MARMARA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

DURAĞAN OLMAYAN ORTAMLARDA KONUM-ZAMAN ANALİZİ YAPILARAK PEKİŞTİRMELİ ÖĞRENME SAĞLAMAK

BURAK M. GÖNCÜ

YÜKSEK LİSANS TEZİ
Bilgisayar Mühendisliği
Anabilim Dalı
Bilgisayar Mühendisliği Programı

DANIŞMAN
Doç. Dr. M. Borahan TÜMER

İSTANBUL, 2017

MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES

REINFORCEMENT LEARNING IN NON-STATIONARY ENVIRONMENTS USING SPATIOTEMPORAL ANALYSIS

BURAK M. GÖNCÜ

MASTER THESIS
Department of Computer Engineering

Thesis Supervisor
Assoc. Prof. M. Borahan TÜMER

ISTANBUL, 2017

MARMARA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

Marmara Üniversitesi Fen Bilimleri Enstitüsü Yüksek Lisans Öğrencisi Burak Muhammed GÖNCÜ'nün "Reinforcement learning in non-stationary environments using spatiotemporal alanalysis" başlıklı tez çalışması, 14 Eylül 2017 tarihinde savunulmuş ve jüri üyeleri tarafından başarılı bulunmuştur.

Jüri Üyeleri

Doç. Dr. Borahan TÜMER (Danışman)
Marmara Üniversitesi ..... (İMZA).....

Prof.Dr. Çiğdem Eroğlu Erdem (Üye)
Marmara Üniversitesi ..... (İMZA).....

Prof. Dr. H. Levent Akın (Üye)
Boğaziçi Üniversitesi..... (İMZA).....

ONAY

Marmara Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun …… tarih ve …… sayılı kararı ile Burak Muhammed GÖNCÜ'nün Bilgisayar Mühendisliği Anabilim Dalı Bilgisayar Mühendisliği Programında Yüksek Lisans derecesi alması onanmıştır.

Fen Bilimleri Enstitüsü Müdürü
Prof. Dr. Bülent EKİCİ

MARMARA UNIVERSITY INSTITUTE FOR GRADUATE STUDIES IN PURE AND APPLIED SCIENCES

Burak Muhammed GÖNCÜ, a Master of Science student of Marmara University Institute for Graduate Studies in Pure and Applied Sciences, defended his thesis entitled "Reinforcement learning in non-stationary environments using spatiotemporal analysis", on September 14, 2017 and has been found to be satisfactory by the jury members.

Jury Members

Assoc. Dr. Borahan TÜMER (Advisor)
Marmara University ..... (SIGN).....

Prof.Dr. Çiğdem Eroğlu Erdem (Jury Member)
Marmara Üniversitesi ..... (SIGN).....

Prof. Dr. H. Levent Akın (Jury Member)
Boğaziçi Üniversitesi ..... (SIGN).....

APPROVAL

Marmara University Institute for Graduate Studies in Pure and Applied Sciences Executive Committee approves that Burak Muhammed GÖNCÜ be granted the degree of Master of Science in Department of Computer Engineering Program on …… . (Resolution no:…….).

Director of the Institute
Prof. Dr. Bülent EKİCİ

ÖNSÖZ

Bu çalışmanın gerçekleştirilmesinde, bilgilerini ve tecrübelerini benimle paylaşan, kendisine ne zaman danışsam bana kıymetli zamanını ayırıp sabırla ve büyük bir ilgiyle bana faydalı olabilmek için elinden geleni yapan Doç. Dr. Borahan Tümer'e teşekkürü bir borç biliyor ve şükranlarımı sunuyorum. Ayriyetten benim bu akademik süreçte bana destek çıkıp, her türlü yardımı benden esirgemeyen kıymetli bölüm başkanımız, Prof. Dr. Haluk Rahmi Topçuoğlu'ya sonsuz saygı ve teşekkürlerimi bir borç bilirim. Yine çalışmamda konu, kaynak, yöntem ve manevi açıdan bana sürekli yardımda bulunan ve gelecekteki hayatında çok daha başarılı olacağına inandığım kıymetli dostum ve meslektaşım Erdem Emekligil'e de sonsuz teşekkürlerimi sunarım. Ayrıca kıymetli zamanını benim bilgi ve birikim açısından büyümeme yardımcı olan Marmara Üniversitesi, Bilgisayar Mühendisliğindeki hocalarımada teşekkürü ve bir borç bilir ve şükranlarımı sunarım.

Teşekkürlerin az kalacağı İstanbul Teknik Üniversitesindeki hocalarımın da bana 4 yıllık üniversite hayatım boyunca kazandırdıkları her şey için ve beni gelecekte söz sahibi yapacak bilgilerle donattıkları için hepsine teker teker teşekkürlerimi sunuyorum ve son olarak çalışmamda desteğini ve bana olan güvenini benden esirgemeyen ve beni bu günlere sevgi ve saygı kelimelerinin anlamlarını bilecek şekilde yetiştirerek getiren ve benden hiçbir zaman desteğini esirgemeyen bu hayattaki en büyük şansım olan aileme sonsuz teşekkürler.

Eylül, 2017 **Burak Muhammed Göncü**

TABLE OF CONTENTS

ÖZET

DURAĞAN OLMAYAN ORTAMLARDA KONUM-ZAMAN ANALİZİ YAPILARAK ÖĞRENME SAĞLAMAK

Geleneksel pekiştirmeli öğrenme (PÖ) yöntemleri ortamın veya hedefin değişkenlik gösterdiği durumlarda öğrenme sağlayamamaktadırlar. Bunun sebebi, PÖ biriminin hali hazırda öğrenmiş olduğu ortamı sil baştan yeniden öğrenememesidir. Bu sorunu çözmek amacıyla yavaş değişen ortamlarda, PÖ biriminin en son yaptığı eylemi yapmasının teşvik edildiği sezgisel yaklaşımlar olsada [Sutton And Barto (1998), Chapter9, Example9.3, p236-238], bunlar PÖ birimiyle aynı hızda hareket eden hedefler için yeterince hızlı sonuç vermemektedirler. Bu yazıda, yukarda belirtilen hareketli hedefler ve rekabet ortamını olduğu durumlar için yeni bir yöntem tartışacağız. Bu sorunun çözümü için hedefin konum-zaman bilgisi kullanılarak hazırlanan Stokastik süreç, PÖ döngüsünde PÖ biriminin ödüllendirme mekanizmasına iliştirilip sorunun çözümü için modüler bir yaklaşım sağlamış olacağız. Ayriyetten bu çalışmamızda yöntemimizin uygulanabilirliği ve performansını farklı problemler ile ölçüp Atari Ms.Pacman oyunu ile değerlendireceğiz. Son olarak yazıda belirtilen yöntemin testleri başarıyla tamamlayıp, hedef noktalarının başaralı bir şekilde tahminini sağladığını ve gerekli stratejileri (pusu kurma, önünü kesme, hedefin amaçlarını anlama) uyguladığını görmüş olacağız.

Eylül, 2017 **Burak Muhammed Göncü**

ABSTRACT

REINFORCEMENT LEARNING IN NON-STATIONARY ENVIRONMENTS USING SPATIOTEMPORAL ANALYSIS

Traditional reinforcement learning (RL) approaches fail to learn a policy to attain a dynamic or non-stationary goal. The reason for this is that the RL agent cannot start learning the changed environment from scratch once it has converged to a policy before the environment has changed. While heuristic solutions where the RL agent is encouraged to use least recently attempted actions are successful for slowly changing environments [Sutton And Barto (1998), Chapter9, Example9.3, p236-238], they do not form a sufficiently fast solution to follow a non-stationary goal state that moves with the same velocity of the RL agent. In this paper, we will discuss a new approach to the problem where there is an adversarial relation present between the dynamic goal and the RL agent. To tackle this, the spatio-temporal information of the dynamic goal state is incorporated, in terms of stochastic processes, as the rewards of the RL agent into the environment model thus enabling a modular solution to the problem. In addition, in this paper we present the method's robustness using different mazes where we assess the performance of our method and also test our algorithm with the Atari Ms.Pacman game for some complex problem solving. Finally, the results of the experiments show that our method successfully predicts the rival agent's behavior and points of interest in which the rival agent will pass through and ambush it at key positions.

