UNIVERSITY OF AGDER

Solving Dynamic Bandit Problems and Decentralized Games using the Kalman Bayesian Learning Automaton

By
Stian Berg

Thesis submitted in Partial Fulfillment of the Requirements for the Degree Master of Science in Information and Communication Technology

Faculty of Engineering and Science
The University of Agder

Grimstad
May 2010

Abstract

Multi-armed bandit problems have been subject to a lot of research in computer science because it captures the fundamental dilemma of exploration versus exploitation in reinforcement learning. The goal of a bandit problem is to determine the optimal balance between the gain of new information (exploration) and immediate reward maximization (exploitation). Dynamic bandit problems are especially challenging because they involve changing environments. Combined with game theory, where one analyze the behavior of agents in multi-agent settings, bandit problems serves as a framework for benchmarking the applicability of learning algorithms in various situations.

In this thesis, we investigate a novel approach to the multi-armed bandit problem, the Kalman Bayesian Learning Automaton, an algorithm which applies concepts from Kalman filtering, a powerful technique for probabilistic reasoning over time. To determine the effectiveness of such an approach we have conducted an empirical study of the Kalman Bayesian Learning Automaton in multi-armed dynamic bandit problems and selected games from game theory. Specifically, we evaluate the performance of the Kalman Bayesian Learning Automaton in randomly changing environments, switching environments, the Goore game, the Prisoners Dilemma and zero-sum games. The scalability and robustness of the algorithm are also examined.

Indeed, we reveal that the strength of the Kalman Bayesian Learning Automatons lies in its excellent tracking abilities, and are among the top performers in all experiments. Unfortunately, it is dependent on tuning of parameters. We believe further work on the approach could solve the parameter problem, but even with the need to tune parameters we consider the Kalman Bayesian Learning Automaton a strong solution to dynamic multi-armed bandit problems and definitely has the potential to be applied in various applications and multi-agent settings.

Preface

Being eager to solve the inertia problems of the Bayesian Learning Automatons in dynamic environments, Ole Christoffer Granmo confronted me late 2009 with the idea of extending the algorithms with a Kalman filter. We investigated the possibilities and discovered the update equations for a random walk in "Artificial Intelligence A Modern Approach', the AI "bible". We tweaked the algorithm to work in a multi-armed bandit setting and ran a few early tests. The results were suprisingly good and we ended up submitting a paper to IEAAE.

Still, the analysis was somewhat limited, so Ole asked me whether I wanted to continue working on the Kalman Bayesian Learning Automaton in my thesis. This time including more environments and games from game theory. With my recent interest in Bayesian statistics, I accepted. While I first did regret the choice, I was under the impression that I would not learn as much due to being a programming junkie and sceptical to the statistical nature of the work, it was quickly forgotten as I was introduced to the fascinating ideas of game theory.

And while I am not completely satisfied with my own work, I do feel that I learned a lot, both in terms of theory and research methodology. Looking back, there is much that could have been done better or in another way, but I have to regard that as valuable experience which I can apply in later projects.

I would like to thank my supervisor Ole Christoffer Granmo for providing me with feedback and ideas during the project. Classmate Tarjei Romtveit also deserves thanks for helpful advice on latex and report writing.

Stian Berg

Contents

Contents **2**

List of Figures **8**

List of Tables **9**

1 Introduction **13**

1.1 Background 14

1.1.1 Research area 14

1.1.2 The Kalman Bayesian Learning Automaton 15

1.2 Importance of topic 16

1.3 Thesis definition 17

1.4 Research questions 17

1.5 Contributions 20

1.6 Target audience 21

1.7 Report outline 22

2 The Bandit Problem and existing solutions **23**

2.1 Sequential decision making under uncertainty 24
2.2 The Bandit Problem - a model of the exploration vs exploitation dilemma in reinforcement learning 25
2.2.1 Exploration versus exploitation 25
2.2.2 Terminology 26
2.2.3 Environments and types of feedback 26
2.3 Other bandit solvers 27
2.3.1 Greedy approaches 27
2.3.2 Learning automata 29
2.3.3 Adaptive Pursuit 31
2.3.4 Confidence interval based algorithms - The UCB family 32
2.3.5 Adapt-EvE 33
2.3.6 Poker 35
2.3.7 Exponential weight schemes 36

3 The Bayesian Learning Automatons **37**

3.1 The Bayesian Learning Automata family for stationary environments 37
3.2 Bayesian inference vs frequentist inference 39
3.2.1 Frequentist inference 39
3.2.2 Bayesian inference 40
3.3 Learning the parameters of a two-armed Bernoulli bandit problem 41
3.4 Conjugate priors 43
3.5 BLA Normal Known σ^2 44
3.6 The Kalman Bayesian Learning Automaton 45

3.6.1 Theoretical background 45
3.6.2 Algorithm 49

4 Game theory **50**

4.1 Introduction 51
4.2 Basic concepts 52
4.2.1 Branches of game theory 52
4.2.2 Representations of non-cooperative games 53
4.2.3 Utility theory 53
4.2.4 Strategies 53
4.3 Solution concepts 54
4.3.1 Dominant strategy solution 54
4.3.2 Nash equilibrium 55
4.3.3 Maximin 55
4.4 Games investigated in this thesis 59
4.4.1 The Prisoner's Dilemma 59
4.4.2 Goore game 61
4.4.3 Zero-sum games 62

5 Experiments **64**

5.1 Performance measures and accuracy 64
5.2 Parametric analysis 67
5.3 Multi-armed bandit problems 70
5.3.1 Types of environments 70

5.3.2 Results random walk environments 71
5.3.3 Results switching environments 75
5.4 Goore game 80
5.4.1 Goore game results 81
5.5 Zero-sum games 87
5.5.1 Results Game1 88
5.5.2 Results Game2 89
5.5.3 Results two-finger Morra 90
5.5.4 Results Rock, paper, scissors extended 92
5.6 Prisoner's Dilemma 94
5.6.1 Results 94
5.7 Sensitivity 96
5.7.1 Sensitivity to incorrectly specified update variance parameter 96
5.7.2 Sensitivity to incorrectly specified sensor variance parameter 98
5.7.3 Sensitivity to incorrectly specified sensor variance parameter and update variance parameter 99
5.8 Robustness to non-Gaussian feedback 101
5.8.1 Binary experiments 101

6 Discussion **106**

6.1 Performance in random walk environments 106
6.2 KBLA in switching environments 107
6.3 Performance in the Goore game 107
6.4 Performance in zero-sum and non zero-sum games 108

6.5 Scalability 109
6.6 Robustness 109
6.6.1 Sensitivity 109
6.6.2 Binary feedback 111
6.7 Impressions on tuning the parameters of the KBLA compared to other adaptive schemes 112
6.8 Application domains 114
6.9 Thoughts on the Kalman filtering approach 115

7 Conclusion **116**
7.1 Conclusion 116
7.2 Further Work 118

Bibliography **120**

Appendices **122**

A Results that were left out of the main report **124**
A.1 Multi-armed bandit problems 124
A.1.1 Sinusoidal environments 124
A.2 Goore game 126
A.2.1 Results for the flat Gaussian function 126
A.3 Zero-sum games 127
A.3.1 Tournament results for Rock, paper, scissors extended 127
A.4 Sensitivity 127

A.4.1 Sensitivity to incorrectly specified sensor variance parameter and update variance parameter 127

B Missing algorithms from chapter 2 **129**

B.1 ϵ_n-Greedy 129

B.2 UCBTuned 129

B.3 Poker 130

List of Figures

2.1 The Tsetlin automaton 31

3.1 Estimates after 200 time steps in a Bernoulli environment where arm 1 has been pulled 110 times with 70 rewards, while arm 2 has been pulled 90 times with 40 rewards. 43

4.1 Two-finger Morra in extended form 56

4.2 Expected payoff by playing $(a_1 : p, a_2 : 1 - p)$ for the pure responses of player two . . . 57

5.1 Variance development over time 68

5.2 Example of random walk environment with 3 arms 70

5.3 Example of switching environment with 3 arms 71

5.4 Regret for KBLA in fifty armed random walks with varying difficulty 74

5.5 Dynamic and stationary schemes in a switching environment 77

5.6 The compromise between learning speed and accuracy 78

5.7 The Goore game functions used for normally distributed feedback 80

5.8 The impact on learning speed when the sensor variance parameter is increased. 86

A.1 Sinusoidal environment with 3 arms 125

List of Tables

3.1 The BLA family for stationary environments 38

3.2 Likelihood functions and their corresponding conjugate prior 44

4.1 The Prisoners Dilemma 52

4.2 Player1's best replies (underlined) in the Rock, Paper, Scissors game 54

4.3 Payoff matrix for two-finger Morra 56

4.4 An extended variant of the well-known Rock, paper, scissors game 58

4.5 The Prisoners Dilemma 60

5.1 Table illustrating the ensemble average, the average of a column. 66

5.2 Random walk configurations 72

5.3 Results for two-armed environment with observation noise $\sigma_z^2 = 1.0$ and transition noise $\sigma_u^2 = 1.0$ 72

5.4 Results for fifty-armed environment with observation noise $\sigma_z^2 = 1.0$ and transition noise $\sigma_u^2 = 1.0$ 73

5.5 Switching environment configuration 1 75

5.6 Switching environment configuration 2 75

5.7 Regret for switching environment configuration 1 with interchanging every 1000 time steps 76

5.8 Regret for switching environment configuration 1 with interchanging every 250 time steps 78

5.9 Regret for switching environment configuration 2 with interchanging every 1000 time steps 79

5.10 Goore game configurations 80

5.11 Results for 10 player Goore game on peaked Gaussian function with amplitude 20 and where 2 yes votes are optimal. 81

5.12 Results for 100 player Goore game on peaked Gaussian function with amplitude 20 and where 20 yes votes are optimal. 82

5.13 KBLA results for varying values of the update variance parameter in 10 player Goore game on peaked Gaussian function with amplitude 20 and where 2 yes votes are optimal. 82

5.14 KBLA results for varying values of the update variance parameter in 100 player Goore game on peaked Gaussian function with amplitude 20 and where 20 yes votes are optimal. 83

5.15 KBLA results for varying values of the update variance parameter in 100 player Goore game on peaked Gaussian function with amplitude 400 and where 20 yes votes are optimal. 83

5.16 The effect of increasing the sensor variance parameter in a 100 player Goore game on peaked Gaussian function with amplitude 20 and where 20 yes votes is optimal. 84

5.17 Results for 100 player Goore game on peaked Gaussian function with amplitude 400 and where 20 yes votes are optimal. 85

5.18 Probability of selecting the dominating strategy in Game1 88

5.19 Probability of playing the optimal action in Game2 89

5.20 Results for the two-finger Morra game where the optimal mixed strategy is (0.58, 0.42) . 90

5.21 Average total score in the two-finger Morra tournament 91

5.22 Tournament results for KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.5$ against each individual player 91

5.23 The KBLA versus a perfect opponent in two-finger Morra 92

5.24 Arm probabilities when playing versus an instance of itself in Rock, paper, scissors extended 93

5.25 Scores vs self in the iterative Prisoner's Dilemma 94

5.26 Prisoner's Dilemma tournament scores 95

5.27 Probability of testifying (T) for the KBLA and UCBNormal versus each individual opponent in the Prisoner's Dilemma tournament. 96

5.28 Regret at $t = 2000$ for incorrectly specified update parameter in two-armed random walk environment with true σ_u =50. 96

5.29 Probability of selecting the optimal arm for incorrectly specified update variance parameter in two-armed random walk environment with true σ_u =50. 97

5.30 Regret at $t = 2000$ for incorrectly specified update parameter in fifty-armed random walk environment with true σ_u =50. 97

5.31 Probability of selecting the optimal arm for incorrectly specified update parameter in fifty-armed random walk environment with true σ_u =50. 97

5.32 Regret at $t = 2000$ for incorrectly specified sensor variance parameter in two-armed random walk environment with true σ_z =50. 98

5.33 Probability of selecting the optimal arm for incorrectly specified sensor variance parameter in two-armed random walk environment with true σ_z =50. 98

5.34 Regret at $t = 2000$ for incorrectly specified sensor parameter in fifty-armed random walk environment with true σ_z =50. 99

5.35 Probability of selecting the optimal arm for incorrectly specified sensor parameter in fifty-armed random walk environment with σ_z =50. 99

5.36 Regret at $t = 2000$ for incorrectly specified sensor variance parameter and update variance parameter in fifty-armed random walk environment with true $\sigma_z^2 = 50$ and true $\sigma_u^2 = 50$. 100

5.37 Regret at $t = 2000$ for incorrectly specified sensor variance parameter and update variance parameter in fifty-armed random walk environment with true $\sigma_z^2 = 50$ and true $\sigma_u^2 = 1$. . 100

5.38 Results for binary zero-sum game where the optimal action one probabilities are 0.25 for player one and 0.5 for player two. 102

5.39 Results for 10 player Goore game on binary function where 8 yes votes is optimal. . . . 103

5.40 Results for 100 player Goore game on binary function where 80 yes votes is optimal. . . 104

5.41 Regret in the EvE challenge for 10^6 time steps averaged over 100 repetitions. Values are $\times 10^3$. 105

A.1 Results for regret on the sinusoidal environment in figure A.1. 125

A.2 Results for 100 player Goore game on flat Gaussian function with amplitude 20 and where 20 yes votes are optimal. 126

A.3 Results for the RPS extended tournament 127

A.4 Regret at $t = 2000$ for incorrectly specified sensor variance parameter and update variance parameter in two-armed random walk environment with true $\sigma_z^2 = 50$ and true $\sigma_u^2 = 50$. 128

A.5 Regret at $t = 2000$ for incorrectly specified sensor variance parameter and update variance parameter in two-armed random walk environment with true $\sigma_z^2 = 50$ and true $\sigma_u^2 = 1$. . 128

Chapter 1

Introduction

The Kalman Bayesian Learning Automaton is made to address the recurring problem of balancing exploration versus exploitation in reinforcement learning. The Bandit problem is the model that captures this dilemma. The problem is usually described in terms of a gambler and a k-armed gambling machine where each arm has an unknown reward probability. The gambler wants to choose the sequence of arm pulls that maximizes his reward. In turns out that such problems are important in many areas of computer science, for instance web queries and routing. Indeed, it also appears in game theory, a mathematical model of interaction between agents, where each decision may be modelled as an arm on a gambling machine with unknown reward. However, the bandit problem has proven to be hard to solve in general. The dynamic variant, where reward probabilities changes over time, is extra challenging.

