agapiou Julian Schrittwieser John Quan Stephen Gaffney Stig Petersen Karen Simonyan Tom Schaul Hado van Hasselt David Silver Timothy Lillicrap Kevin Calderone Paul Keet Anthony Brunasso David Lawrence Anders Ekermo Jacob Repp Rodney Tsing Oriol Vinyals, Timo Ewalds. Starcraft ii: A new challenge for reinforcement learning. *ArXiv Computer Science, Artificial Intelligence*.

[14] Sudharsan Ravichandiran. *Hands-On Reinforcement Learning with Python*. Packts Publishing Ltd, June 2018.

[15] National Institute of Standard and Technology, February 2005. https://xlinux.nist.gov/dads/HTML/greedyalgo.html.

[16] Han Hoogeveen. Multicriteria scheduling. *European Journal of Operational Research*, 167:592–623, 2005.

[17] William Albert Hiltner. Astronomical techniques. *University of Chicago Press*, 184, 1962. LCCN: 62-9113 (BKS2).

[18] The Editors of Encyclopaedia Britannica, December 2018. https://www.britannica.com/science/seeing.

[19] Youssef Billawala1 Dan Potter R. F. Loewenstein Fred Mrozek John Bally, David Theil and James P. Lloyd. A hartmann differential image motion monitor (h-dimm) for atmospheric turbulence characterisation. *the Astronomical Society of Australia*, 13:22–27, 1995.

[20] OpenAI company, December 2018. https://gym.openai.com/.

[21] B. M. Morris, E. Tollerud, B. Sipőcz, C. Deil, S. T. Douglas, J. Berlanga Medina, K. Vyhmeister, T. R. Smith, S. Littlefair, A. M. Price-Whelan, W. T. Gee, and E. Jeschke. Astroplan: An open source observation planning package in python. 155:128, March 2018.