September, 2017 **Burak Muhammed Göncü**

SYMBOLS

s : State *s* at *a* time instance *t*

s′ : The state transitioned by taking action *a*

a : Action taken by the agent according to its policy π

r : Reward/feedback returned by the environment by taking action *a* at state *s*

$\boldsymbol{Q^*(s,a)}$: Optimal value state action function (Bellman optimality principle)

$\boldsymbol{Q(s,a)}$: Value function of the state action pair

$\boldsymbol{\gamma}$: Discount rate

$\boldsymbol{\alpha}$: Learning rate

$\boldsymbol{\pi(s,a)}$: Probability of taking action *a* at state *s* under the policy π

f(s, t) : Number of visits to state *s* at time *t*

c(s, t) : Number of catches at state *s* at time *t*

P(s, t) : Probability of dynamic goal being at *s(t)* at time *t*

ABBREVIATIONS

RL **:** Reinforcement Learning

ANN **:** Artificial Neural Network

STRL **:** Spatio Temporal Reinforcement Learning

TD **:** Temporal Difference

CD **:** Context Detection

HRL **:** Hierarchical RL

DG **:** Distributed Generation

MC **:** Monte Carlo

SARSA **:** State Action Reward, next State next Action

RG **:** Residual Gradient

MDP **:** Markov Decision Processes

PS **:** Prioritized Sweeping

DQN **:** Deep Q-Network

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Animals learn their environment by taking actions that lead to some consequences. These consequences, in turn, contribute to their experience for learning their environment and behaving as necessary for getting as high a chance to stay alive as possible. As in nature, we use the similar principles to compute possibly optimal policies for reinforcement learning (RL) problems. RL foundation is a generic method to learn and optimize a behavior in an arbitrary environment. The problem of finding a near-optimal policy for a stationary environment is well dealt with by classical RL algorithms such as the TD learning algorithm by Sutton (1988), SARSA algorithm by Rummery and Niranjan (1994) and the Q learning algorithm by Barto et al. (1989). These algorithms generally envision the environment by a goal state which upon the visit of the agent will maximize the average total reward of the goal state and by states that connect the start state to the goal state. While classical RL algorithms provide satisfactory solutions for the stationary environment problem, they fail in non-stationary environments which either change the environment, the goal state or both at some time instances *t*. Some attempts to solve this problem have been made such as the DG learning algorithm by Kaelbling (1993) where the values are not a function of state action pairs but of a triplet of *(s, a, g)* where the goal is considered to be dynamic. In this study, we present a method where an RL agent learns to attain a dynamic goal in a grid world environment by making use of the spatio-temporal position of the dynamic goal. The spatio-temporal position information of the dynamic goal is represented by a stochastic process [Doob (1953)] defining the probability *p(s(t), t)* that the dynamic goal is at some state *s(t)* at some time instance *t*. The spatio-temporal position information is available to the RL agent in terms of rewards/reinforcements provided whenever the agent reaches the dynamic goal during its learning phase. Also because of the stochastic process our approach is superior to the DG learning algorithm since our approach also takes the changing environment into account using the stochastic process data. The novelty of our approach is that it

incorporates spatio-temporal information of the dynamic goal in terms of stochastic processes so as to adjust the rewarding mechanism. The rewards are modified so that the agent can learn how to attain a dynamic goal following a specific policy. As a result, the agent learns the policy of a dynamic goal thus also learns the patterns and behaviors inherently while also being robust. Finally, we will see that our method supports various learning algorithms that can learn from a similar model such as SARSA, Q learning and TD learning. The rest of the paper is planned as follows: in section 2, a brief discussion is given on the most relevant previous work on reinforcement learning at non-stationary environments. In section 3, we discuss our method in detail. First, the learning algorithm is specified. Then we state how the spatio-temporal information is represented as a stochastic process and is incorporated to the RL paradigm as a component of rewards. In the last subsection, a variant of the work where the RL agent and dynamic goal have different execution times of their actions is provided. In section 4, we first indicate the test environments we use to conduct the experiments for the basic environment, the environment where there is a different execution time for the dynamic goal and the RL agent and finally we will compare the algorithm with Deep Q-Networks using the MS.Pacman environment. We then proceed with the experimentation process and finally share and discuss the results of the experiments for both situations. Finally, we conclude and communicate a possible set of following works in sections 5 and 6, respectively. From here on we will name the spatio-temporal RL algorithm as STRL for convenience.

2. RELATED WORK

Some efforts were made to deal with non-stationary environments which the traditional algorithms fail to learn. One such solution is the RL-CD algorithm by Da Silva et al. (2006) which aims to overcome this problem using predicted partial models of the environment. These partial models are evaluated with a quality variable, which is a value that determines how well this environment fits this model of the environment. For each action, the model with the highest quality is selected to execute, when there is no other model that passes the minimum quality threshold is present, a new model will be created thus the context changes are detected when the model qualities fall below the minimum quality threshold. An extension of the RL-CD which further improves on this by incorporating the hierarchical RL is the HRL-CD algorithm by Yücesoy and Tümer (2015) which attempts to detect context switches in a periodically changing environment employing hierarchical RL (HRL) where during the search for context switches as in RL-CD algorithm HRL-CD agent also builds up a partial model for each context detected and looks for sub goals in these models to form options and use them to speed up the learning especially in problems with large state spaces. HRL-CD's option based policies stem from the SMDPs by Sutton et.al., in which the policy consists of options where each option consists of several actions. As such the general problem is broken down into several smaller problems and the general strategy is being carried on over the options. Both RL-CD and HRL-CD view the change as discrete regimes and thus prepare or use an existing model for each context detection. Since both of these algorithms generate a model for each context, their complexity grows as the number of contexts increases. To solve this STRL rather than predicting contexts, predicts the general behavior/pattern of the change in the goal. Furthermore, both of these algorithms only consider the change of the environment model such as shifting obstacles or changing actions, they do not take the effect of a moving or changing goal state into account which is supported by STRL. Due to these two reasons, RL-CD and HRL-CD do not apply to our problem of catching a goal that moves continuously and within some order. The DG learning algorithm by Kaelbling

(1993) takes the non-stationary goal state problem into account but since it generates a value table for each *(s,a,g)* triplet for each new goal state, it has excessive memory requirements and further in case the goal moves not randomly but within a specific order it does not exploit this order. Since STRL uses stochastic data to predict the direction of goal's motion (i.e., where it is moving) and incorporate this into the RL algorithm, it can efficiently find the goal state and further ambush it if necessary. Faußer and Schwenker (2011) presents a solution for the problem of learning complex problems in short amounts of time by using multiple agents acting as one and an ensemble of functions to construct the policy such as TD and RG update formulas. They use a modified version of the TD algorithm which uses parametrized state value functions instead of the classical state value table. Their approach is to predict the parameters of the functions of the model thus learning the problems much faster and to provide a good solution in a large MDP. But even the approach of function approximation and by using multiple model functions, they are still prone to highly non-stationary environments where the function is often times too complex to get a good fitting policy. Śnieżyński (2009) approached the predator-prey problem by using rule induction and multiple agents in one environment in which they transfer knowledge, sensory data and new rules for certain situations. The rules are generated using the AQ algorithm developed by Gehrke and Wojtusiak (2008). Using these rules, the problem is solved as follows. The processing module is responsible for keeping the basic learning process by managing the training data to learn new knowledge and to receive the environment's responses to the actions taken by this system from the environment. After the necessary state and reward values are received from the environment, the processing module will either execute a new action or learn new knowledge which is needed to solve the problem represented by the current state moved at by the environment. The processing module describes and transmits the current problem needed to be learned to the learning module. In turn the learning module will generate a near-optimal solution to the problem formulated by the processing module using the previously generated knowledge. After that the processing module will use the

newly gained knowledge to solve the problem and will determine whether or not to store this new knowledge in the generated knowledge bin. While this approach is good at finding abstract rules to problems it requires multiple agents in an environment and provided training data to solve the problems and devise rules for it. On the other hand, there is a work done by Doya et al. (2002) which solves a hunting problem in order to demonstrate his approach. Their approach views the nonstationary problem solution as a mixture of experts, in which the experts represent the different models of the current environment model. Kenji Doya et.al. utilize these models by using a soft-max selection method to select the appropriate expert modules. Furthermore, recent studies in Deep learning as the DQN (Deep Q-Network) algorithm by Mnih et al. (2015) promises near human level solutions to complex high-dimensional learning problems such as playing atari games which requires the agent to succeed in a wide variety of tasks.

3. METHODOLOGY

3.1. Preliminaries

Reinforcement learning (RL) is a behavioral learning paradigm where a learning actor, the agent, learns how to behave from the accumulation of immediate rewards received as a response to each action [Kaelbling et al. (1996); Sutton and Barto (1998)]. RL can be modeled as a Markov Decision Process (MDP), a sequential, discrete time, decision making framework specified by four variables (*s, a, s', r*) where $s \in S$ and $s' \in S \cup \{goal\}$ are the current and next states of the environment, *a* is the current action the agent executes and $r \in R$ is the immediate reward provided to the agent in return to its current action. Set up in discrete time (but can be generalized to continuous time), the agent executes an action a_t at the current state s_t and receives an immediate reward r_{t+1} in return to a_t and observes the next state s_{t+1} the environment moves to. At each step, the agent selects an action among possible actions by taking account of probabilities from policy, π. The agent applies the policy to determine which action *a* to be chosen at state *s*. In other words, the policy $\pi(s,a)$ denotes the probability of taking action *a* at state *s* under policy π. The agent selects actions at a state *s* with the purpose to maximize the expected total reward.

$$Q(s,a) = E_\pi[R_t|s_t = s,\ a_t = a\ ] =\ E_\pi[\sum_{k=0}^{\infty} \gamma^k\, r_{t+k+1}\ |s_t = s,\ a_t = a] \quad (3.1)$$

where γ is the discount rate, $0 \leq \gamma \leq 1$.