Solving dynamic bandit problems by applying Kalman filtering is, to the best of our knowledge, a novel approach. We are interested in how this approach compares to existing solutions. In brief, we intend to evaluate the Kalman Bayesian Learning Automaton in multi-armed dynamic bandit problems and selected multiplayer games from game theory. In addition, we are interested in how the performance scales with the number of actions and interacting players. Last, but not least, since the KBLA has to incorporate prior beliefs, its sensitivity to incorrectly specified priors will also be considered.

By carrying out this work we believe our contribution to the field of artificial intelligence is a comprehensive empirical evaluation revealing several strengths and weaknesses of a Kalman filtering approach in dynamic multi-armed bandit problems.

1.1 Background

1.1.1 Research area

This master thesis fits into an area of machine learning called reinforcement learning. In brief, reinforcement learning involves *agents* that performs *actions* in an *environment*. The environment is capable of giving the agents *feedback* about the actions they perform. The agents adjust their actions based on the feedback they receive.

The learning scheme evaluated in this thesis is a solution to the *bandit problem*. The bandit problem is a well known problem in reinforcement learning modelling the trade-off between exploration and exploitation. The bandit problem can be viewed in terms of a slot machine with k arms and a gambler. Each arm on the machine has a certain, unknown, probability of giving a reward. The gambler wants to maximize the rewards received, how should he proceed?

This is where the fundamental trade-off between exploration and exploitation comes in. The gambler has to try different arms, i.e explore, in order to obtain information about the reward probabilities for each arm, but each time he pulls an arm he might have been better off exploiting existing knowledge by choosing the arm believed to be optimal presently. The problem here is to balance the exploration of arms and the exploitation of existing knowledge in such a way that the long-term reward is maximized [SB98].

Bandit problems can be either stationary or dynamic. If the reward probabilities for each arm are constant then the environment is said to be *stationary*. In contrast, if the reward probabilities are changing with time, we are dealing with a *dynamic* environment. The learning scheme evaluated in this thesis, the Kalman Bayesian Learning Automaton, is designed for both stationary and dynamic environments, but our emphasis in this thesis is first and foremost on environments containing some dynamic aspects.

An environment can be dynamic either in the sense of a time-varying process, or perceived dynamic due to constant influence by some unknown factor. Such unknown factors may for instance be the decisions of adversaries or cooperating players in multi-agent settings. The decisions of each agent influence the state of the environment and therefore also the feedback given each agent. These settings can be analyzed by a a branch of mathematics called game theory, which is becoming increasingly important in computer science because of its applicability in the design and analysis of agents for multi-agent systems[RN02].

Thus, in this thesis we apply the Kalman Bayesian Learning Automaton to both general multi-armed

bandit problems and selected games from game theory with the intention of revealing its strengths and weaknesses in various settings.

1.1.2 The Kalman Bayesian Learning Automaton

The *Bayesian Learning Automaton* (BLA) was first introduced by Granmo in [Gra08]. In this paper, Granmo shows how Bayesian parameter learning techniques can be applied to stationary bandit problems with Boolean feedback.

Granmo's work was later extended by Braadland and Norheim in their master thesis [BN09]. Using the same Bayesian estimation techniques they introduce three new variants of the Bayesian Learning Automaton, which they call the *BLA Poisson*, the *BLA Normal Known* σ^2 and the *BLA Normal Unknown* σ^2. The names are given according to what feedback distribution the scheme handles. The schemes are evaluated thoroughly and reveals strong results in stationary environments. However, in dynamic environments, inertia hinders performance.

Using the scheme designed for stationary environments with normally distributed feedback, the BLA Normal Known σ^2, as basis, we introduce a new scheme in [GB10] intended for environments where the reward probabilities are changing over time. This scheme is a slightly modified version of the BLA Normal Known σ^2 with the addition of a Kalman filter. The Kalman filter is a Bayesian state estimation technique intended for probabilistic reasoning over time. The filter is a recursive data processing algorithm which incorporates all noisy measurements that can be provided to it, and creates an estimate of the variables in interest in such a way that the error is minimized[May79]. It is this scheme, which we call the Kalman Bayesian Learning Automaton, that is investigated in this thesis. Specifically, we will study the performance, scalability and robustness of the scheme. By answering these questions we hope to determine the strengths and weaknesses of the approach.

1.2 Importance of topic

Bandit problems are important because it is a formal model of many real problems. Indeed, the trade-off between exploration and exploitation modelled by the bandit problem is fundamental in many areas of research and our daily life. This is because situations and problems where we can choose between several different actions, and each action has a stochastic outcome, can often be viewed as bandit problems. The payoff for an action corresponds to the payoff for pulling an arm. Trying out different actions, i.e exploration, is risky because each time we might have been better off choosing another action. However, we are forced do some exploration in order to determine which actions are worthwhile. That is, we must explore to maximize our long-term well-being and exploit to maximize our reward.

When the payoff one receives for choosing an action is invariant in time the environment is said to be stationary. However, many real world problems are dynamic in nature. One example is web monitoring where one has to deal with a continuously changing Internet. Another example is adaptive routing where one aims to minimize the network delay in an environment where changes in the network topology is common due to connection loss between nodes.

Most importantly for this thesis however, is that dynamic environments appear in decentralized learning. Decentralized decision making can be formulated as a game, where the problem is to coordinate decentralized players which know nothing about each other or the true nature of the environment. This thesis focus on selected games from game theory which from the players‘ point of view may be treated as a bandit problem. Such games has been used in benchmarking of learning algorithms before[NT89] [ACFS03][ACFS98][IK03][TK93], and we thus consider them an important part of determining the applicability of the Kalman Bayesian Learning Automaton.

In the context of related work we believe our topic is important because of the, to the best of our knowledge, unique approach of using Kalman filtering in non-stationary multi-armed bandit problems. Therefore, even if the results turns out to be inferior compared to other solutions, we consider the evaluation of the Kalman Bayesian Learning Automaton, a learning scheme with Kalman filtering techniques, a valuable contribution to the field of reinforcement learning.

1.3 Thesis definition

Dynamic bandit problems arise in many areas of computer science like adaptive routing, web queries and decentralized learning. In contrast to stationary bandit problems, dynamic bandit problems involves stochastic environments where the reward probabilities of the optimal arm may change with time. Algorithms intended for stationary environments are not suited for such dynamic environments because of inertia. In a recent paper, ***Solving Non-Stationary Bandit Problems by Random Sampling from Sibling Kalman Filters****, we introduce the Kalman Bayesian Learning Automaton which is designed for dynamic bandit problems with normally distributed feedback. The results are promising, but more experiments are needed in order to fully understand the strengths and weaknesses of the approach. In this thesis we intend to extend the work on the Kalman Bayesian Learning Automaton. Specifically, we will propose how the automaton can be used to solve various dynamic bandit problems and selected games from game theory, together with a performance evaluation of the scheme in these contexts. In addition, we will also investigate the scaleability and robustness of the scheme. Based on the outcome of these results we intend to propose application domains where we consider the automaton most applicable.*

1.4 Research questions

The essence of this thesis is to investigate a novel scheme for bandit problems, which is called the Kalman Bayesian Learning Automaton (KBLA). We want to see how this new scheme can be applied in several settings related to the bandit problem. Specifically, our main objectives is to determine the KBLA's performance in general multi-armed bandit problems, its ability to solve selected games from game theory, its scalability and its robustness.

The following research questions will be answered:

- **How does the Kalman Bayesian Learning Automaton perform compared to the BLA Normal Known σ^2 and other bandit learning schemes in dynamic multi-armed bandit problems?**

  The bandit problem is a well known problem modelling the trade-off between exploration and exploitation in reinforcement learning. There exist several variants of this problem, but the essence of the problem can be viewed in terms of a gambler and a slot machine with k arms. Each arm has a certain, unknown, probability of giving a reward when pulled. The gambler wants to maximize

the rewards received.

The predecessor for the Kalman Bayesian Learning Automaton, the BLA Normal Known σ^2 assumes that the probability of getting a reward is constant and should therefore converge to a specific arm. As such, it may be hindered by inertia in dynamic bandit problems where the reward probabilities of the optimal arm may change. The Kalman Bayesian Learning Automaton(KBLA), however, can take into account the fact that the best arm may change from time to time and can therefore avoid converging towards a single choice. We intend to investigate this property by determining how it compares to other bandit learning schemes in such dynamic environments.

- **Is the KBLA able to find better solutions in multiplayer games compared to the BLA Normal Known σ^2 and other learning schemes?**

  Game theory forms the basis for multi-agent systems that make decisions in a distributed fashion. Each agent has limited knowledge of the system and therefore each decision it makes involves uncertainty. In addition, the outcome of each decision is dependant on what the other agents do. Consequently, we may relate game theory to dynamic bandit problems where we view each decision available to the agent (or *player*) as an unknown 'arm' changing over time depending on the actions of the adversaries(or *collaborators*).

  To study the applicability of the Kalman Bayesian Learning Automaton in decentralized systems, we will evaluate its performance in selected games from game theory. In the interest of comprehensiveness, the selected games covers the main elements found in game theory. This involves testing cooperative, non-cooperative, zero-sum, non-zero-sum and mixed strategy games, as discussed in the following paragraphs.

  Cooperative games involves several players which need to cooperate to achieve a common goal, there is no individual payoffs. In a non-cooperative game the players make decisions individually. We will study the Goore game, introduced by Tsetlin in [Tse74], which is a hybrid game with both cooperative and non-cooperative elements. We want to determine whether a team of KBLA players can solve the Goore game faster than other learning schemes.

  Zero-sum games are games where the payoff adds to zero. This implies that a two-player zero-sum game has identical, but opposite, payoffs for each player. The solution to a zero-sum game, the best outcome, can be in terms of playing a fixed strategy or it can be by playing a 'mix' of strategies[1].

[1] Often referred to as *pure* and *mixed* strategies in game theory literature.

We will test the KBLA in both zero-sum games where it is optimal to play a fixed strategy and zero-sum games where mixed strategies achieves the best outcome.

Mixed strategies are not only good for being unpredictable, but also for minimizing the information an adversary can gain by observing one's actions[RN02]. An example of a mixed strategy game is *Rock-paper-scissors* where *rock* beats *scissors* but loses to *paper*. Clearly it is easy to beat a player that always selects rock. Since many learning schemes, for instance the BLA Normal Known σ^2, are made to converge towards a single strategy we believe that it is suboptimal to the KBLA in such mixed strategy games. To investigate whether our belief is true we want to put the KBLA up against other bandit learning schemes in mixed strategy games.

Another game we will investigate is the classical Prisoner's Dilemma. The Prisoner's Dilemma is a two-player, non-cooperative, non-zero-sum game where the most likely outcome, the equilibrium point, is worse than the optimal outcome. This dilemma has relevance in many areas, among others psychology, economics, and politics. When the game is played in succession it is called the iterative prisoners dilemma. The goal for each player is then to maximize their outcome over n iterations. It is this iterative version of the Prisoner's dilemma we will use to check the behavior of the KBLA. Does it converge to the equilibrium point, which is the rational strategy, or does end up cooperating?

- **Does the KBLA suffer from scaleability problems?**

  When the number of players rise it gets increasingly hard for most learning schemes to determine the optimal solution in the Goore game. Many real-life problems like [IK03] is dependent on a large number of agents. Therefore it is important to see how the performance of a KBLA team in the Goore game varies with the number of players in the team.

  Scaleability also concerns the number of "arms" in the environment, the available actions a bandit player(or agent) may apply, and thus we also intend to test multi-armed bandit problems and zero-sum games with various number of actions.

- **How is the performance of the KBLA affected by incorrect parameters? What if the assumption of normally distributed feedback is wrong?**

  The KBLA takes all available information about the environment into account and creates an estimate of the unknown parameters. The *update model* for the environment models how the environment evolves over time and the *sensor model* models how we interpret observations from the environment. However, in practice, it is often the case that we do not know the true update model and sensor model for the environment. One might have to guess or estimate the update model and/or

sensor model. Therefore it is important to determine whether the KBLA is robust and can handle situations where there exist uncertainty in the update model and sensor model.

Also, the KBLA assumes normally distributed rewards, what if it is applied in an environment where this assumption is wrong and instead it receives Bernoulli distributed feedback. Bernoulli distributed feedback is either a *reward* or a *penalty*, that is, a 0 or a 1. Many problems in reinforcement learning have this type of feedback[SB98] and we therefore feel it is important to investigate whether the KBLA is able to learn the best actions in environments like this.

1.5 Contributions

By answering our research questions we will identify strengths and weaknesses of the Kalman Bayesian Learning Automaton (KBLA) in dynamic multi-armed bandit problems. Specifically, our contributions are:

- A performance evaluation of the KBLA in dynamic multi-armed bandit problems. In chapter 5 we compare the KBLA to other bandit learning schemes in several dynamic environments. We show that the KBLA are especially strong in randomly changing environments, and is among the top performers in all tested environments.
- A study of the KBLA in zero-sum and non zero-sum games. theoretic viewpoint. Such games are fundamental for the analysis of rational agents. The experiments in chapter 5 reveal the KBLA as a very rational player in all tested games. In fact, it quickly determines a near-optimal strategy even in the mixed strategy case.
- An analysis of the KBLA's ability to function as a team, by applying it in the Goore game. The Goore game appear in decentralized learning where one aims to coordinate decision makers in systems with large amounts of noise and uncertainty. In chapter 5 we show that, even though it may need extensive tuning of parameters to do so, the KBLA is able to get close to the optimal solution in the Goore game.
- A look at how the number of interacting players and amount of actions available affects the performance of the KBLA. Indeed, we reveal that the KBLA handles an increasing number of actions

better than traditional learning automatons. Even a randomly changing bandit problem with fifty-arms is tracked with surprisingly high accuracy. Also, the KBLA identifies strong mixed strategies in zero-sum games when increasing the number of actions.

Increasing the number of players in the Goore game on the other hand, unfortunately makes the KBLA very sensitive to its parameter configuration.

- A sensitivity analysis of the KBLA. Both in terms of incorrect parameter configuration and binary feedback. It is shown that the performance of the KBLA is rather sensitive to its update variance parameter in random walks, but the sensor variance parameter does not have a large influence as long as the environment contains a moderate amount of noise. In fact, it is revealed that in some cases an incorrect sensor parameter can significantly reduce regret.

  It is also revealed that the KBLA can deal with binary feedback, and approaches optimal solutions in binary zero-sum games and the binary Goore game. It even performs surprisingly good in an online exploration vs exploitation competition with binary feedback. However, in turns out that the performance of the KBLA in such binary environments is highly dependent on the parameter configuration.

1.6 Target audience

The thesis is written with the intention of being self-contained and in a way that fellow students without any particular previous domain knowledge should be able to follow. Since we touch a lot of different topics like reinforcement learning, learning automata, game theory, decentralized learning and Bayesian statistics, we consider the audience to be anyone interested in these topics. Though, perhaps especially researches wanting to see how Bayesian state estimation techniques can be applied to dynamic multi-armed bandit problems.

By identifying possible domains where the strength of the automaton lies, we also hope our work is of interest to application developers wishing to apply 'state-of-the-art' research to their applications.

1.7 Report outline

Central to this thesis is the bandit problem, the Bayesian learning automatons and game theory. Each of these subjects have received their own chapter. We start with the bandit problem in chapter 2. Here we describe the fundamental theory of Markov decision processes that underlies reinforcement learning and associates this theory to the exploration vs exploitation dilemma which is captured by the bandit problem. Chapter 3 introduces the various Bayesian learning automatons. Specifically, we briefly introduce some of the most important concepts from Bayesian statistics and show an example of how these concepts can be applied in a stationary binary bandit problem, before we derive the equations applied in the Kalman Bayesian Learning Automaton.

Chapter 4 contains an overview of game theory and the games we intend to investigate in this thesis. We also show how to derive the mixed strategy solutions for the games we have included in our experiments. Chapter 5 contains the evaluation of the Kalman Bayesian Learning Automaton. Here, it is compared to several other state-of-the-art algorithms in randomly changing environments, switching environments, zero-sum games, Prisoner's Dilemma and the Goore game. We analyze the results in chapter 6, together with a brief discussion on possible applications domains for the KBLA. Chapter 7 sums up everything together with several interesting proposals for further work.

Chapter 2

The Bandit Problem and existing solutions

The Bandit Problem is a formal model of the exploration versus exploitation dilemma. It asks how a bandit player, or agent, should choose its sequence of actions as to maximize a reward criterion. Exploring actions, i.e gaining information, is important because it allows the agent to estimate the utility of a certain action, but too much exploration drops performance as one may have been better of exploiting existing knowledge.

The amount of exploration needed by the agent is largely dependent on the mechanics of the environment it interacts with, and as such significant work has gone into the development of a general mathematical model for the interaction between an agent and its environment. Currently, the most widespread theoretical models describing this interaction are Markov Decision Processes and Partially Observable Markov Decision Processes. Thus, due to their importance, we use the first part of this chapter to briefly introduce these ideas.

Reinforcement learning is the general term for the approach where agents learn from feedback. We touch into this subject in slightly more depth after introducing the general theory of Markov processes. Furthermore, we provide a more verbose description of the bandit problem and relate the theory of Markov processes to the environments encountered in this thesis. The last part of this chapter is devoted to existing solutions, various algorithms created to address the exploration versus exploitation dilemma.

2.1 Sequential decision making under uncertainty

The Kalman Bayesian Learning Automaton has to make decisions in sequences. One decision per time step. This decision influences the feedback it receives from the environment. Each decision is made according to its own belief - that is, by being Bayesian[1] - about the state of the world/environment.

Sequential decision making problems can be modelled in terms of of a *Markov Decision Process*[RN02], a mathematical model that describes the interaction between the agent and the environment. The only requirement is that the problem satisfies the *Markov property*[SB98, p66]. This property says that the preceding state is only dependent on the previous state, and not on the entire history of states.

A Markov Decision Process (MDP) is given in terms of[RN02]:

- Initial state: s_0
- Transition model: $T(s, a, t)$ - the probability of reaching state t when executing action a in state s.
- Reward model: $R(s, a, r)$ - the probability of receiving reward r when performing action a in state s.

As we shall see, this model is not entirely appropriate for the environments the KBLA is applied to, but for now it suffices to say that MDP's are present in all areas of AI[SB98, p64], and is the main underlying idea that ties together theoretical work in reinforcement learning. A solution to an MDP is called a policy, and defines how the agent should behave (what action is should apply) at a given time[RN02][SB98, p7]. We adopt the notation from [SB98] and [RN02] where a policy is denoted π.

We said earlier that an MDP was not entirely appropriate to model the interaction between the Kalman Bayesian Learning Automaton and the environments encountered in this thesis. An MDP assumes that the environment is fully observable, that is, the agent knows what state the environment is in. The environments encountered in this thesis are only partially observable. In partially observable environments the agent does not know its present state. We will consider environments which are partially observable in the sense that the exist uncertainty about the state of the environment because the feedback from the environment is noisy. There are various ways to deal with uncertainty about state, but the KBLA handles this through a state estimation technique known as Kalman filtering.

[1] Chapter 3 elaborates on what it means to be Bayesian.

A partially observable Markov decision process (POMDP) describes such environments[RN02]. A POMDP contains the same elements as an MDP in addition to an *observation model*[2]. The observation model is $O(s, a, o)$, the probability of perceiving observation o when performing action a in state s. The decision cycle of a POMDP is[RN02]:

1. Based on inferences about the state of the environment, execute the action that maximizes expected outcome $a = \pi^*(s)$.
2. Receive observation o.
3. Predict the next state of the environment by incorporating belief about environment dynamics (transition model) and the knowledge gained from observation o (sensor model).

When the transition model T, reward model R or in the case of POMDP's observation model O is unknown the agent must learn the environment by experimentation. For each action the agent performs, the environment responds with feedback for the purpose of evaluation. Such feedback is called a *reward* or *reinforcement*[RN02]. Also, depending on the environment, the agent may affect the future state of the environment (chess is one example).

Reinforcement learning can be passive or active. Passive reinforcement learning involves agents which has precomputed all actions. Active agents on the other hand, like the Kalman Bayesian Learning Automaton, must decide for itself which actions to perform, and as such need to explore the environment to determine the best actions. It turns out that just how much an agent should explore of the environment is very hard to solve in general. Exploration problems of this kind has generalized under the term *bandit problem*.

2.2 The Bandit Problem - a model of the exploration vs exploitation dilemma in reinforcement learning

2.2.1 Exploration versus exploitation

As previously mentioned, reinforcement learning involves agents that performs actions in an environment. The agent is not aware of the environment it operates in, so it has no way to know whether an action

[2]Or *sensor model*. We use the terms interchangeably.

is worthwhile or not. However, each time the agent performs an action the environment will give the agent feedback about that particular action. And as such, the agent can use the information obtained by performing an action to reason about the utility of the corresponding action. The overall goal of the agent is to choose its sequence of actions in such a way that the overall feedback is maximized[3].

The process of testing out an action to determine whether it is worthwhile is called *exploration*. Thus, the exploration part in the dilemma simply refers to the activity of gaining knowledge about the unknown environment it operates in. However, too much exploration can hurt the performance of the agent. Instead, it should use existing knowledge to determine which action it should perform next. This is the *exploitation* part of the dilemma. That is, when should the agent exploit existing knowledge and choose what it believes is the best action?

The problem is to find the optimal balance between exploration and exploitation.

2.2.2 Terminology

The terms agent, bandit player and learning scheme are used throughout this thesis to refer to algorithms. "Arms" and "actions" are also used interchangeably in this thesis, which refers to the algorithms estimate of a certain choice one can apply in an environment.

2.2.3 Environments and types of feedback

It is important to make the distinction between stationary and non-stationary bandit problems. In a stationary environment the feedback function associated with each arm does not depend on any varying factors such as time. Thus, in a stationary environment the "arms" of the environment has only one state and can therefore be described with a Markov decision process where the transition function $T(s, a, t)$ is independent of the action a and the target state t is always equal to s. The reward function $R(s, a, r)$ on the other hand is unknown and must be learned online be reinforcement techniques.

In a dynamic environment however, the feedback function associated with each arm in the environment is dependent on time. Also, an environment can appear dynamic to an agent when for instance the reward depends on the choice of other agents. In addition, most environments considered in this thesis consists of continuous variables, which implies an infinite state space. As such, it is intractable to describe these

[3] Or minimized, depending on the problem.

environments with a Markov decision process, but it suffices to say that we want to learn, or perhaps more precisely, *track*, a changing continuous reward distribution for each arm using an observation model $O(t, a, o)$ where the observation o depends on the action[4] a at time t.

We also distinguish between a stochastic and non-stochastic environment. Stochastic environments have an element of randomness, which usually means that the feedback function is a probability distribution. All environments considered in this thesis are stochastic.

Two different feedback functions are encountered in this thesis. The first type of feedback is *normally distributed* feedback. Such feedback is sampled from a normal distribution with mean μ and variance σ^2. We refer to the variance as the *noise* of the environment. High noise indicates a hard environment. The Kalman Bayesian Learning Automaton is made for precisely this type of feedback.

The second type of feedback we consider, though not as thoroughly, is perhaps the most discussed environment in literature, namely environments where the feedback function is a Bernoulli distribution. A Bernoulli distribution has boolean feedback where the probability of receiving 1, often referred to as a *reward*, is p and the probability of receiving 0 (a *penalty*) is $1 - p$.

2.3 Other bandit solvers

Here we describe several approaches to the bandit problem, from the most basic to the most advanced schemes we consider to be state-of-the-art in the field.

2.3.1 Greedy approaches

The most obvious approach to the bandit problem would be to calculate the average reward received from an arm and use this as an estimate of the actual value of that arm. This is called the *sample average* method[SB98]. Thus, the estimated value Q of an arm that has been chosen n times with rewards $r_1 \dots r_n$ is:

$$Q = \frac{r_1 + r_2 + \ldots + r_k}{n} \tag{2.1}$$

[4]Since the state of each arm are continuously changing and unobservable except for the arm that is played.

One possibility is to always selected the arm with the highest reward estimate in the action selection process. In other words a pure exploitation strategy that maximizes immediate reward. Such an approach might fail to identify the best arm due to the lack of exploration, so an alternative is to use ϵ-Greedy methods, which involves selecting a random action with a small probability ϵ, as described in [SB98]. The pseudocode of this approach is shown in listing 1, where Q is the reward estimate vector, K is the number of actions and n_i is the number of times arm/action i has been selected.

Algorithm 1 Pseudocode for ϵ-Greedy

1: **Parameters**: $\epsilon \in [0, 1]$
2: **Init**: $Q_i = 0$ for $i \in 1 \dots K$
3: **loop**
4: **if** $Random \in \mathbb{R}[0, 1] < \epsilon$ **then**
5: $i = Random \in \mathbb{Z}[1, K]$
6: **else**
7: $i = \arg\max \frac{Q_j}{n_j}$ where $j \in \{1..K\}$
8: **end if**
9: r = feedback from action a_i
10: $n_i = n_i + 1$
11: $Q_i = Q_i + r$
12: **end loop**

However, balancing exploration and exploitation in this way is not optimal because of constant exploration even when one are quite sure of the reward values. In addition, it is equally likely to explore the worst arm as it is to explore ones with high reward estimates.

[ACF02] describes a variant of ϵ-Greedy which let the ϵ parameter gradually decrease to shift focus from exploration to exploitation. In their paper, the rate of exploration is given by equation 2.2, where c is a tuning constant, K is the number of arms, n is the total number of selections and d is the distance between the best and the second best arm.

$$\epsilon = \min(1, \frac{cK}{d^2 n}) \tag{2.2}$$

We note that, in a dynamic environment this variant might not do any good, as more exploration is needed and the algorithm will suffer performance if its focus is changed entirely to exploitation.

Due to the similarity with ϵ-Greedy the pseudocode of this approach is not shown here, but resides in

appendix B.1.

Another possibility is to create action probability functions of the estimated values[SB98, 30]. That is, a weighted function that ranks each arm according to their value estimates. The arms with the highest reward estimates will then have the highest probabilities of being explored. This is the *softmax* action selection mechanism where the action a on the nth play is selected with probability[SB98]:

$$\frac{e^{Q(a)/\tau}}{\Sigma_{b=1}^{n} e^{Q(b)/\tau}} \tag{2.3}$$

Here, τ is a parameter for controlling the similarity of the action probabilities. That is, a large τ value gives all actions very similar probabilities and becomes a random search in the limit. Oppositely, a small τ value reduces exploration and the limit $\lim_{\tau \to 0}$ equals the greedy action selection mechanism[SB98].

[SB98] also discusses another method for calculating averages. It is based on the motivation that in non-stationary environments the most recent rewards are usually more important than the old rewards. The *exponentially, recency-weighted average* weighs a reward depending on how long ago it was observed and the reward estimate Q at time t is given by[SB98]:

$$Q_t = (1-\alpha)^t Q_0 + \sum_{i=1}^{t} \alpha(1-\alpha)^{t-i} r_i \tag{2.4}$$

The α parameter is a value in the range $[0, 1]$ and controls the weights. The higher the α parameter is, the more rapidly it "forgets" its past and the larger the difference between the past rewards and the most recently received reward. Listing 2 shows the pseudocode for the SoftMax algorithm with exponential recency-weighted averages, but equation 2.4 is rewritten in a recursive manner to avoid the need for considering the entire history of rewards each time step.

2.3.2 Learning automata

Among learning automata (LA) based solutions we find the Linear-Reward-Inaction (L_{R-I}) scheme, Linear-Reward-Penalty (L_{R-P}) scheme and the Pursuit scheme. L_{R-I} and L_{R-P} are based on linear updating of arm selection probabilities while the Pursuit scheme maintains maximum likelihood estimates for the reward probabilites of each arm. Learning automatas can operate in unknown stochastic envi-

Algorithm 2 Pseudocode for SoftMaxEWA

Parameters: $\tau \in \mathbb{R}^+$
Init: $Q_i = 0$ for $i \in 1 \dots K$
Init: $P_i = 1/K$ for $i \in 1 \dots K$
loop
  i = action drawn from probability vector P.
  r = feedback from action a_i
  $Q_i(t) = Q_i(t-1) + \alpha(r - Q_i(t-1))$
  for $j \in \{1 \dots K\}$ **do**
    $P_j = \frac{e^{Q_j(t)/\tau}}{\Sigma_k^K e^{\frac{Q_k(t)}{\tau}}}$
  end for
end loop

ronments and combine rapid and accurate convergence with low computational complexity, but they are dependant on external tuning of parameters[NT89].