The expected total reward for an action *a* at some state *s* is defined as the value $Q(s,a)$ of that state-action pair. The agent's goal is to find the best policy that maximizes the expected total reward or an optimal policy. Optimal policies are characterized by the maximum or optimal action value. They are defined as in equation 3.2:

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) \quad (3.2)$$

Further an action with the optimal value must move the environment to a state

with a maximum expected return [Sutton and Barto (1998)]: This is the so-called Bellman optimality principle. The Bellman optimality equation is given in equation 3.3:

$$Q^*(s, a) = E[r_{t+1} + \gamma \max_b Q^*(s_{t+1}, b) \,|s_t = s,\ a_t = a] \qquad (3.3)$$

In the context of RL, an environment model is defined by the transition probability $P^a_{s,s'}$ and return $R^a_{s,s'}$ for each possible triplet $\langle s, s', a \rangle$ of source state, destination state and action. If the environment model is perfect, i.e., the pair ($P^a_{s,s'}$, $R^a_{s,s'}$) is available for each $\langle s, s', a \rangle$ then finding the optimal policy is a matter of value iteration and does not require learning. This can be solved using dynamic programming. If, on the other hand, the model is not perfectly available or perfect but it is extremely complex to obtain the optimal/near-optimal policy through dynamic programming based methods then learning is required to find a satisfactory (optimal, near-optimal or one that is sufficient for the given problem) policy. In this case basic techniques such as *Monte Carlo* (MC) and *Temporal Difference* (TD) are available for model-free RL and *prioritized sweeping* (PS) is available as one model-based RL technique among many others in both classes of techniques.

3.2. The Stochastic Process

We will use the analogy of a cat and mouse in a maze to represent the problem, which we devised to effectively test and represent this problem. We are attempting to solve in this thesis the problem of the cat and mouse, but as we mentioned before this method can be applied to any RL setting. We assume that the mouse has its path to the cheese as a predefined path in the maze. Hence the mouse will represent the dynamic goal of our problem. The cat (that is oblivious of the mouse's start position, goal position and the maze) will learn to catch the mouse using the reinforcement provided from the rewards that are adjusted using the spatiotemporal stochastic process. In this setup, the cat fills up two separate spatio-temporal tables below:

$$f(s,t) \rightarrow number\ of\ cat\ visits\ at\ (s,t) \tag{3.4}$$

$$c(s,t) \rightarrow number\ of\ catches\ at\ (s,t) \tag{3.5}$$

where $f(s,t)$ and $c(s,t)$ denote, respectively, the number of times the cat has been at state *s* at time *t* and the cat has caught the mouse at state *s* at time *t*. Using these two tables the agent forms a spatio-temporal distribution model for the mouse used to estimate the other states probabilities at time *t*. The probability for each state *s* at time *t* is calculated as follows:

$$P(s,t) = \frac{c(s,t)}{f(s,t)} \tag{3.6}$$

These will produce a bi-variate distribution for each time instance (or step number) *t* thus creating a three-dimensional distribution (a.k.a. spatio-temporal) data [Doob (1953)] in the model supplied as a reinforcement to the agent. Because the model is learned over time the cat gradually produces a better policy as long as the distribution model is near optimal values. This enables the agent to even handle situations where the environment also changes. This is due to the fact that the reinforcement that the agent takes is a combination of the environment reward and stochastic process.

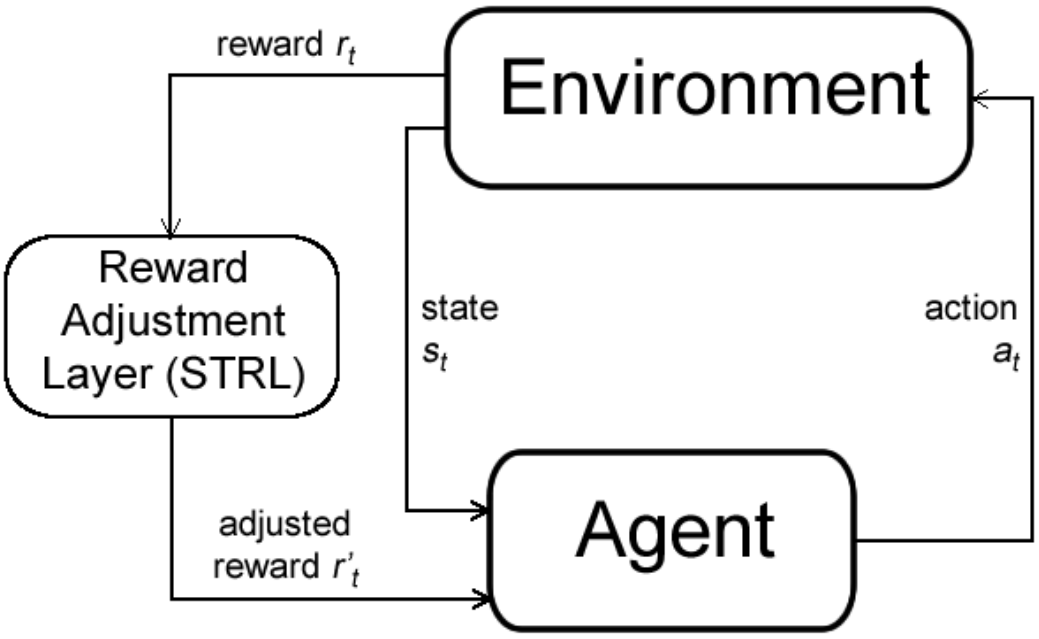


Figure 3.1 The STRL Loop

The STRL algorithm functions by injecting itself in between the reward mechanism between the agent and the environment. This layer contains the spatio-temporal data which is used to adjust the reward.

As can be seen in Figure 3.2, the cat caught the mouse around C1 and C2 in a time instance t. After some finite time has passed the cat gathers enough data for the model so that the model coefficients can be correctly predicted. Consequently, the correlation variables can be predicted (represented as the blue lines) provided sufficient time is allowed for learning variables μ, σ_1, σ_2 of the model. Thus, using these as rewards incorporated into the model gives the cat a clear goal towards the mean of the model which acts as an artificial goal at a time instance *t*.

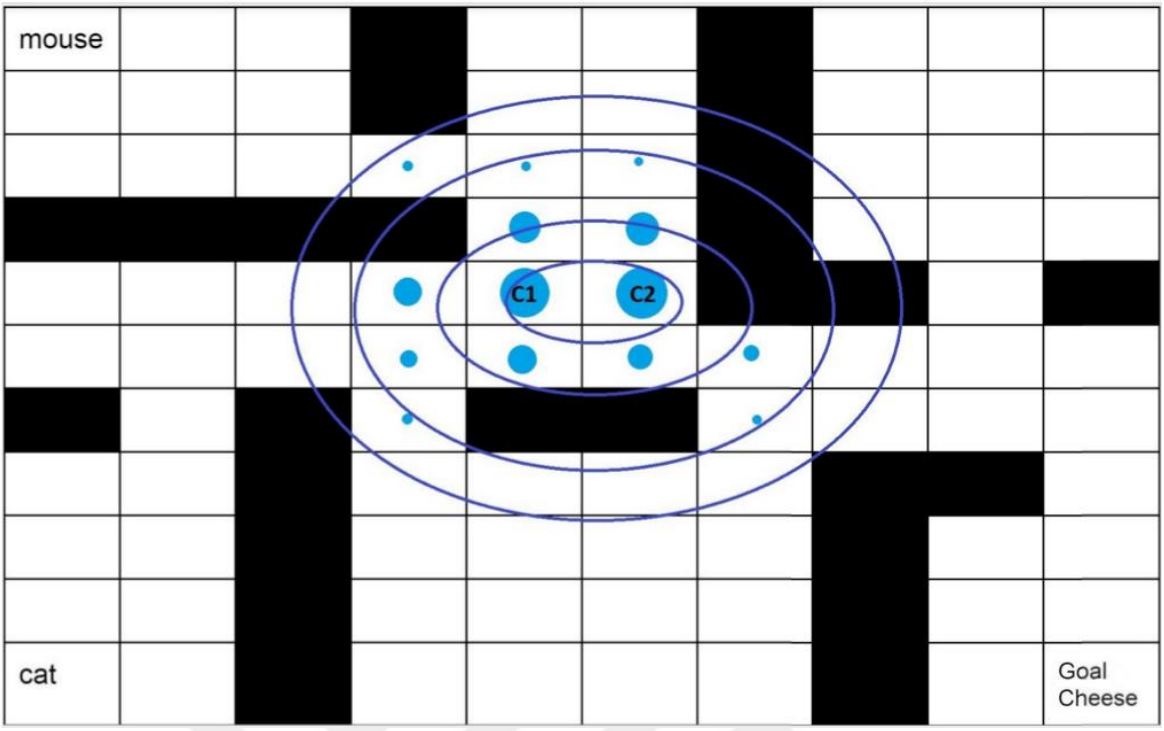


Figure 3.2 Sample Stochastic process view

This figure shows the frequency distribution of the catches of the cat sampled from the stochastic process. Here the cat caught the mouse mainly at *C1* and *C2* at an arbitrary time instance *t*.