Due to their similarity, we show only the pseudocode for Pursuit here, listing 3.

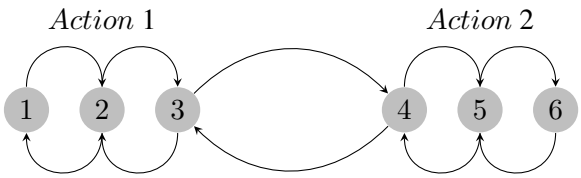
Algorithm 3 Pseudocode for Pursuit

Parameters: $\alpha \in [0, 1]$
Init: $P_j = \frac{1}{K}$ for $j \in 1 \dots K$
loop
  i = action drawn from probability vector P
  $n_i = n_i + 1$
  r = feedback from action i
  $r_i = r_i + r$
  i^* = $\arg\max \frac{r_j}{n_j}$, where $j \in \{1 \dots K\}$
  for $j \in \{1 \dots K\}$ **do**
    $P_j = (1 - \alpha)P_j$
  end for
  $P_{i^*} = P_{i^*} + \alpha$
end loop

The L_{R-P} scheme can avoid converging to a single action by setting the parameters of the algorithm sufficiently small [NT89], the two other schemes however are not made for dynamic environments and tries to converge towards a single action. A learning automaton that is able to handle dynamic environments is the Tsetlin automaton[Tse74] which is very simple. It only keeps track of which state it is in and performs

an action reflecting this state, if the action yields a reward it will become more sure of this action, while on a penalty it will gradually work its way towards a new state and therefore also another action. This is illustrated in figure 2.1. Despite its simple nature it performs remarkably good in some types of dynamic environments and it has been proven in [NT89] that a team of Tsetlin automata can solve the Goore game with sufficient iterations. A downside of the Tsetlin automaton is the requirement of Bernoulli feedback.

Figure 2.1: The Tsetlin automaton



2.3.3 Adaptive Pursuit

Adaptive Pursuit is a variant of the Pursuit scheme specifically designed for non-stationary environments[Thi05]. It uses a parameter P_{min} which denotes the minimum probability an action can have. In this way it ensures that no actions are lost, however this also means that the maximum probability any action can achieve is $P_{max} = 1 - (K - 1)P_{min}$ [Thi05], where K is the number of actions. Thus we see that the maximum performance of this algorithm will only degrade when the number of actions increases because P_{max} decreases.

Listing 4 contains the pseudocde for AdaptivePursuit [Thi05].

Here Q is a vector with value estimates for each action. The parameter α determines how much to weigh the current reward, hence a low value means that it does not take the current reward too seriously while a high value weighs the current reward heavily. A high α is probably benefitial in non-stationary environments since recent rewards can give a better picture of the environment than the reward history.

The β parameter is similiar to the α parameter given to the normal Pursuit scheme. It determines how "similar" the action probabilities are. Thus, a small β parameter leads to more exploration since there is less difference between action probability values.

Algorithm 4 Pseudocode for Adaptive Pursuit

Parameters: $P_{min} \in [0, \frac{1}{K}], \alpha \in [0,1], \beta \in [0,1]$
Init: $P_i = \frac{1}{K}$ for $i \in 1 \dots K$
Init: $Q_i = 1.0$ for $i \in 1 \dots K$
loop
  i = action drawn from probability vector P
  r = feedback from action a_i
  $Q_i(t) = Q_i(t-1) + \alpha[r - Q_i(t-1)]$
  $i^* = \arg\max Q_j(t)$, where $j \in \{1 \dots K\}$
  for $j \in \{1 \dots K\}$ **do**
    if $j \neq i^*$ **then**
      $P_j = P_j + \beta[P_{min} - P_j]$
    end if
  end for
  $P_{i^*} = P_{i^*} + \beta[P_{max} - P_{i^*}]$
end loop

2.3.4 Confidence interval based algorithms - The UCB family

These algorithms assign an optimistic reward estimate within a certain confidence interval [VM05] to each arm. Based on frequency inference, meaning that the more an arm is chosen the closer its optimistic reward estimate will be to the true reward mean. At each step the arm with the highest optimistic mean is chosen.

Auer et al[ACF02] performs an analysis of some confidence algorithms in their paper. Three for binary feedback, the UCB1, UCB2 and UCB1-Tuned algorithms and one for normally distributed feedback, the UCBNormal algorithm. Among the binary schemes, the results show that UCB1-Tuned is the best performer. Common for all of them is that the balancing between exploration and exploitation can be seen by the optimistic reward estimates for the arms, for UCB-Tuned this is:

$$\mu_i + \sqrt{\frac{\ln n}{n_i} \min\{\frac{1}{4}, V_j(n_j)\}} \tag{2.5}$$

Here, n is the total number of arm pulls and n_j is the number of times arm j has been pulled. The V_j term is an upper confidence bound for the variance of arm j and is rather complicated. The interested reader should consult [ACF02] for all details.

We see that arms with few pulls will have greater uncertainty and therefore also more likely to be explored. As the number of arm pulls increases this uncertainty will be lower for all arms and the algorithm is more likely to choose the best arm. However, this also means that the more effort spent on an option will increase the time needed to rediscover the best strategy should the optimal arm suddenly change as in an dynamic environment.

Listing 5 contains the algorithm for UCBNormal as it is described in [ACF02], see appendix B.2 for a listing of the UCBTuned algorithm.

Algorithm 5 Pseudocode for UCBNormal

Initialization phase: $n = 1$, $j = 1$
while $j <= K$ **do**
  if $n_j < 8 \log n$ **then**
    Play action j, update m_j and increment both n_j and n by 1.
  else
    $j = j + 1$
  end if
end while
loop
  i = $\arg\max_j \; m_j + \sqrt{16 \cdot \frac{q_j - n_j m_j^2}{n_j - 1} \cdot \frac{ln(n-1)}{n_j}}$ for $j \in \{1 \ldots K\}$
  r = feedback from action i
  Update mean m_i from r and increment both n_i and n
end loop

2.3.5 Adapt-EvE

The AdaptEvE (Adaptive Exploration vs Exploitation) algorithm from [HBG+07] is specifically intended for both stationary and dynamic environments. It aims to overcome the inertia that hinders the performance of the UCB-based algorithms described above under dynamic environments. It applies a so called change-point mechanism based on Page-Hinkley statistics that is intended to track both sudden and slow changes in the environment.

AdaptEve was the top performer in the EvE challenge hosted by [AHC+06].

AdaptEvE uses UCBTuned at its core. In fact, the algorithm is identical except for the change point mechanism and the way such change points are handled. Their paper they describe two different ways of

dealing with a change point. They call the first the γ-restart mechanism, this variant restarts the parameters of the UCBTuned by a factor of γ. Which they claim that is best when the value is zero, which equals a complete restart of the UCBTuned.

The other variant uses a so-called meta-bandit intended to detect wether a change point was a false alarm. This meta-bandit variant consists of starting a new UCBTuned with two actions, which are referred to as *oldbandit* and *newbandit* where oldbandit is the same UCBTuned running up to this point and newbandit is a UCBTuned started from scratch. The "meta" UCBTuned keeps track of which of the oldbandit and newbandit that receives most rewards and after MT steps the best of the two bandit becomes the new core UCBTuned. This process is repeated at each change point.

While only UCBTuned as the internal UCB algorithm is described in their paper, we assume it is possible to extend their approach to normally distributed feedback with UCBNormal. Thus, when we compare against these algorithms in chapter 5 we use a version with the UCBNormal algorithm in our Gaussian environments and a version with the UCBTuned algorithm in our Bernoulli environments. In listing 6 and 7 we don't make the distinction, and only refer to the internal algorithm as "UCB".

Algorithm 6 Pseudocode for AdaptEvEγ-Restart

Parameters: $\lambda \in \mathbb{R}^+$, $\delta \in \mathbb{R}^+$
Init: PH_0=0
loop
  i = action chosen according to the arm selection process in UCB
  r_i = feedback from action a_i
  $\hat{r_i}$ = average reward for action i
  $PH_t = \max(PH_{t-1} - r_i + \hat{r_i} - \delta, 0)$
  if $PH_t > \lambda$ **then**
    Restart internal UCB algorithm and set PH_0=0
  end if
end loop

Algorithm 7 Pseudocode for AdaptEvE-MetaBandit

Parameters: $\lambda \in \mathbb{R}^+$, $\delta \in \mathbb{R}^+$, $MT \in \mathbb{Z}^+$
Init: PH_0=0, $metaphase = false$
loop
  if $metaphase$ **then**
    Play best of $newbandit$ and $oldbandit$
    Update reward estimate of played bandit and increment n_{meta}
    if $n_{meta} == MT$ **then**
      Set internal UCB as best of $newbandit$ and $oldbandit$
      Reset all variables
    end if
  else
    i = action chosen according to the arm selection process in UCB
    r_i = feedback from action a_i
    $\hat{r_i}$ = average reward for action i
    PH_t = $\max(PH_{t-1} - r_i + \hat{r_i} - \delta, 0)$
    if $PH_t > \lambda$ **then**
      $metaphase = true$
      $oldbandit$=current internal UCB, $newbandit$ = fresh start UCB
    end if
  end if
end loop

2.3.6 Poker

Poker is an abbreviation for the *Price of Knowledge and Estimated Reward* algorithm introduced in [VM05]. According to its creators: "The POKER scheme relies on three main ideas: "pricing uncertainty, exploiting the lever distribution and taking account into account the horizon".

Pricing uncertainty means that it tries to quantify the knowledge gained while pulling a particular lever. Exploiting the lever distribution means that, based on the knowledge it has gained, it tries to estimate the unobserved levers. The last point, taking the horizon into account, says that the remaining number of rounds to play is to be used as a parameter.

Assuming normally distributed feedback, these three properties leads to the following price formula[VM05]:

$$p_i = \hat{\mu}_i + P[\mu_i \geq \hat{\mu} + \delta_\mu]\delta_\mu H \tag{2.6}$$

where $\hat{\mu}_i$ is the estimated reward mean associated with arm i, δ is the estimated reward improvement and H is the rounds remaining.

The algorithm for Poker is fairly long and has therefore been put in appendix B.3.

2.3.7 Exponential weight schemes

Exp3

Exp3 is the simplest of several related exponential weight schemes introduced in [ACFS03]. At each time step t, the probabilities for each arm is given by[ACFS03]:

$$p_j(t) = (1 - \alpha)\frac{w_j(t)}{\sum_{i=k}^{K} w_i(t)} + \frac{\alpha}{K} \tag{2.7}$$

where K is the total number of actions, α is a tuning parameter in the range (0, 1] and w_j is a weight value associated with each action. This probability distribution is a mix of both a uniform distribution and a distribution assigning each arm a probability mass exponential in the estimated cumulative reward for that arm [ACFS03]. The uniform distribution is there because it adds uncertainty and therefore ensures that all arms are explored.

At each time step, the weight value w_j is updated according to

$$w_j(t+1) = w_j(t)exp(\gamma\hat{x}_j(t)/K) \tag{2.8}$$

where $\hat{x}_j$ is $feedback/p_j$ for the selected arm and 0 for the rest. This is the main idea of the algorithm. That is, by dividing the gain by the probability of choosing that action, arms with more rewards will increase their probability of being chosen. Again, we see that similar to many of the previous algorithms, the time needed to adapt in the case of a dynamic environment will increase with the effort spent on a particular arm.

Chapter 3

The Bayesian Learning Automatons

The essence of this thesis is to investigate the Kalman Bayesian Learning Automaton, a learning scheme intended for dynamic environments. However, the concepts of this automaton is largely related to those found in the Bayesian learning automatons intended for stationary environments. Therefore, we feel it is appropriate to devote this chapter to introduce both the Bayesian learning automaton family for stationary environments and the Kalman Bayesian Learning Automaton, together with their theoretical background.

Common and fundamental for all the Bayesian learning automatons is Bayesian inference where Bayes' theorem is central. Bayes' theorem tells us how to update our belief in light of new evidence. By applying Bayes' theorem and *conjugate priors*, probability distributions with special properties, the Bayesian learning automatons can calculate its belief state with suprisingly simple procedures.

The Kalman Bayesian Learning Automaton also applies Bayes' theorem and conjugate priors in light of new observations, however, since the environment may change from time to time it also has to incorporate some extra information regarding the dynamics of the environment. It turns out that, once again, conjugate priors provides the solution.

3.1 The Bayesian Learning Automata family for stationary environments

As mentioned briefly in chapter 1, the first Bayesian Learning Automaton (BLA) was introduced by Granmo in [Gra08]. The BLA was intended for stationary multi-armed bandit problems with Bernoulli

feedback. In such bandit problems, each each "arm" in the environment has a certain probability p of giving a *reward*. In the absence of a reward, a *penalty* is given.

Previous Bayesian approaches to the bandit problem was in general computationally inefficient, and the BLA was therefore made to address precisely this issue. The novel features of the BLA can be summed up in the following:

- It maintains a beta distribution of each "arm" in the environment, where each of these distributions is an estimate of the reward probabilities associated with a particular arm. These beliefs, the *Bayesian estimates*, are used in the arm selection process. This is in contrast to the traditional learning automatons mentioned in chapter 2, which maintains an action probability vector as basis for arm selection.
- It applies a randomized arm selection mechanism based on the Bayesian estimates. That is, during arm selection, instead of using the estimates directly, the automaton draws *random samples* from each of the beta distributions and selects the arm with highest sampled value.

Braadland and Norheim pursued these ideas further in their master thesis ([BN09]) where they showed that Granmo's solution could be generalized to handle different types of feedback. Specifically, they created three new Bayesian learning automatons to handle stationary environments with poisson distributed feedback, normally distributed feedback with known variance and normally distributed feedback with unknown variance, respectively. Table 3.1 gives an overview of the Bayesian learning automatons and the type of feedback they are intended for, while listing 8 contains the general pseudocode.