To better understand the situation let us assume the case in Figure 3.3, where we represent the agent's state in some arbitrary state S^i. The agent has the option to choose from various actions ranging from a_1 to a_K. Assuming the agent took the action a_1 to traverse to state S^j, it will get an adjusted reward of $R^{a_1}_{s^i s^j} * P(s^j, t)$ thus actively adjusting the environment's reward to accommodate the non-stationary patterns of the goal state. As seen in Eqn.3.8 the stochastic process $P(S_{s^j}, j)$ will adjust the environment reward at every action taken. Thus, we will maximize the expected total reward of a problem where the goal is non-stationary.

$$V^{\pi}(s^i) = E_{\pi}[R_t * P(s_t, t)|s_t = s] =$$

$$= E_{\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} * P(s_{t+k+1}, t+k+1)|s_t = s,\ a_t = a] =$$

$$= \sum_a \pi(s^i, a) \sum_j [R^a_{s^i s^j} * P(S_{s^j}, j) + \gamma V^{\pi}(s^j)] \qquad (3.7)$$

where $\boldsymbol{\gamma}$ is the discount rate, $0 \leq \boldsymbol{\gamma} \leq 1$ and the stochastic process $P(S_{s^j}, j)$ is sufficiently constructed with sampled values during training.

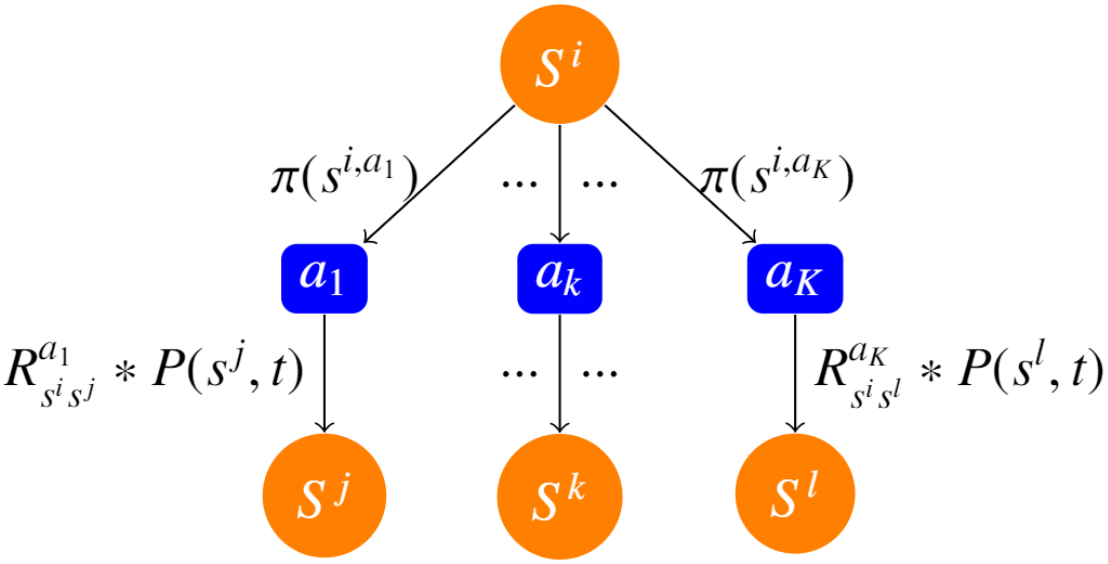


Figure 3.3 STRL backup diagram

The backup diagram of STRL providing the Recursive Value Functions the necessary feedback signal adjustment from state $\boldsymbol{S^i}$ to some arbitrary state $\boldsymbol{S}$ by taking some arbitrary action $\boldsymbol{a}$ (Assuming a stationary environment with transition probabilities as 1).

3.3. Learning Algorithm

The cat performs an analysis on the general behavior of the mouse by sampling the position-time (x, y, t) data of the catch point. Using these data, the cat gradually trains a distribution of the spatio-temporal behavior of the mouse. The distribution model follows a stochastic process [Doob (1953)]. Thus, the cat incorporates the stochastic spatio-temporal or position time data of the mouse. We also expect that different step sizes (speeds) of the agent will affect the means of these distributions at various time instances. The cat integrates this distribution model into its learning process to reinforce its state-action values.

Algorithm 1 General algorithm

Initialize Q(s, a) arbitrarily.
For each episode:
Repeat:
Initialize s.
Choose a from s using policy derived from Q.
Take action a and observe r and s'.
$P(s,t) \Leftarrow c(s,t) \,/\, f(s,t)$
$r \Leftarrow P(s,t) * r$
$Q(s,a) \Leftarrow Q(s,a) + \alpha * [r + \gamma * Q(s',a') - Q(s,a)]$
$s \Leftarrow s'$
$t \Leftarrow t + 1$
$f(s,t) \Leftarrow f(s,t) + 1$
If Cat catches mouse **then**
Train the stochastic process model with (x, y, t) as training data
c(s, t)⇐c(s, t) + 1
Terminate episode
endif
until Reached *MaxStepInEpisode* is true or terminal state reached

---

We can implement the additional data from the stochastic process into the environment model and combine it with the rewards to move the cat towards the mouse by simply providing adjusted rewards (original environment reward multiplied by the weight of the stochastic process) regarding the distribution model as follows;

$$AdjustedRewards = P(s,t) \times EnvironmentReward \quad (3.8)$$

Since the method incorporates the stochastic process as part of the reward of the environment it supports several classical algorithms such as SARSA, Q learning and TD learning. A goal state that moves by a specific policy (i.e., a mouse running for a piece of cheese) makes the relevant environment non-stationary. By using an approach as defined above we could integrate the probability data from our stochastic process into the rewards of the model, thus separating the stochastic process logic from the classic RL algorithm and incorporating the stochastic process logic making its contribution as a part of immediate rewards. In our example, we will use a modified Q learning algorithm Rummery and Niranjan (1994) as the learning method of our cat as shown in Algorithm1, that incorporates the reinforcement from the stochastic process.

4. RESULTS AND DISCUSSION

4.1. Cat and Mouse test environment

To test our method, we created an implementation of the problem using the dotRL framework [Papis and Wawrzynski (2013)]. We created several environments where the mazes have several *choke points*. By choke points we mean *bottlenecks* which, in case the environment is represented as a graph, would have higher values of centrality than other points in the graph. There are also variants of each maze where the agents start at different positions to better understand the behavior of the agents in cases where the cat is near the mouse start state, near the goal state or in equal distance to both of them. The main components of our problem are:

Mouse : The dynamic goal that has a defined course from the start point to the cheese.
Cheese : The goal/end point for the mouse.
Cat : The RL agent which attempts to catch the mouse, which has a defined course to the cheese.

In this paper, the basic environment of a grid world problem is modified to support two agents and also to provide the foundations of a stochastic process based on the rewarding scheme. The stochastic process is built with the catch point information and visit frequencies. The mazes are constructed as a part of the data generation considering the advantage and disadvantage of the agents' position to each other. Each maze situation is used to manifest how the algorithm performs depending upon the different initial positions of the agents. An additional maze, which tests the speed of the algorithm, is also created for testing. The STRL algorithm will be compared against the Q-learning algorithm to provide some basis of the performance.

4.2. Cat and Mouse test results

To have a baseline of test results we prepare a test environment where we create a grid world with no obstacles, in which the mouse starts with a predefined course which

will lead it to the cheese. The mouse begins at the start state (0,0) and moves by the predefined course towards the goal (10,10). In the meantime, the cat moves in the grid world using a random walk policy to catch the mouse. To log the instances of a catch the cat logs the states (the memory of the cat) where the cat and the mouse coincide on the same state (x, y).

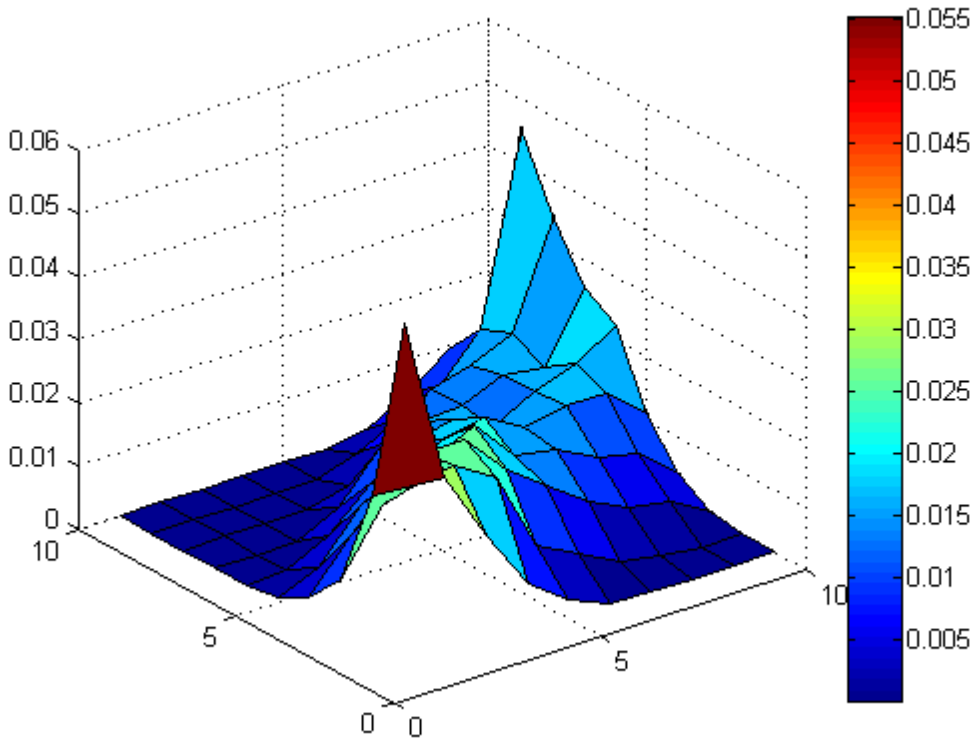


Figure 4.1 Distribution of a random walk

The frequency of catches in the grid world (frequency, x, y). In a 10.000-episode simulation.

In Figure 4.1 even in a time independent stochastic environment (cat is a random walk agent in this experiment) we can clearly observe that the catches tend to accumulate in the choke points (points in the environment that force the mouse to go through, can be tunnels, doorways, spawn and goal) which are in this case the start and the goal state of the mouse.

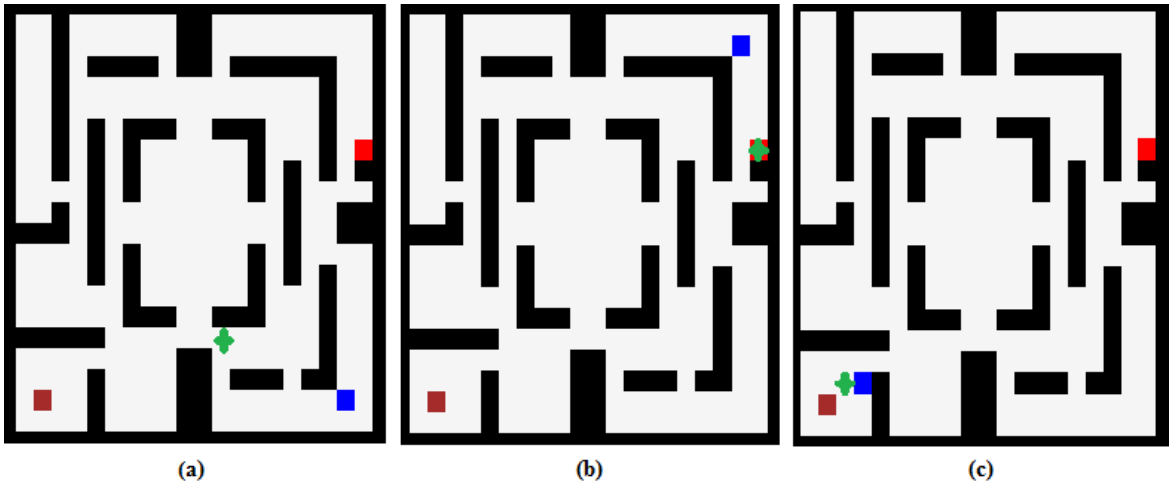
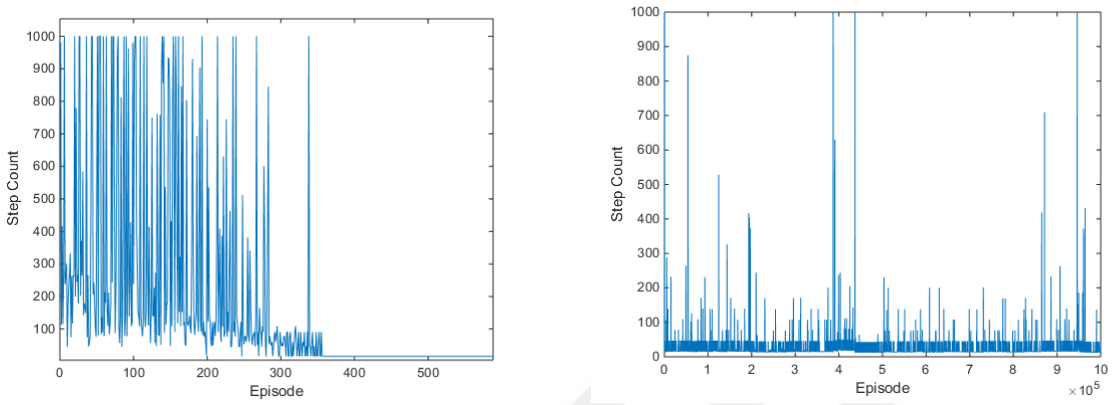


Figure 4.2 Generic test maze (Maze 1)

[Brown square: mouse, Blue square: cat, Red Square: cheese. Green Cross: Cats ambush point] (a) the cat is in the middle of the cheese and spawn point of the mouse, which will result the cat to move towards the mouse and ambush it near the tunnel at the middle-bottom of the maze thus testing the general case/use of the STRL algorithm. (b)the cat is near the goal of the mouse which, in turn, results in the cat to figure out that this choke point is the goal of the mouse, thus trying to measure whether the STRL can detect points of interests. (c) we measure whether the cat can catch/ambush the mouse in a limited time frame (tight policy, no room for errors for the cat or it will wait for a full mouse policy cycle) thus measuring its learning rate.

Moving onwards from these results we tested our approach in a maze which will have three variations in which the start point of the cat is in equal distance to both the start point of the mouse and the cheese, near the goal state of the mouse and near the mouse start position, respectively, as shown in Figure 4.2*(a), (b)* and *(c)*. The parameters used for the experiments are as follows: $\varepsilon = 0.1$ (with a 0.0001 reduction after each episode and a minimum boundary as 0.001) are selected to encourage exploration and to build the spatio-temporal tables faster in the beginning, $\gamma = 0.95$ as we want to have the current value heavily depend on the future actions' values and $\alpha = 0.1$ to reach the optimal policy faster in stable means. The spatio-temporal tables; *f(s, t)* and *c(s, t)* are initialized with a zero value to form an empty frequency data. These values were based on the grid

world parameters suggested in Sutton and Barto (1998). The aim of this test is to find out how well our algorithm detects points of interest and points of high centrality values (a.k.a. choke points).



(a) STRL (b) Q-Learning

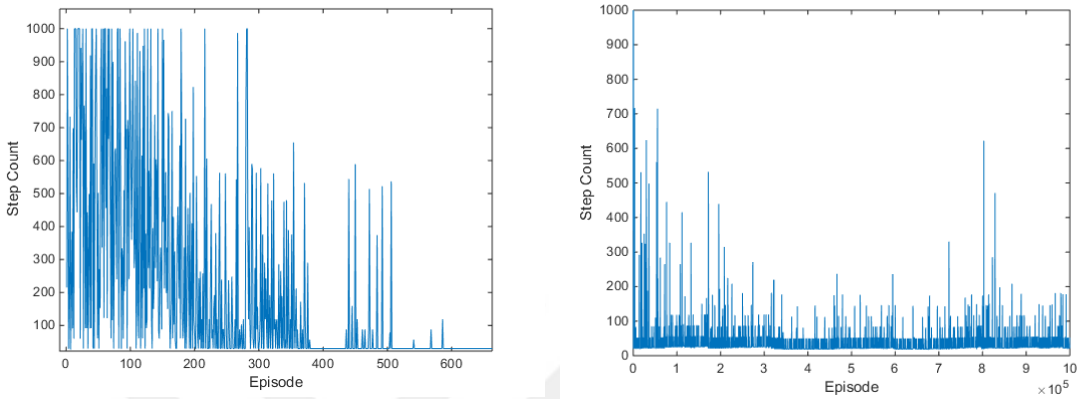
Figure 4.3 Step count results of Maze 1.a

The step count of the maze in Figure 4.2(a) for each episode. Policy convergence is around the 400th episode for the STRL while the Q learning agent cannot converge and does fluctuate.

In Figures 4.3, 4.4 and 4.5 the learning performance of STRL and flat Q-learning in respective order are illustrated in terms of step counts for mazes in Figure 4.2 (a), (b) and (c). In the same fashion, the maximum action value is shown in Figures 4.6, 4.7 and 4.8, respectively, for our algorithm and for flat Q-learning for the same mazes in Figure 4.2 (a), (b) and (c), in respective order.