Table 3.1: The BLA family for stationary environments

Name	**Feedback type**
BLA Bernoulli	Boolean variable
BLA Poisson	Poisson distributed
BLA Normal known σ^2	Normally distributed with known variance
BLA Normal unknown σ^2	Normally distributed with unknown variance

In the following sections we will introduce the concepts from Bayesian statistics relevant to understanding the inner workings of the stationary BLA family. However, since our primary goal is give the reader an understanding the of Kalman Bayesian Learning Automaton, the coverage of these concepts will not be

Algorithm 8 Generic pseudocode for the BLA family

Init: K = number of arms/actions
Init: Set parameters according to prior belief
loop
    Sample x_j from P_j, where P is a probability distribution and $j \in \{1 \dots K\}$
    $i = \arg\max x_j$, where $j \in \{1, K\}$
    $r =$ feedback from action a_i
    Update belief of parameters by using r
end loop

comprehensive. Readers interested in a more complete overview of the stationary BLA family should consult Braadland and Norheims thesis.

3.2 Bayesian inference vs frequentist inference

The Bayesian learning automaton family relies heavily on Bayesian inference. Inference is the process of drawing conclusions from evidence and prior beliefs rather than direct observation. From a statistical viewpoint there are two main approaches to inference, namely Bayesian inference and frequentist inference. While the Bayesian viewpoint is of most interest here, understanding how they differ is helpful for getting in an Bayesian state-of-mind.

3.2.1 Frequentist inference

Frequentists views probability as a property of the system under observation[Nyb]. This property is determined by imagining infinite repetitions of the same test.

Consider flipping a coin. Then there is two possible outcomes. Heads or tails. To determine the probability of receiving heads, a frequentist would repeatedly flip the coin and count the number of heads. Naturally, he cannot perform this experiment forever, so instead he *imagines* what happens if he had performed the experiment infinitely many times. Thus, the probability is inferred by performing a finite number of experiments and with these experiments as basis the real value is approximated. Equation 3.1 generalizes this approach. Here $P(A)$ is the probability of a certain outcome, n is the number of trials and $a(n)$ is the number of such outcomes observed so far.

$$P(A) = \lim_{n \to \infty} \frac{a(n)}{n} \tag{3.1}$$

It is vital to realize that *imagining infinite repetitions* implies using *unobserved* data in the inference process. An important distinction between frequentist inference and Bayesian inference.

3.2.2 Bayesian inference

Bayesians applies same method as the frequentist, they perform experiments repeatedly, but in contrast to frequentist inference, only the actually observed data is used to infer the probability. More specifically, Bayesian inference starts with a *prior* belief about the probabilities. Then, as data arrives, the prior is used together with the newly observed data to infer the new probability. This update procedure is done according to Bayes' theorem, equation 3.2.

$$P(a|b) = \frac{P(a)P(b|a)}{P(b)} \tag{3.2}$$

Bayes theorem is the heart of Bayesian inference and underlies all modern AI systems for probalistic inference [RN02].

When applied to parameter learning, it seems common in scientific literature to use a different notation for a and b to clarify the distinction between data and parameters. We adopt the notation from [GCSR03] where Bayes' theorem is expressed as:

$$P(\theta|y) = \frac{P(\theta)P(y|\theta)}{P(y)} \tag{3.3}$$

Here, θ is the parameter and y is an observed value. The key quantities are $P(\theta)$ and $P(y|\theta)$ which are referred to as the *prior distribution* and *likelihood function* respectively. $P(\theta|y)$ is the inferred distribution and is usually referred to as the *posterior* density.

The factor $P(y)$ is independent of the parameter θ and may therefore be regarded as a constant. As such, for inference we can omit this factor and use the unormalized version of Bayes' theorem, which is given by[GCSR03]:

$$P(\theta|y) = P(\theta)P(y|\theta) \tag{3.4}$$

3.3 Learning the parameters of a two-armed Bernoulli bandit problem

To illustrate how the Bayesian Learning Automaton applies Bayesian inference to learn the parameters of the "arms" in a two-armed Bernoulli environment, we go through an example.

In a Bernoulli environment each arm has a certain, unknown probability of giving a reward. Thus, for each arm we associate a random variable Θ, which can take on any value in the interval $[0, 1]$. This value, which we denote θ, is the parameter we want to learn. Meaning that, at each time step, what we want to know is the probability of a reward given the range of data, $d_1 \dots d_N$, observed so far, that is $P(\Theta = \theta|d_1 \dots d_N)$. This term is called the *posterior*, the *conditional probability*. However, the Bayesian approach to parameter learning updates this probability at each time step[RN02], and as such only the most recent observation $d_i = y$ needs to be considered. Therefore it is sufficient to find the probability at a given time step $P(\Theta = \theta|d_i = y)$ with $y \in \{reward, penalty\}$.

A Bayesian view needs a prior belief. Initially, we have no information available, every value is equally likely. The uniform distribution is one way to represent this, but as we will see in the next section about *conjugate priors*, a beta distribution the perfect choice. A beta distribution for a parameter θ is given in terms of two hyperparameters[1] a and b, which determines the shape of the function. Function 3.5 is the beta distribution. Note that the denominator does not depend on the parameter and serves as a normalization constant.

$$f(\theta, a, b) = \frac{\theta^{a-1}(1-\theta)^{b-1}}{\int_0^1 u^{a-1}(1-u)^{b-1}du} \tag{3.5}$$

Denoting the normalization constant α, we have our prior distribution for Bernoulli environments in equation 3.6.

$$P(\Theta = \theta) = beta[a, b](\theta) = \alpha\theta^{a-1}(1-\theta)^{b-1} \tag{3.6}$$

[1]The term *hyperparameter* is used in Bayesian statistics to avoid confusion with the parameters of the system.

The second quantity we need for a Bayesian update step is the likelihood function. How likely are we to receive a reward or a penalty? Intuitively, receiving a reward is determined exclusively by the parameter θ, the probability of a reward. And as such, the probability of a penalty is just the complement $1-\theta$. With 1 representing a reward and 0 representing a penalty we get:

$$\begin{aligned} P(1|\theta) &= \theta \\ P(0|\theta) &= 1-\theta \\ P(Y|\theta) &= \theta^y \cdot (1-\theta)^{1-y} \end{aligned} \tag{3.7}$$

With both the prior and likelihood in place, we have the complete equation for a Bayesian update step in a Bernoulli environment.

$$posterior = prior \times likelihood \tag{3.8}$$

$$P(\Theta = \theta|d_i = y) = beta[a,b](\theta) \times \theta^y(1-\theta)^{y-1} \tag{3.9}$$

Finally, we can consider what happens when the Bayesian learning automaton applies an action in the environment. If the feedback from the environment is 1, then:

$$P(\Theta = \theta|d_1 = 1) = \alpha\theta^{a-1}(1-\theta)^{b-1} \cdot \theta \tag{3.10}$$

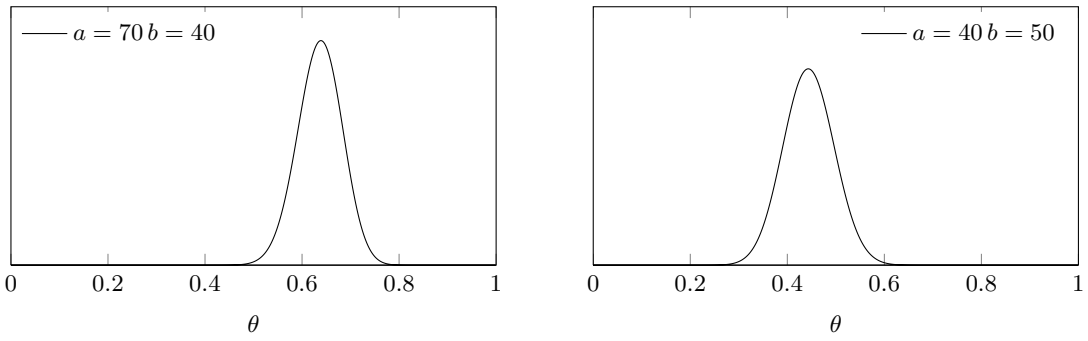
$$= \alpha\theta^a(1-\theta)^{b-1} \tag{3.11}$$

We see that the posterior distribution is just the beta distribution we used as a prior with the a parameter increment by one. Similarly, if we had received 0 as feedback:

$$P(\Theta = \theta|d_1 = 0) = \alpha\theta^{a-1}(1-\theta)^{b-1} \cdot (\theta - 1) \tag{3.12}$$

$$= \alpha\theta^{a-1}(1-\theta)^b \tag{3.13}$$

Then the b parameter is incremented by one. This means that, at every time step, the automaton only needs to increment its parameters to update its belief. The a parameter is incremented on reward and the b parameter incremented on a penalty. Figure 3.1 shows the arm estimates produced by the BLA in a hypothetical case where the first arm has been selected 110 times with 70 rewards and the second arm has been selected 90 times with 40 rewards.



(a) Belief arm 1

(b) Belief arm 2

Figure 3.1: Estimates after 200 time steps in a Bernoulli environment where arm 1 has been pulled 110 times with 70 rewards, while arm 2 has been pulled 90 times with 40 rewards.

As we will see in the next section, this remarkable property of equal parametric forms in the prior and posterior is not unique to this combination of prior and likelihood.

3.4 Conjugate priors

According to the choice of prior distribution, the product of the prior and likelihood may take on different parametric forms. In the case were the posterior is of the same parametric form as the prior, the prior is called a *conjugate prior* for the *likelihood function*. This is vital because then only the parameters need updating when determining the posterior. Only updating the parameters means that we avoid complex mathematical calculations.

In the previous section, we saw that the beta distribution is a conjugate prior to a boolean value, but it can be shown that all probability distributions of the *exponential family* have conjugate priors. Each member of the stationary BLA family are made for handling one such probability distribution. Table 3.2 show four likelihood functions and their corresponding conjugate as applied in the BLA family for stationary environments.

Table 3.2: Likelihood functions and their corresponding conjugate prior

Likelihood function $p(y\|\theta)$	***Conjugate prior*** $p(\theta)$
Bernoulli	Beta
Poisson	Gamma
Normal unknown μ and known σ^2	Normal
Normal unknown μ and unknown σ^2	Normal-Scaled Inverse-X^2

3.5 BLA Normal Known σ^2

As mentioned in chapter 1, the Kalman Bayesian Learning Automaton bears close resemblance to the BLA Normal Known σ^2, because of the assumption of normally distributed feedback. As the other members of the BLA family it is reliant on a conjugate prior to update its parameters in light of new evidence. A conjugate prior for a normally distributed likelihood function is also a normal distribution[Nyb].

Therefore, to learn a stationary, continuous variable X we again apply Bayes' theorem. Denoting the value of the variable x, the first observation z_1 and the variance of the observation σ_z^2 we get:

$$P(x|z_1) = P(z_1|x) \times P(x) = e^{-\frac{1}{2}(\frac{z_1-x}{\sigma_z^2})} \times e^{-\frac{1}{2}(\frac{x-\mu}{\sigma^2})} \tag{3.14}$$

From here, it is just a matter of standard algebra[2] to show that:

$$P(x|z_1) = e^{-\frac{1}{2}\frac{(x-\frac{(\sigma^2 * z_1 + \sigma_z^2 \mu)}{\sigma^2+\sigma_{z_1}^2})^2}{\sigma^2\sigma_{z_1}^2/\sigma^2+\sigma_{z_1}^2}} \tag{3.15}$$

Remembering that a Gaussian distribution is given in terms of $N(\mu, \sigma) = \alpha e^{-\frac{1}{2}(\frac{x-\mu}{\sigma^2})}$, we see that the new mean and variance is calculated from equation 3.15 as follows:

[2]Combine the exponents and *complete the square*, the next section elaborates on this technique.

$$\mu_{n+1} = \frac{\sigma_n^2 z_n + \sigma_z^2 \mu_{n-1}}{\sigma_n^2 + \sigma_z^2}$$

$$\sigma_{n+1}^2 = \frac{\sigma_n^2 \sigma_z^2}{\sigma_n^2 + \sigma_z^2}$$

Together with the sampling process, these equations forms the algorithm for the BLA Normal Known σ^2, for which the pseudocode from [BN09] can be seen in listing 9.

Algorithm 9 Pseudocode for BLA Normal Known σ^2

Init: Set m_j and s_j^2 according to prior belief of μ and σ^2
loop
  Sample x_j from $N(m_j, s_j^2)$, for each $j \in \{1 \dots K\}$
  $i = \arg\max x_j$, where $j \in \{1 \dots K\}$
  $r =$ feedback from action a_i
  $m_i = (s_i^2 r + \sigma_z^2 m_i)/(s_i^2 + \sigma_z^2)$
  $s_i^2 = (s_i^2 \sigma_z^2)/(s_i^2 + \sigma_z^2)$
end loop

3.6 The Kalman Bayesian Learning Automaton

3.6.1 Theoretical background

The Kalman Bayesian Automaton is also made for environments where the feedback is normally distributed like the BLA Normal Known σ^2, but in addition the feedback may change at each point in time. Thus, this time, the variable is dependent on the time t, and we denote the unknown variable X_t for which we want to learn x_t, the value at time t. The prior and likelihood functions are the same as for the BLA Normal Known σ^2.

With a fixed value these equations would suffice, but since we know that the value might change at every time step, we need to know how to update this equation for every time step. Fortunately, we can use the same interesting property that we saw earlier in the section about conjugate priors. Combining a Gaussian function with another Gaussian function gives a posterior distribution which is also a Gaussian

distribution, and may thus also be used to predict the next state if our transition/update model is Gaussian [RN02].

Specifically, the one-step prediction is as equation 3.16[RN02]:

$$P(x_{t+1}) = \int_{-\infty}^{\infty} P(x_{t+1}|x_t)P(x_t)dx_t \tag{3.16}$$

The integral is there because we are updating the state of a continuous variable. The term $P(x_{t+1}|x_t)$ describes how the next state distribution is given from the previous state distribution and is commonly referred to as a *transition model* or *update model* of the environment.

To understand how to create an update model for a Gaussian distributed random walk we can again examine the properties of the normal distribution $\alpha e^{-\frac{1}{2}(\frac{x-\mu}{\sigma^2})}$. The nominator with the mean decides how much the function has been translated rightwards or leftwards and the variance in the denominator stretches the function horizontally. Therefore for a random walk where the mean may shift leftwards or rightwards the nominator for an update step is just the difference $x_{t+1} - x_t$, and the uncertainty of this shifting is represented by σ_x^2 in the denominator.

$$P(x_{t+1}|x_t) = \alpha e^{-\frac{1}{2}(\frac{(x_{t+1}-x_t)^2}{\sigma_x^2})} \tag{3.17}$$

Incorporating this with the one-step prediction gives us the following estimate of x at t_1 before the measurement is taken.

$$
\begin{aligned}
P(x_1) &= \int_{-\infty}^{\infty} P(x_1|x_0)P(x_0)dx_0 \\
&= \alpha \int_{-\infty}^{\infty} e^{-\frac{1}{2}(\frac{(x_1-x_0)^2}{\sigma_x^2})} e^{-\frac{1}{2}(\frac{(x_0-\mu_0)^2}{\sigma_0^2})} dx_0 \\
&= \alpha \int_{-\infty}^{\infty} e^{-\frac{1}{2}(\frac{\sigma_0^2(x_1-x_0)^2+\sigma_x^2(x_0-\mu_0)^2}{\sigma_0^2\sigma_x^2})} dx_0 \\
&= \alpha \int_{-\infty}^{\infty} e^{-\frac{1}{2}(\frac{\sigma_0^2x_1^2-2x_1x_0\sigma_0^2+x_0^2\sigma_0^2+x_0^2\sigma_x^2-2x_0\sigma_x^2\mu_0+\sigma_x^2\mu_0^2}{\sigma_0^2\sigma_x^2})} dx_0 \\
&= \alpha \int_{-\infty}^{\infty} e^{-\frac{1}{2}(\frac{x_0^2(\sigma_0^2+\sigma_x^2)-x_0 2(x_1\sigma_0^2+\mu_0\sigma_x^2)+\sigma_x^2\mu_0^2+\sigma_0^2x_1^2}{\sigma_0^2\sigma_x^2})} dx_0
\end{aligned}
\tag{3.18}
$$

With see that we have an quadratic equation in x_0, that is, $ax_0^2 + bx_0 + c$ with $a = \frac{(\sigma_0^2+\sigma_x^2)}{\sigma_0^2\sigma_x^2}$, $b = \frac{2(x_1\sigma_0^2+\mu_0\sigma_x^2)}{\sigma_0^2\sigma_x^2}$ and $c = \frac{\sigma_x^2\mu_0^2+\sigma_0^2x_1^2}{\sigma_0^2\sigma_x^2}$. A technique known as *completing the square* allows us to rewrite $ax_0^2 + bx_0 + c$ as $a(x_0 - \frac{-b}{2a})^2 + c - \frac{b^2}{4a}$ [RN02]. The residual term does not depend on x_0 and may thus be placed outside the integral.

$$
\begin{aligned}
P(x_1) &= \int_{-\infty}^{\infty} e^{-\frac{1}{2}(a(x_0-\frac{-b}{2a})^2+c-\frac{b^2}{4a})} dx_0 \\
&= e^{-\frac{1}{2}(c-\frac{b^2}{4a})} \int_{-\infty}^{\infty} e^{-\frac{1}{2}(a(x_0-\frac{-b}{2a})^2)} dx_0
\end{aligned}
\tag{3.19}
$$

Now the integral is on the form of a normal distribution with x_0 as the variable, and since, by definition, the integral of a normal distribution from $-\infty \rightarrow \infty$ is 1 we are left with just the residual term.

$$\begin{aligned} P(x_1) &= e^{-\frac{1}{2}(c-\frac{b^2}{4a})} \\ &= e^{-\frac{1}{2}(\frac{\sigma_x^2\mu_0^2+\sigma_0^2x_1^2}{\sigma_0^2\sigma_x^2}-\frac{(\frac{2(x_1\sigma_0^2+\mu_0\sigma_x^2)}{\sigma_0^2\sigma_x^2})^2}{4\frac{(\sigma_0^2+\sigma_x^2)}{\sigma_0^2\sigma_x^2}})} \\ &= e^{-\frac{1}{2}(\frac{\sigma_x^2\mu_0^2+\sigma_0^2x_1^2}{\sigma_0^2\sigma_x^2}-\frac{(x_1\sigma_0^2+\mu_0\sigma_x^2)^2}{(\sigma_0^2+\sigma_x^2)\sigma_0^2\sigma_x^2})} \\ &= e^{-\frac{1}{2}(\frac{x_1^2\sigma_0^2\sigma_x^2-2x_1\sigma_0^2\sigma_x^2\mu_0+\sigma_0^2\sigma_x^2\mu_0^2}{(\sigma_0^2+\sigma_x^2)\sigma_0^2\sigma_x^2})} \end{aligned} \tag{3.20}$$

Again, we have a quadratic, this time in x_1. After completing the square and simplifying we obtain the one-step prediction in equation 3.21[RN02]. We see that the prediction is just a normal distribution with the same mean, but the function is stretched further horizontally because there is more uncertainty in regard to its new value.

$$P(x_1) = \alpha e^{\frac{1}{2}(\frac{(x_1-\mu_0)^2}{\sigma_0^2+\sigma_x^2})}. \tag{3.21}$$

Then, after incorporating the observation our belief is reflected in equation 3.22. We omit the complete derivation, but yet again, this is solved by standard algebra and completing the square with regard to x_1.

$$\begin{aligned} P(x_1|z_1) &= \alpha P(z_1|x_1)P(x_1) \\ &= \alpha e^{-\frac{1}{2}(\frac{(z_1-x_1)^2}{\sigma_z^2})} e^{-\frac{1}{2}(\frac{(x_1-\mu_0)^2}{\sigma_0^2+\sigma_x^2})} \\ &= \alpha e^{-\frac{1}{2}\left(\frac{(x_1-\frac{(\sigma_0^2+\sigma_x^2)z_1+\sigma_z^2\mu_0}{\sigma_0^2+\sigma_x^2+\sigma_z^2})^2}{(\sigma_0^2+\sigma_x^2)\sigma_z^2/(\sigma_0^2+\sigma_x^2+\sigma_z^2)}\right)} \end{aligned} \tag{3.22}$$

From this, we see that the new mean and variance can be updated recursively as follows.

$$\mu_{t+1} = \frac{(\sigma_t^2+\sigma_x^2)z_{t+1}+\sigma_z^2\mu_t}{\sigma_t^2+\sigma_x^2+\sigma_z^2} \tag{3.23}$$

$$\sigma_{t+1}^2 = \frac{(\sigma_t^2+\sigma_x^2)\sigma_z^2}{\sigma_t^2+\sigma_x^2+\sigma_z^2} \tag{3.24}$$

3.6.2 Algorithm

The Kalman Bayesian Learning Automaton works like the other members BLA family. First it samples values from all of its arms and selects the arm with the highest sampled value. The index of this selection is passed to the environment which provides the automaton with feedback, feedback which is used for updating its parameters.

The KBLA differs from the BLA Normal Known σ^2 in two ways. The first is the update equation for the parameters, which we saw earlier, is almost identical to the BLA Normal Known σ^2. The second difference is due to the fact that time goes on, and we have to assume changes in the environment. This means that, for each time step, one becomes more uncertain about the arms that were not selected. Therefore, the variance of the these unobserved arms are incremented by the variance of the update model.

Algorithm 10 Pseudocode for the Kalman Bayesian Learning Automaton

Init: Set σ_u^2 and σ_z^2 according to the environments update model and sensor model
Init: Set m_j and s_j^2 according to prior belief of μ and σ^2
loop
  Sample x_j from $N(m_j, s_j^2)$, for each $j \in \{1 \dots K\}$
  $i = \arg\max x_j$, where $j \in \{1, K\}$
  $r =$ feedback from action a_i
  $m_i = \frac{(s_i^2+\sigma_u^2)r+\sigma_z^2 m_i}{(s_i^2+\sigma_u^2+\sigma_z^2)}$
  $s_i^2 = \frac{(s_i^2+\sigma_u^2)\sigma_z^2}{(s_i^2+\sigma_u^2+\sigma_z^2)}$
  for $j \in \{1 \dots K\}$ **do**
    if $j \neq i$ **then**
      $s_j^2 = s_j^2 + \sigma_u^2$
    end if
  end for
end loop

Chapter 4

Game theory

Game theory is a fascinating model of interaction. Interestingly, it also has it uses in computer science and has become a major part of this thesis. How the Kalman Bayesian Learning Automaton behaves in such games can answer questions related to the applicability of the automaton in various settings.

Game theory allows us to analyze the behavior of agents when they are influenced by other agents. Games are commonly represented as a matrix, where each cell specifies the payoff for a particular combination of strategies. A strategy profile can either be in pure strategies or mixed strategies. A pure strategy consists of only playing a single action, while a mixed strategy randomizes between actions. Often, it is not immediately obvious what one should consider an optimal strategy to a game, but there exist several solution concepts, where the most important for us are the Nash Equilbrium solution and the minimax solution. These solutions often calls for mixed strategies.

In this thesis we focus on zero-sum games, where the total payoff adds to zero. We consider both zero-sum games with pure strategy solutions and zero-sum games with mixed strategy solutions. We also consider the Goore game, where a team of players aims to maximize a global reward criterion without any communication.

4.1 Introduction

Game theory is applicable in a wide range of domains as for instance biology, politics and social sciences. For our purposes it sufficies to say that game theory helps us model the interactions between agents. From chapter 2 we know that the bandit problem is concerned with the decisions of a single agent. Game theory takes this one step further and asks what happens when an agent (or *bandit player*) starts interacting with other agents and as such are influenced by their decisions.

More specifically, in computer science there is two main areas where game theory has proven to be important[RN02]:

- Rational agent design - How should an agent be designed to maximize its performance? A *rational* agent is one that takes prior beliefs and received evidence into account, and selects the action that maximizes its performance measure. Game theory is important in agent design because it allows one to analyze the agents decisions and compute the expected utility for each action, and therefore also determine the best action[RN02].
- Mechanism design[1] - The task of designing environments that maximizes the global utility. Even though agents are acting rational and trying to maximize their own utility, mechanism design seeks to design environments in such a way that the collective decision favour a particular outcome. For instance, how do you design a protocol for the internet where users are greedy and want to maximize their own throughput? The current solution, the TCP protocol is not designed with game theory principles. Instead it relies on settings being hidden from the normal user in the operating system, but for a knowledgeable and "greedy" user it is easy to exploit this protocol. In the case where everybody acts greedy the overall performance is lowered for everyone. Mechanism design on the other hand assumes that everyone acts rational, that is, to maximize their own well-begin, and aims to work around the aforementioned issues by designing environments where the global maximum performance is reached exactly in the case where every agent is acting rational.

To see exactly what kind of questions game theory can help us answer, we feel it is worthwhile to continue with an example of a game before we jump into the theory. For this example we have chosen the Prisoners Dilemma, the game that popularized game theory[HV04]. The classical Prisoners Dilemma is described in the following way[Cho07]:

[1] Also called inverse game theory[NRTV07].

> ”Two suspects, A and B, are arrested by the police. The police have insufficient evidence for a conviction, and, having separated both prisoners, visit each of them to offer the same deal: if one testifies for the prosecution against the other and the other remains silent, the betrayer goes free and the silent accomplice receives the full 10-year sentence. If both stay silent, the police can sentence both prisoners to only six months in jail for a minor charge. If each betrays the other, each will receive a two-year sentence. Each prisoner must make the choice of whether to betray the other or remain silent. However, neither prisoner knows for sure what choice the other prisoner will make.”

Game theory illustrates this situation by a *payoff matrix*, where the cells contains the payoff (years in prison) to each player for a particular combination of decisions. The payoff matrix for the Prisoners Dilemma is shown in table 4.5.

Table 4.1: The Prisoners Dilemma

A/B	Refuse	Testify
Refuse	A=-0.6,B=-0.6	A=-10, B=0
Testify	A=0,B=-10	A=-2,B=-2

Knowing this, several questions arises. Can we predict the behavior of the players? What is the most rational decision? What can be considered a “solution” to this game? Would allowing communication between the players influence the decisions? Would the outcome be different if played repeatedly against the same opponent?

From here, we will continue with concepts relating to the understanding of how game theory can answer these questions.

4.2 Basic concepts

4.2.1 Branches of game theory

We can divide game theory into two branches, which is *non-cooperative game theory* and cooperative game theory. In non-cooperative game theory we are concerned with the individual. We want to study

what actions we can expect from an agent under certain conditions. In coalitional game theory however, we are concerned with how an entire group behaves[SL08].

Our focus in this thesis is non-cooperative games, and as such most of the remaining text in this chapter is related to non-cooperative games. Only one of the games we intend to study, the Goore game, has some cooperative elements.

4.2.2 Representations of non-cooperative games

There are two common ways to represent a game in game theory[HV04, 45]. The first is the *normal form*. This is the "matrix" form we used for the prisoners dilemma in the start of the chapter. If we think of a normal form game as an environment, then each cell in the matrix corresponds to a state, and the state is only dependent on the combined actions taken by the participants.

The second representation form is the *extended form*. Extended form is appropriate for illustrating turn based games. We don't study any turn based games in this thesis, but it turns out that viewing a simultaneous game as a turn based game can be very helpful in understanding some important game-theoretic concepts. This will be illustrated in section 4.3.3, the minimax theorem.

4.2.3 Utility theory

Utility theory can help us understand why an agent would prefer a certain decision under a certain condition. In utility theory we are concerned with utility functions. Utility functions maps states to real numbers which represent the agents level of happiness in the given state[SL08]. It is reasonable to assume that an agent want to maximize the value of its utility function.

4.2.4 Strategies

The outcome of a game is determined by the combination of each players action [HV04].

Table 4.2: Player1's best replies (underlined) in the Rock, Paper, Scissors game

P1/P2	Rock	Paper	Scissors
Rock	0,0	0,1	<u>1</u>,0
Paper	<u>1</u>,0	0,0	0, 1
Scissors	0,1	<u>1</u>,0	0,0

Pure and mixed strategies

If an agent has n available actions, $a_1, \ldots, a_n$, then sticking to a certain action $a_i, i \in n$ is referred to as a *pure strategy*. If the player instead plays a linear combination of the available actions, $p_1a_1, \ldots, p_na_n$, it is called a *mixed strategy*[HV04, 44]. In fact, we see that a pure strategy is just a special case of a mixed strategy when one of the action probabilities $p_1, \ldots, p_n$ happens to equal 1.