In Figure 4.3(a) and (b) the number of step counts at which STRL and Q-learning have converged to their policies, in respective order, for the maze in Figure 4.2 (a) are shown throughout the learning process over episodes. We observe in Figure 4.3(a) that STRL converges to its optimal policy around 350th episode while in Figure 4.3(b) the step counts of Q-learning continuously tend to decay towards the optimal step count as long as the cat catches the mouse at the same state. Whenever it does not abide to the optimal policy, the catch-up increases the step count. The frequency of overshoots shows the cat's misses at the state that it used to catch the mouse at. The frequency of the misses

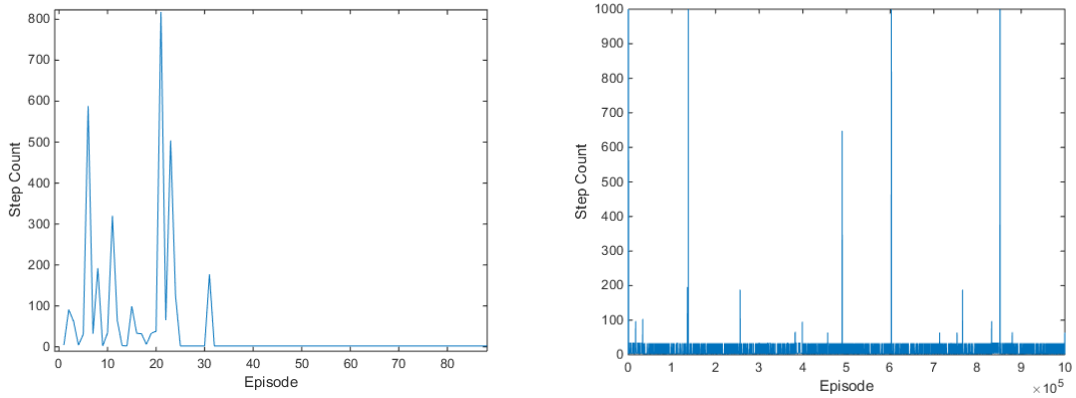
in Figure 4.5(b) is relatively lower than those in Figure 4.3(b) and Figure 4.4(b) since the starting states of both the mouse and the cat are very close (see maze in Figure 4.2(c)) which makes the cat's catch, which is near the mouse, highly probable.



(a) STRL (b) Q-Learning

Figure 4.4 Step count results of Maze 1.b

The step count of the maze in Figure 4.2(b) for each episode. Policy convergence is around the 400th episode for the STRL while the Q learning agent cannot converge because it forgets the already learned policy at each change.



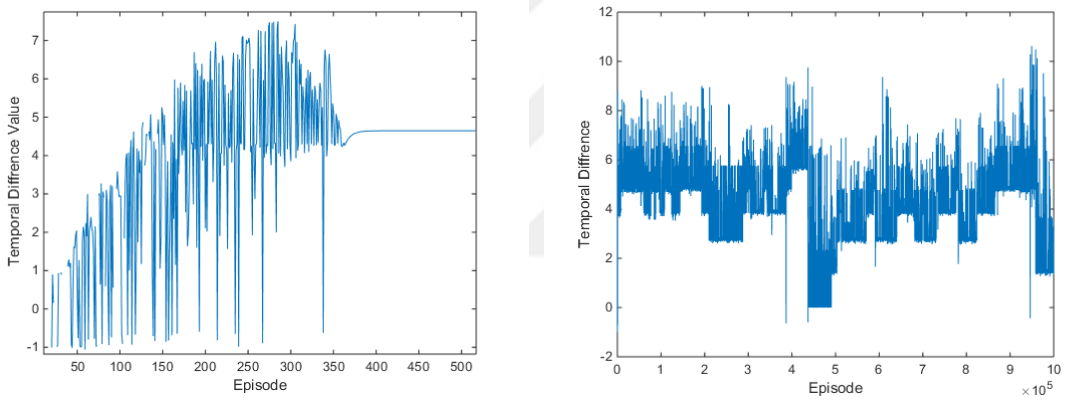
(a) STRL (b) Q-Learning

Figure 4.5 Step count results of Maze 1.c

The step count of the maze in Figure 4.2(c) for each episode. Policy convergence is around the 30th episode for the STRL while the Q learning agent cannot converge and

does fluctuate a little because of the distance to the goal.

At the same time, it never converges as we see from the continuous fluctuations in Figure 4.5(b) at the bottom close to the horizontal axis; i.e., it continuously forgets what it learned since there is no space for error; if it misses the mouse it needs to wait until the mouse restarts running for the cheese again by restarting at its spawn point (Mouse Spawn). When the mouse restarts its run, the cat, which implements the Q-Learning algorithm, is highly likely to catch the mouse at its start state. Thus, the Q table of the cat is reinforced for this catch state. Because the cat sometimes catches the mouse in the general surrounding of the start state of the mouse, the cat policy quickly returns to near optimal step counts between each relatively less occurring over shoots as it reinforces this behavior at some time instance t.

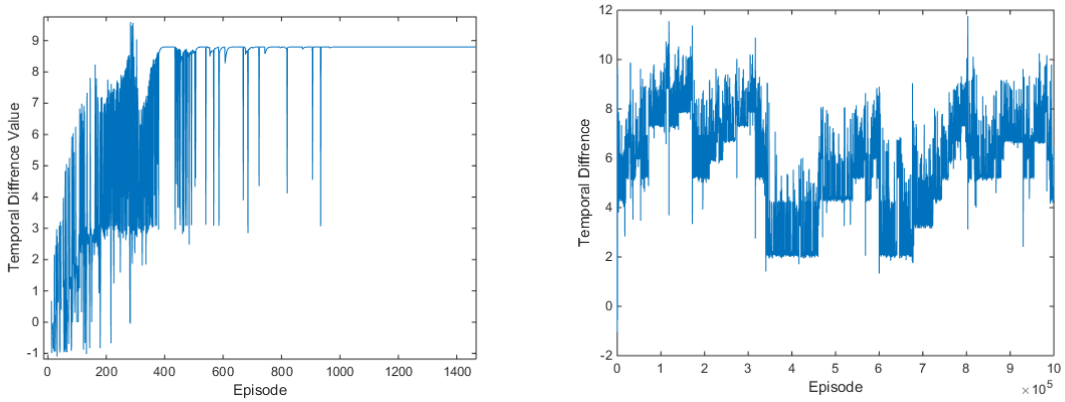


(a) STRL (b) Q-Learning

Figure 4.6 The TD of Maze 1.a

The TD of the maze in Figure 4.2(a) for each episode. Policy convergence is around the 400th episode for the STRL while the Q learning agent's TD values fluctuate.

In Figure 4.6(a) we see that the cat explores the maze and finds the mouse on some occasions thus raising its temporal difference. After the convergence point around episode 400, the cat has learned the policy of the mouse thus the stabilization in the temporal difference while the Q-learning agent in Figure 4.6(b) is unable to pick up the changes and incorporate and thus never converges.

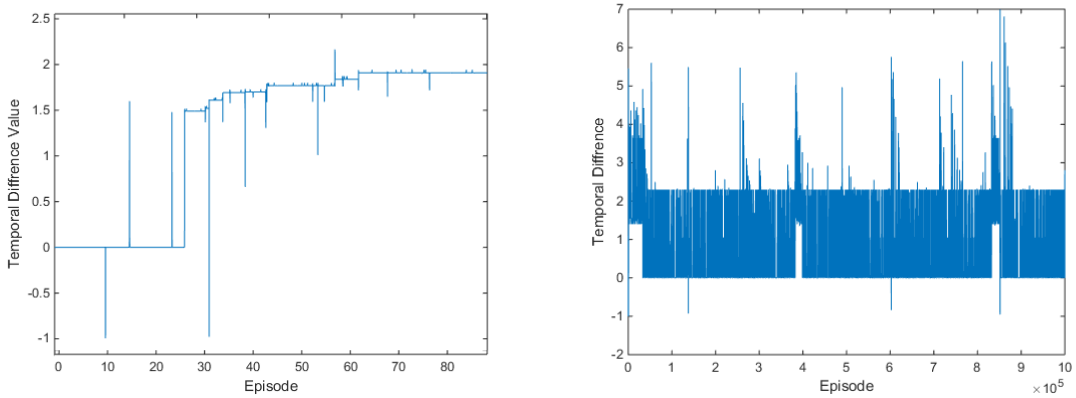


(a) STRL (b) Q-Learning

Figure 4.7 The TD of Maze 1.b

The TD of the maze in Figure 4.2(b) for each episode. Policy convergence is around the 400th episode for the STRL while the Q learning agent's TD values fluctuate.

In Figure 4.7(a) the cat searches the same as in case Figure 4.2(a), but this time the cat discovers the point of interest/choke point that is the goal state of the mouse around episode 450 (the fluctuations are due to the exploration rate which can be adjusted) while the Q-learning agent in Figure 4.7(b) still fails to converge to an optimal policy due to the non-stationary goal.



(a) STRL (b) Q-Learning

Figure 4.8 The TD of Maze 1.c

The TD of the maze in Figure 4.2(c) for each episode. Policy convergence is around the 30th episode for the STRL while the Q-Learning agent's TD values fluctuate a little because of the distance to the goal.