Best reply

The most basic concept is the idea of a *best response*. It is simply defined as the strategy that yields the largest pay-off against a particular choice of action by the adversaries. Table 4.2 illustrates player one's best replies in the well-known Rock, Paper, Scissors game.

If the same reply is the best for every choice, then there exist a best strategy that is independent of the strategy played by the adversaries, such a strategy is referred to as a *dominant strategy*.

4.3 Solution concepts

4.3.1 Dominant strategy solution

In the case where each player has a dominant strategy, that is, a unique best strategy, the game has a *dominant strategy solution*. Few games satisfies this property, but the solution concept is very important in mechanism design where one aims to design games that have dominant strategy solutions[**?**].

4.3.2 Nash equilibrium

Nash equilibrium is the most important solution concept in game theory [HV04]. It applies to a much wider range of games than the dominant strategy solution, in fact Nash proved that every finite game has a Nash equilibrium solution. In a Nash equilibrium neither player is tempted to change strategy. Meaning that, for a certain outcome to be a Nash equilibrium, both players would only lose by deviating. For instance, in the Prisoners Dilemma, both players confessing is a Nash equilibrium. Should either player deviate from confessing, he would be worse off. Nash equilibrium strategies are always rationalizable[SL08].

4.3.3 Maximin

Von Neuman invented a technique for finding the optimal mixed strategy for two-player zero-sum games. As the name implies, it is the strategy that *maximizes* your *minimum* expected payoff. Assuming that the adversary plays an optimal strategy. Player $i's$ maximin strategy is defined as equation 4.1 [NT89][SL08], where u_i is the payoff to player one while a_i and a_j is the, possibly mixed, strategies of player one and two respectively.

$$\arg\max_{a_i} \min_{a_j} u_i(a_i, a_j) \tag{4.1}$$

Similarly, the *minimax* strategy is the strategy that keeps the maximum expected payoff to the adversary at a minimum.

$$\arg\min_{a_i} \min_{a_j} u_j(a_i, a_j) \tag{4.2}$$

It is easier to understand this solution concept if one imagines a turn based game. For instance, each player may ask themselves:

> *"I start by playing a certain strategy with probability* $\mathbf{p}$*, what is my worst outcome?"*

With the assumption of an optimal opponent, we can consider the payoff for every value of p. We will use the two-finger Morra game and an extended version of Rock,paper,scissors to illustrate this.

Two-finger Morra

In this game player one wins if the number of fingers are even and player two wins if the number of fingers are odd, matrix 4.3 illustrates this.

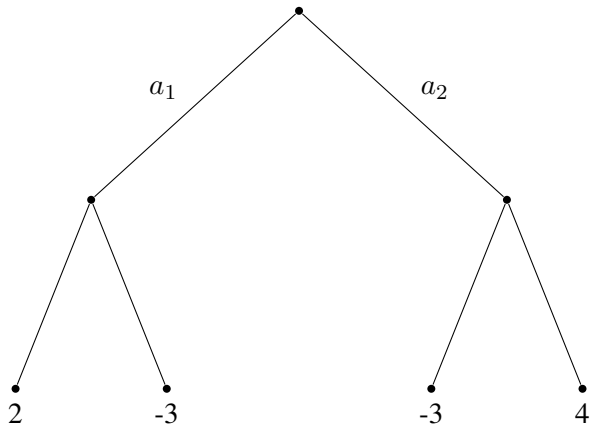
Table 4.3: Payoff matrix for two-finger Morra

P1/P2	C1	C2
R1	2,-2	-3,3
R2	-3,3	4,-4

We call player one the *maximizer* because he follows the maximin strategy. Player two is referred to as the *minimizer*, because his intention is to minimize player one's reward. When player two minimizes player one's gain he actually maximizes his own. Why? Because in a zero-sum game one man's loss is another man's gain.

The maximin strategy is in general in terms of a mixed strategy[SL08]. Consider the case where the maximizer player plays a pure strategy, then the game in extended form is as figure 4.1, where the maximizer makes his move at the root node. For every response of the minimizer, the terminal nodes contains the payoffs to the maximizer.

Figure 4.1: Two-finger Morra in extended form



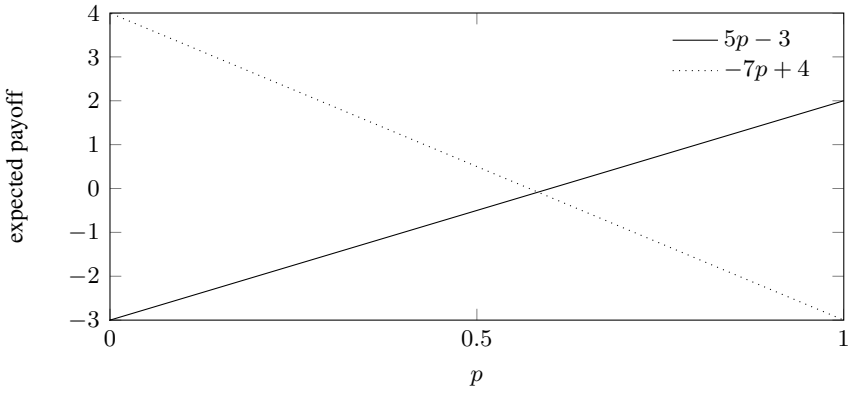
Since we assume that the adversary plays optimally, we see that the maximum expected payoff to player

one is -3 no matter what pure strategy he adopts.

Now, if player one instead considers a mixed strategy $(a_1 : p, a_2 : 1 - p)$, then his expected payoff for a_1 is $(2p + -3(1 - p))q = (5p - 3)q$ where q is the probability that player two also applies a_1. Similarly, for a_2: $(-3p + 4(1 - p))(1 - q) = (-7p + 4)(1 - q)$. We could visualize these in 3d space and solve for the intersection, but here we will make a different point, because it is vital to realize that, in a turn-based game, the second player does not need to play a mixed strategy. If he does play a mixed strategy then his payoff is a linear combination of his payoffs from the pure strategies $pu_1 + (1 - p)u_2$ [RN02]. It is simply impossible for this combination to be better than the best of u_1 and u_2[2]! Hence, he might as well adopt a pure strategy when he knows the choice of p as we assume.

The first player can again reason as follows: "if my adversary responds to my $(a_1 : p, a_2 : 1 - p)$ strategy with a_1 then I get $5p - 3$, and with a_2 as the response I get $-7p + 4$". These lines are plotted in graph 4.2. For instance, for a choice $p = 1$, the payoff to player one is 2 if player two plays strategy a_1 and -3 for strategy a_2. Of course, player two, the minimizer, plays optimally and always choose the response that minimizes player one's payoff, the lower of the two lines. Therefore the best player one can do is to choose p at the intersection, where player two is indifferent about his choices. Here, p is $5p - 3 = -7p + 4 = 7/12$, thus $(a_1 : 7/12, a_2 : 5/12)$ is optimal, and the expected payoff for player one with this mixed strategy is $5 * 7/12 - 3 = -0.083 = -1/12$. Consequently, the zero-sum property implies $1/12$ as the expected payoff to player two.

Figure 4.2: Expected payoff by playing $(a_1 : p, a_2 : 1 - p)$ for the pure responses of player two



[2]The constraint $p \in [0, 1]$ yields the range $[u_1, u_2]$.

A similar derivation would reveal the optimal mixed strategy for player two, but in this case it is unnecessary because of symmetry in the game matrix, and we also have $(a_1 : 7/12, a_2 : 5/12)$ as the optimal mixed strategy for player two. We said earlier that the second player might as well play a pure strategy if he knew the choice of p for the adversary, and indeed, this gives the same expected payoff. However, if the game is played repeatedly this is not beneficial because the opponent could learn and end up adjusting his own strategy to exploit the pure strategy[RN02][SL08][HV04].

Rock, paper, scissors extended

As for the second game we will use an extended version of Rock, paper, scissors. The original version has a mixed strategy equilibrium of $(1/3), (1/3), (1/3)$ which is the same as one get by playing randomly and therefore does not pose the same challenge. In this extended version there are two additional actions, fire and water. Fire beats rock, paper and scissors, but loses to water. Water loses against rock, paper and scissors, but wins against fire. This is illustrated by table 4.4, the payoff matrix.

Table 4.4: An extended variant of the well-known Rock, paper, scissors game

P1/P2	Rock	Paper	Scissors	Fire	Water
Rock	0	-1	1	-1	1
Paper	1	0	-1	-1	1
Scissors	-1	1	0	-1	1
Fire	1	1	1	0	-1
Water	-1	-1	-1	1	0

Using the same approach as for two-finger Morra, where player one considers the expected payoff for the different pure strategy choices of player two, we can determine the optimal mixed strategy. Denoting the probability of playing rock r, paper p, scissors s, fire f and water w we get the following equations for the expected payoffs.

$$\begin{aligned} Rock &= -p + s - f + w \\ Paper &= r - s - f + w \\ Scissors &= -r + p - f + w \\ Fire &= r + p + s - w \\ Water &= -r - p - s + f \end{aligned} \tag{4.3}$$

By applying linear algebra we can solve for the intersection of the hyperplanes. Matrix form and row reducing yields:

$$\begin{bmatrix} 0 & -1 & 1 & -1 & 1 & R \\ 1 & 0 & -1 & -1 & 1 & P \\ -1 & 1 & 0 & -1 & 1 & S \\ 1 & 1 & 1 & 0 & -1 & F \\ -1 & -1 & -1 & 1 & 0 & W \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 & -\frac{1}{3} & 0 \\ 0 & 1 & 0 & 0 & -\frac{1}{3} & 0 \\ 0 & 0 & 1 & 0 & -\frac{1}{3} & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & W \end{bmatrix}$$

First we notice that the expected payoff is zero. Furthermore, $r - \frac{1}{3}w = 0$ so p equals $\frac{1}{3}w$, similarly for p and s, while $f = w$. Applying the constraint $p + r + s + f + w = 1$, we find $3 * \frac{1}{3}w + w + w = 1 \longleftrightarrow w = 1/3$, which means that the optimal mixed strategy is $(r : 1/9, p : 1/9, s : 1/9, f : 1/3, w : 1/3)$. Again, by symmetry, the same strategy is optimal for player two.

In both these games the minimax solution coincides with the Nash Equilibrium. This is always the case for zero-sum games, but not for non-zero sum[HV04, 43].

4.4 Games investigated in this thesis

4.4.1 The Prisoner's Dilemma

The Prisoner's Dilemma is a paradox that has received a lot of attention, especially in social sciences. It fascinates social scientists because, even though both players behave rational, the outcome is sub-optimal

for all[HV04]. To see why, consider the utility payoffs in the following matrix, which were also shown in the beginning of this chapter.

Table 4.5: The Prisoners Dilemma

A/B	Refuse	Testify
Refuse	A=-0.6,B=-0.6	A=-10, B=0
Testify	A=0,B=-10	A=-2,B=-2

Obviously, the best outcome would be for both players to refuse. However, a rational player only interested in maximizing its own utility would reason as follows: "If my adversary decides to confess I get 2 years by confessing and 10 years if I deny, so in that case confessing is better. On the other hand, if my adversary decides to defect then I get 0 years for confessing and 6 months if I defect, so in that case confessing is better". Similarly, if the other player is rational he would reach the same conclusion and both ends up confessing. This is the paradox of the Prisoners Dilemma.

The PD has also been studied extensively from a computer science perspective. Perhaps most known is Robert Axelrods tournament, described in his book "The Evolution of Cooperation", where he invited game theorists to submit an algorithm to compete in a Prisoners Dilemma tournament. The tournament consisted of every scheme playing 200 rounds against all others over an average of 5 repetitions[HV04].

Surprisingly, one of the simplest schemes, known as *Tit-for-Tat* won the tournament. Tit-for-tat starts by a cooperative move and on the subsequent rounds it does what the opponent did the round before.

Later, Axelrod hosted the tournament once more, this time the number of participants where must larger and the number of rounds where no longer fixed. However, *Tit-for-tat* won this time too.

We intend to put the KBLA and the other bandit schemes from chapter 2 up against each other in a Prisoner's Dilemma tournament, but in contrast to Axelrods tournament we are not so interested in the score, but rather embraces rational behavior. A rational agent is a predictable agent, which is a vital property in software agents[RN02].

4.4.2 Goore game

The Goore game, also referred to as "Gur game", is the only game we have included that involves teams of players. It is also special in the sense that it contains both cooperative and non-cooperative elements. It is cooperative in the sense that there is a team involved, but none of the players actually know that they are part of a team and as such only plays to maximize their own reward as in a non-cooperative setting.

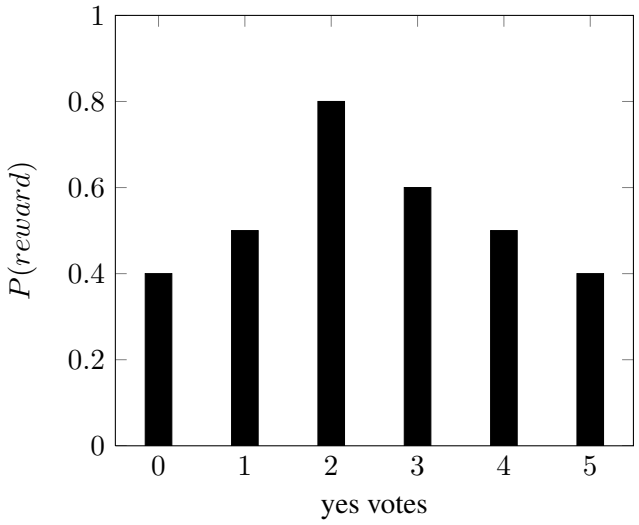
[NT89] describes the game in the following way:

> "Imagine a large room containing N cubicles and a raised platform. One person (voter) sits in each cubicle and a referee stands on the platform. The referee conducts a series of voting rounds as follows. On each round the voters vote yes or no (the issue is unimportant) simultaneously and independently (they do not see each other) and the referee counts to the fraction θ of yes votes. The referee has a unimodal[3] performance criterion $g(\theta) \in [0, 1]$, which is optimized when the fraction of yes votes is exactly θ^*. The current voting rounds ends with the referee awarding a dollar with probability $g(\theta)$ and assessing a dollar with probability $1 - g(\theta)$ to every voter independently. On the basis of their individual gains and losses, the voters then decide, again independently, how to cast their votes next round."