Finally, at Figure 4.8(b) we can see that the cat struggled to find the mouse in the limited time frame and oscillates without convergence. The reason that the agent struggles is because there is no room for the cat's policy to make a single error which will cause the mouse to escape the cat thus increasing the step counts in the episode as the cat cannot catch the mouse on its first run to the cheese. On Figure 4.8(a) we can see that at episode 30 the cat encounters the mouse on its second run, thus a non-optimal policy has occurred since the optimal policy on the maze shown in Figure 4.2(c) is to catch the mouse while it is still near the mouse's spawn point. After that the cat, which incorporates STRL, learns the point of interest and begins to catch the mouse, which stabilizes its TD values.

Additionally, we tested STRL to study a variant of the work where the STRL agent and dynamic goal have different execution times of their actions in a specifically designed maze. Here the cat is half as fast as the mouse thus has only two possible catch points. The reason for that is that this maze also features cells not traversable by the cat which are only available for the mouse to traverse.

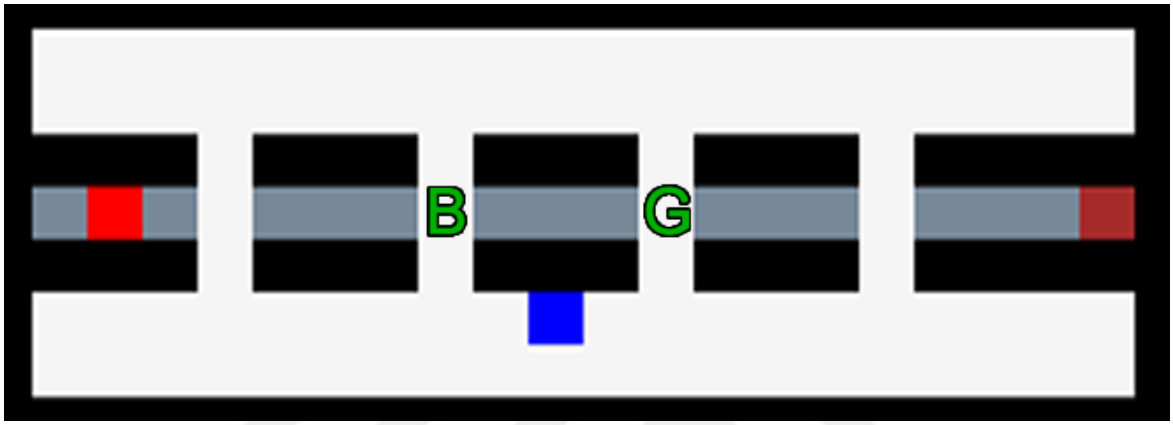
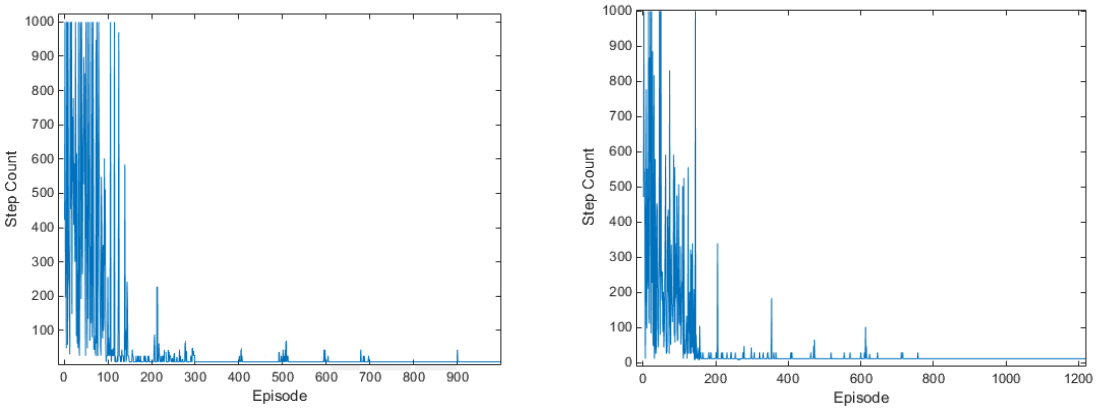


Figure 4.9 The varying speed test maze (Maze 2)

Brown square: mouse, Blue square: cat, Red Square: cheese. Green B: Cats non-optimal (a.k.a. Bad) ambush point, Green G: Cats optimal (a.k.a. Good) ambush point, Grey Cells: Cells which only the mouse can traverse trough.

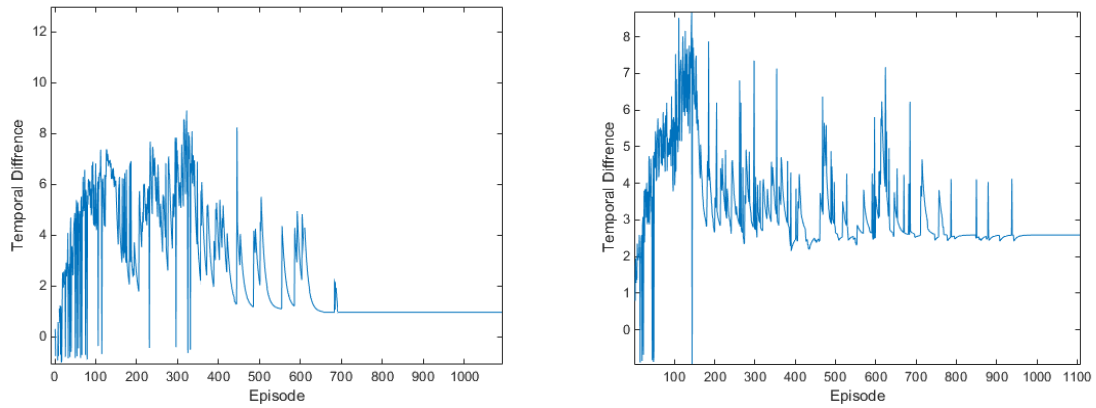
In Figure 4.9, point G is the optimal point of ambush for the cat since cat will catch mouse sooner. This point is a difficult point to learn since the agent has no room for an erroneous move to make which will cause the cat to miss the mouse. The cat can also catch the mouse on point B but this will be a non-optimal policy and we will regard this policy as a poor solution to the problem. In this test, the cat that uses the STRL algorithm devises a policy that catches the mouse at Point G while the cat that uses the Q-Learning algorithm catches the mouse at Point B. While both agents catch the mouse, the policy learned by the STRL cat is an optimal policy since it catches the agent in 8 steps while the Q-Learning agent catches the mouse in 12 steps by going to Point B as it can be seen in Figure 4.10.



(a) STRL (b) Q-Learning

Figure 4.10 The step count of Maze 2

The step count of the maze in Figure 4.9 for each episode. The Policy converges around the 300th episode in the STRL algorithm while the Q learning agent learns it around 350th episode. The STRL agents final policy step length is 8, while the Q-Learning agents is 12.



(a) STRL (b) Q-Learning

Figure 4.11 The TD of Maze 2

The TD of the maze in Figure 4.9 for each episode. The STRL algorithm converges faster than the Q-Learning algorithm.

On each test, the agent quickly picks up the general behavior and the correct interest points of the mouse and creates an optimal policy in short number of episodes while the Q-learning agent fluctuates or finds a non-optimal policy, thus failing to converge to an optimal policy. This also proves the robustness of our approach. The fluctuations in the TD values after the convergence point are due to the rate of exploration. While we lower the exploration rate at each episode, we keep a non-zero exploration rate at its minimum to let it further refine the policy if there is any possible improvement. Such an approach could prove useful when we try to catch an agent which implements an evasive behavior. Also, the exploration rate encourages the cat to discover new choke-points at which it can catch the mouse more efficiently.

4.3. Ms.Pacman test environment

To test our method against other approaches, we tested our algorithm against the Dynamic Q-Network (DQN) by Mnih et al. (2015). We used the Ms. Pacman benchmark in their paper as a basis of our test which has several *choke points*. This test consists of our agent (Pacman) which always begins at the same location each turn. The game also features 4 nemesis AI units called "Ghosts". Each ghost has its own behavior to follow; the Red Ghost (Blinky) chases the Pacman when it is in its line of sight, the Pink Ghost (Pinky) tries to ambush the Pacman ahead of it, the Blue Ghost (Inky) uses the position and orientation of the Pacman and itself to determine its target destination and the Orange Ghost (Clyde) is in a loop mode in some arbitrary path unless being disturbed by the Pacman, in which case it will scatter around and loop again. The environment features pellets which will add *10pts* to the players score, by eating the Power Pellet (a.k.a. Capsule) the player gains *50pts* to their score and the Ghosts transition into a "Scared State" in which the Pacman can eat them too. Each ghost eaten will reward the player with $(100 * 2^{(\text{ghost eat streak})})$*pts.* We will only use the first level of Ms.Pacman to compare STRL to DQN as shown in Figure 4.12.

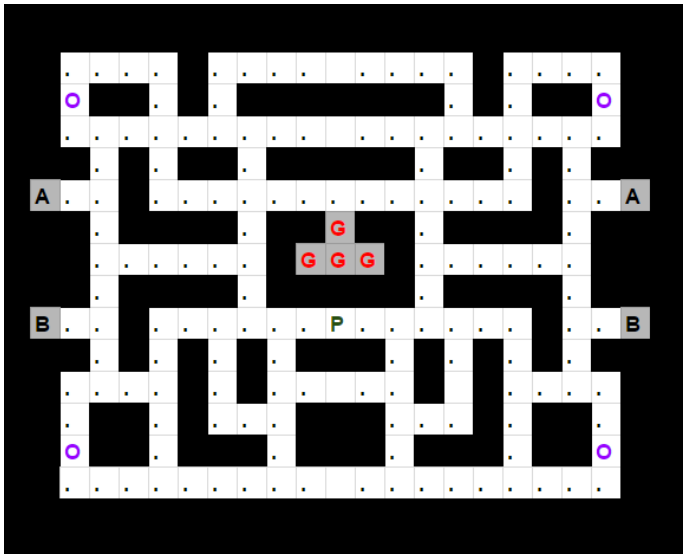


Figure 4.12 The first level of Atari Ms.Pacman

Red G denotes the spawn of the ghosts, which is inaccessible by the pacman. Green P denotes the spawn of the pacman. Purple O's denote the capsules, which consumed will change the state of the ghosts to scared so that the pacman can eat them within a certain time frame. The gray areas marked as A and B are connected cells, meaning that the pacman can go through one and can come out of the other side. The foods are denoted as the little black dots across the maze.

4.4. Ms.Pacman test results

We implemented the STRL algorithm to play the MS. Pacman game which we will compare ourselves to study case for the DQN algorithm by Mnih et al. (2015). For this experiment, we have used the first level of the game as shown in Figure 4.12 since it is also used by Mnih et al. (2015). In this experiment, we used the same parameters as the DQN algorithm to ensure comparable results ($\varepsilon = 1.0$ (with a 0.0001 reduction after each episode and a minimum boundary as 0.1), $\gamma = 0.99$ and $\alpha = 0.1$. The spatio-temporal tables; $f(s, t)$ and $c(s, t)$ are initialized with a zero value to form an empty frequency data.

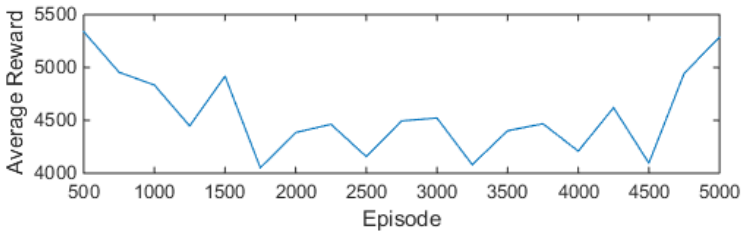


Figure 4.13 Running average reward of 5000 episode trainings of MS.Pacman.

Table 4.1 Comparison between different algorithms in Ms.Pacman

(Some results are taken from Mnih et al. (2015)).

	Avg. Score
Random play	307.3
SARSA	1227
DQN	2311± (525)
STRL	4556.15
Human	15693

As to the reduction of the state space complexity we used feature extraction as part of our algorithm. We gave the agent the following *x(s)* feature vector;

- The distance to the closest food.
- The state of the ghosts (scared or normal).
- The number of capsules left.

Using this *x(s)* we trained the agent in conjunction with STRL and obtained the results as shown in Figure 4.13 in a span of 5000-episode learning. Our agent can learn policies yielding a success rate of an average of 86.62% (Success is being measured whether the agent can clear the level). Furthermore, our agent outperformed the DQN algorithm in Ms.Pacman by 62% as shown in the comparison in Table 4.1.

5. CONCLUSION

According to our tests the cat learns the goal of the mouse and instead of chasing the mouse it possibly attempts to ambush it or intercept it, thus, greatly increasing its performance by reducing the learning time. In cases where the cat is near the spawn point of the mouse, the cat tends to camp on the exit points of the mouse's lair. In cases where the STRL agent and dynamic goal have different execution times, STRL tries to intercept the dynamic goal ahead of its trajectory. Our results show that our algorithm exceeds algorithms such as the Q-Learning algorithm which cannot converge to an optimal policy fails at detecting the non-stationary goal state problem. STRL provides fast and reliable solutions by detecting the dynamic goal in a non-stationary environment and deciphers its pattern and learns its points of interest by detecting the context switches faster. As for the Ms.Pacman test the STRL learns the environment in less episodes than DQN and yields greater results by 62% which shows a promising field for further study. As this thesis suggests, STRL can detect patterns of dynamic goal states, lead the RL agent to the optimal solution faster using reward manipulation and is highly generic to use in a variety of scenarios and can be combined with other RL techniques which can be studied even further. Despite being a good solution to the dynamic goal problem, the algorithm uses a stochastic process, which is at minimum, the size of the optimal policy Q^* thus presenting an extra memory requirement and it also requires an extra design consideration of the problem formulation. As the results suggest this kind of a solution to the non-stationary RL problem brings many interesting applications to the field.

6. Future Work

Several possible extensions of the current research may be the case where the mouse is smart and tries to avoid the cat to further understand the nature of this approach. In this context, the mouse could adopt an avoidance strategy while trying to reach the goal. In such a scenario, we could feed the mouse with the catch points and allow the mouse to develop a new policy by using the catch points and giving the mouse a bad reward that will decay starting at time of catch. Thus, the mouse will possibly learn to avoid the cat with each new policy. Further optimizations can be made by giving priority to some states to encourage the camping ambush behavior. The solution to this problem can be further investigated by analyzing cases where the mouse also implements an evasive policy instead of a learned/predefined policy. Also further research should be conducted on situations like which is observed in the Ms.Pacman problem. One can implement other classic Atari games to further refine the algorithm and benchmark it with algorithms such as the DQN. Furthermore, the extra memory requirement and the extra input on the problem formulation might be further optimized with further research in this field.

REFERENCES

Barto, A.G., Sutton, R.S., Watkins, C.J., 1989. Learning and sequential decision making, in: Learning and computational neuroscience, Citeseer.

Da Silva, B.C., Basso, E.W., Bazzan, A.L., Engel, P.M., 2006. Dealing with non-stationary environments using context detection, in: Proceedings of the 23rd international conference on Machine learning, ACM. pp. 217–224.

Doob, J.L., 1953. Stochastic processes. volume 101. New York Wiley.

Doya, K., Samejima, K., Katagiri, K.i., Kawato, M., 2002. Multiple model based reinforcement learning. Neural computation 14, 1347–1369.

Faußer, S., Schwenker, F., 2011. Ensemble methods for reinforcement learning with function approximation, in: International Workshop on Multiple Classifier Systems, Springer. pp. 56–65.

Gehrke, J.D., Wojtusiak, J., 2008. Traffic prediction for agent route planning, in: International conference on computational science, Springer. pp. 692– 701.

Kaelbling, L.P.,1993. Learning to achieve goals, in: IJCAI, Citeseer.pp.1094– 1099.

Kaelbling, L.P., Littman, M.L., Moore, A.W., 1996. Reinforcement learning: A survey. Journal of artificial intelligence research, 237–285.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.,2015. Human-level control through deep reinforcement learning. Nature518, 529–533

Papis, B., Wawrzynski, P., 2013. dotrl: A platform for rapid reinforcement learning methods development and validation, in: Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on, IEEE. pp. 129–136.

Rummery, G.A., Niranjan, M., 1994. On-line q-learning using connectionist systems.

Śnieżyński, B., 2009. Agent strategy generation by rule induction in predator prey problem, in: International Conference on Computational Science, Springer. pp. 895–903.

Sutton, R.S.,1988. Learning to predict by the methods of temporal differences. Machine learning 3, 9–44.

Sutton, R.S., Barto, A.G., 1998. Reinforcement learning: An introduction. volume 1. MIT press Cambridge.

Sutton, Richard S., Doina Precup, and Satinder Singh., 1999. "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning." Artificial
intelligence 112.1-2 181-211.

Yücesoy, Y.E., Tümer, M.B., 2015. Hierarchical reinforcement learning with context detection (hrl-cd). International Journal of Machine Learning and Computing 5, 353–358.

CURRICULUM VIRTAE

BURAK GÖNCÜ

Esenler Mah. Miraç Sok. No.: 10 D.: 9 Pendik/İstanbul

Phone: +90 (216) 397 74 27 / GSM: +90 (555) 538 85 80

E-**Mail**: bmgoncu@gmail.com

EDUCATION

2014-Present:	Marmara University (MS in Computer Engineering), ISTANBUL
2010-2014:	Istanbul Technical University (BS in Computer Engineering), ISTANBUL
2006-2010:	Tuzla Anatolian Technical High school (Web Development), ISTANBUL

WORK EXPERIENCE

2014-Present:	GAMEGOS (Game Developer), ISTANBUL
2013-2014:	Microsoft (MSP), ISTANBUL
2012-2012:	Wallit (Web Developer), SAN FRANCISCO
2011-2011:	GuyGood (IOS Developer), ISTANBUL

PERSONAL INFORMATION

Birth Date and Place: 08.09.1992 – Vienna /Austria

RESEARCH INTERESTS

Artificial Intelligence

Machine and Reinforcement Learning