As we see, this equals a binary bandit problem for each player where the probability of receiving a reward is $1 - g(\theta)$. However, since the Kalman Bayesian learning automaton assumes normally distributed feedback we have also created a version of the Goore game with such feedback. Relating to the description above, our version has the performance criterion $g(\theta) \in [0, A]$ where A is the amplitude of Gaussian function. And instead of assessing a reward with probability $1 - g$ the reward to each player is instead sampled from $N(\mu, \sigma)$ where μ is given by the proportion of players that vote yes θ.

Figure 4.4.2 illustrates a case with five players where the referee's probability of reward is highest when two players vote yes. With two yes votes the probability is 0.8, and as such the goal for the players is to learn that two yes votes are optimal, without any communication at all.

[3] Only one local maxima.



Our motivation for empirically evaluating the Kalman Bayesian Learning Automaton in the Goore game is due to its distributed nature, and the fact that it has been studied from practical settings before (Quality of Service in [IK03], wing optimization in [HNP$^+$03] and distributed control in [TK93]). Since there is no communication between players the Goore game is resolved in a completely distributed manner.

It has been shown that a team of Tsetlin automata with large memory can solve the game if the number of rounds is sufficiently large[NT89]. However, from a practical viewpoint it is just as important to quickly reach near-optima instead of using a long time to solve the game exactly.

4.4.3 Zero-sum games

Zero-sum are games where the payoffs, as the name implies, sums to zero. Should the payoff to one participant equal ten, then the other participant must suffer a loss of the same amount, that is the fundamental property of zero-sum games. Thus, when we represent zero-sum games in matrix form each cell sums to zero. For normally distributed payoff matrices the value in each cell is the mean value μ of a normal distribution $N(\mu, \sigma)$, while the matrices with binary feedback contains the probability of a reward $P(1)$ in its cells. However, note that the sum of the *actual feedback received* might be different from zero in normally distributed zero-sum games because of noise.

Zero-sum games has been studied extensively from a computer science perspective before and according

to [NT89] it serves as a valuable benchmark for testing rationality of learning automata. In this thesis we consider both zero-sum games with normally distributed payoff and zero-sum games with binary feedback are considered .

The solution to a zero-sum game might be in pure strategies or mixed strategies. For our benchmark we have used two games with pure strategy solutions and two with mixed strategies. The pure strategy games has been created in a somewhat similar way to how it was done by [NT89] in their zero-sum analysis, and should emphasize specific aspects of the learning algorithms. In the first pure strategy game both players have dominating strategies, while in the second pure strategy game only player one has a dominating strategy. The mixed strategy games are the same as we derived the minimax solution for earlier, the two-finger Morra game and, for the scalability aspect, an extended version of Rock, paper, scissors.

Chapter 5

Experiments

This chapter contains an empirical evaluation of the Kalman Bayesian Learning Automaton. The performance in terms of regret and action probabilities are compared between the Kalman Bayesian Learning Automaton and the other bandit schemes introduced in chapter 2. Preceding the comparison is a parameter analysis of the KBLA to illustrate how the sensor variance parameter and update variance parameter may be used to balance exploration and exploitation.

The schemes are compared in random walk environments, switching environments, the Goore game and zero-sum games. Both normally distributed feedback and binary feedback are considered. Naturally, since our focus is on the KBLA, many experiments are performed with the KBLA alone with the intention to emphasize certain aspects of its behavior. For instance, the Goore game is tested with a wide range of parameters for the KBLA to understand how to control the amount of exploration, while the two-finger Morra zero-sum game is tested against an optimal opponent. We also include results for the Exploration vs Exploitation challenge, a competition which is a direct commercial application of the multi-armed bandit problem and involves both stationary and non-stationary binary environments.

5.1 Performance measures and accuracy

The two most fundamental performance measures we have used is the 'probability of a selecting a certain action' and 'regret'. The first consists of keeping track of how many times the automaton selected a certain

action during an experiment. In general, we are often most interested in how the *probability of selecting the optimal action* develops over time. This enables us to observe for instance learning speed and points in time where performance is dropping rapidly.

Regret is a but popular measure of success in the exploration vs exploitation dilemma [ACF02]. It is a measure of *the difference between the sum of rewards actually gained and what one would have gained by only playing the optimal action*. In this way it is a little more "forgiving" than the "probability of selecting the optimal action" measure because choosing the next best action might not be very regretful in many cases. If we denote the reward of the best action at a time step t, r_t^*, and the reward from the action actually played at t, r_t, then the regret after T time steps is:

$$\sum_{t=1}^{T} r_t^* - r_t \tag{5.1}$$

Due to the stochastic elements in the experiments these measures are noisy and thus rather imprecise without an average, which brings us to our second point, accuracy.

Most of our experiments is performed in a sequence of interactions with an environment, performed over *n* discrete time steps, where each sequence is repeated 1000 times. When computing the results for a specific sequence we consider the entire collection of repetitions, the *ensemble*. From this we calculate the *ensemble average*. Our motivation for using the ensemble average is due to randomness. We only control the simulation at a macroscopic level, not the microscopic details. An example of a microscopic detail is the value sampled from a normal distribution. This value is random, so there is certain amount of "luck" involved in regard to whether this value is increasing the performance of the algorithm at a certain point in time. Therefore, every iteration can, and will, yield a different result.

To obtain an accurate measure from these fluctuations caused by the uncontrollable microscopic elements, we are interested in the mean value, which is exactly what the ensemble average represents.

As an example, table 5.1 shows a calculation of the ensemble average for each time step in a case with just 4 repetitions. The rows contain the repetitions, while each column is a certain point in time. The values in each cell is action selected at that point in time for the repetition. For instance, repetition 1 selected arm 1 on the first and fourth time step while on repetition 4 it selected this arm on all time steps.

Table 5.1: Table illustrating the ensemble average, the average of a column.

Repetition/Time	t1	t2	t3	t4
r1	1	0	0	1
r2	0	0	0	1
r3	0	1	1	0
r4	1	1	1	1
Ensemble average	2/4	2/4	2/4	3/4

5.2 Parametric analysis

It is important to understand how different numerical values for the parameters of the Kalman Bayesian Learning Automaton influence its decisions. We know that during the arm selection process, the sampled value is determined by the mean and variance of the arms. Arms with high variance will give values in a greater range, and thus the effect is that the arm selection process has more "noise" and the probability of selecting another arm is higher.

As we saw in chapter 3, the mean and variance of each arm is given by the following recursive equations:

$$\mu_{t+1} = \frac{(\sigma_t^2+\sigma_x^2)z_{t+1}+\sigma_z^2\mu_t}{\sigma_t^2+\sigma_x^2+\sigma_z^2}$$

$$\sigma_{t+1}^2 = \frac{(\sigma_t^2+\sigma_x^2)\sigma_z^2}{\sigma_t^2+\sigma_x^2+\sigma_z^2}$$

We refer to σ_z^2 and σ_u^2 as the *sensor variance parameter* and the *update variance parameter* respectively. The sensor variance parameter is supposed to match the noise of the observations in the environment, while the update variance parameter is the automatons belief of how the environment changes each time step.

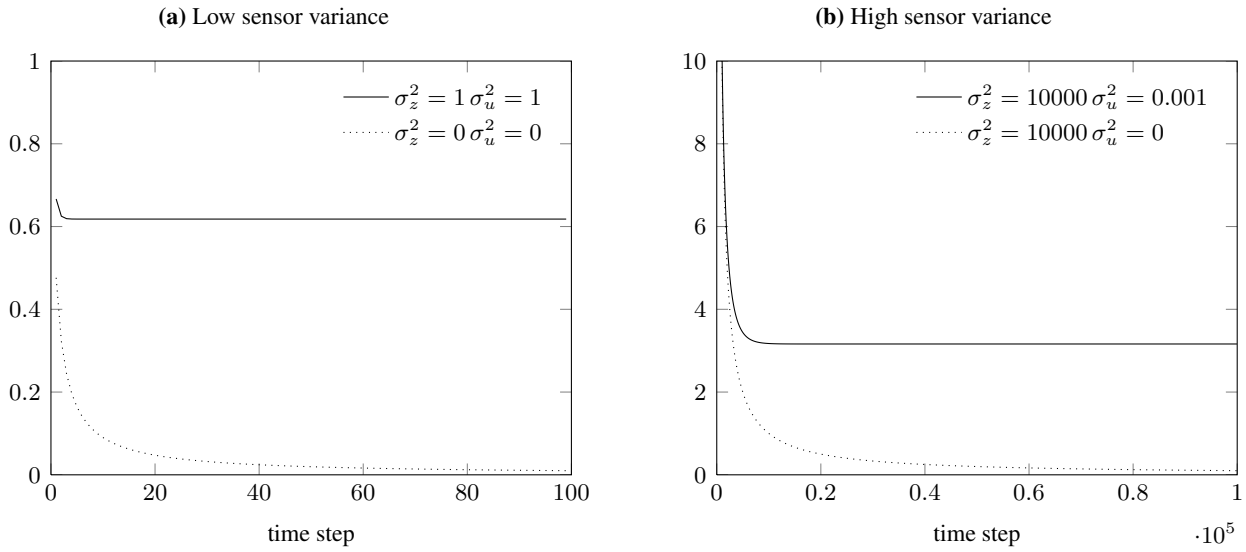
The development of the variance[1] over time for some combinations of the sensor variance parameter and update variance parameter is plotted in figure 5.1.

From the figure we note several interesting properties.

- It seems that an update variance parameter of zero causes the variance to approach 0. This is in conformance with prior beliefs, an update variance parameter of zero implies that the tracked object is stationary and therefore its position is deterministic.
- A nonzero update variance parameter on the other hand causes the the variance to converge to a certain value. Again, this seems reasonable, one can never be completely sure about the location of a randomly moving object. In bandit terms, this implies that the update variance parameter decides the rate of constant exploration.

[1]The variance is initially set to a very high value as to not exclude any true value of the mean.

Figure 5.1: Variance development over time



- A high sensor value causes the variance to reduce at a much slower rate than a low sensor value. From a bandit point of view a Kalman Bayesian Learning Automaton with high sensor variance parameter explores more in the beginning.

Also note the very quick convergence of the variance when the sensor variance parameter is small with a nonzero update variance parameter. The array below shows the variance value for the 50 first time steps when both the sensor and update equals 1.

1	2	5	10	20	50
10000	0.9999	0.619047	0.618034	0.618034	0.618034

We can also see this with the variance formula, where the fixed point is given by:

$$\begin{aligned}
\sigma^2 &= \frac{(\sigma^2+\sigma_u^2)\sigma_z^2}{\sigma^2+\sigma_u^2+\sigma_z^2} = \frac{\sigma^2\sigma_z^2}{\sigma^2+\sigma_u^2+\sigma_z^2} + \frac{\sigma_u^2\sigma_z^2}{\sigma^2+\sigma_u^2+\sigma_z^2} \\
0 &= \sigma^2 - \frac{\sigma^2\sigma_z^2}{\sigma^2+\sigma_u^2+\sigma_z^2} - \frac{\sigma_u^2 * \sigma_z^2}{\sigma^2+\sigma_u^2+\sigma_z^2} \\
0 &= \sigma^2(\sigma^2+\sigma_z^2+\sigma_u^2) - \sigma^2\sigma_z^2 - \sigma_u^2\sigma_z^2 \\
0 &= \sigma^4 + \sigma^2\sigma_u^2 + \sigma^2\sigma_z^2 - \sigma^2\sigma_z^2 - \sigma_u^2\sigma_z^2 = \sigma^4 + \sigma^2\sigma_u^2 - \sigma_u^2\sigma_z^2
\end{aligned}$$

And since σ_u^2 and $\sigma_z^2 =$ are constants, we can apply the quadratic formula $ax^2+bx+c=0 \longleftrightarrow x = \frac{-b\pm\sqrt{b^2-4ac}}{2a}$ where $x=\sigma^2$, $a=1$, $b=\sigma_u^2$ and $c=\sigma_u^2\sigma_z^2$. Remembering that the variance must be positive, we end up with equation 5.2.

$$\sigma^2 = \frac{-\sigma_u^2 + \sqrt{\sigma_u^4 + 4\sigma_u^2\sigma_z^2}}{2} \tag{5.2}$$

If we apply the values $\sigma_u^2 = 1.0$ and $\sigma_z^2 = 1.0$ we see that it coincides with the value in the table above. ($\sigma^2 = \frac{-1+\sqrt{5}}{2} = 0.618034$). It also confirms the discussed properties above like converging to zero when the update variance parameter approaches 0.

$$\lim_{\sigma_u^2 \to 0} \frac{-\sigma_u^2 + \sqrt{\sigma_u^4 + 4\sigma_u^2\sigma_z^2}}{2} = 0$$

5.3 Multi-armed bandit problems

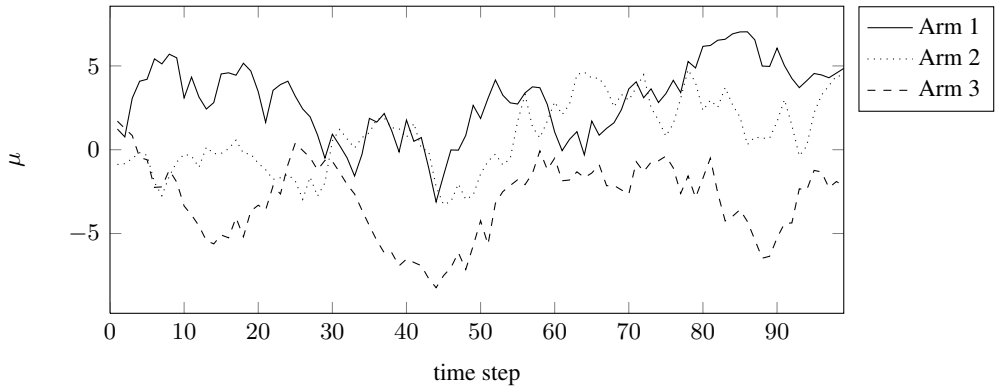
5.3.1 Types of environments

For the general multi-armed bandit problem we have used two types of environments.

Random walk environment: The recursive equations from section 3.6 was derived from a Gaussian distributed random walk, and since they form the basis of the Kalman Bayesian Learning Automaton we say that a random walk environment is exactly the type of environment the KBLA is designed for. The parameters of the environment, the observation noise σ_z^2 and update model σ_u^2, matches the parameters of the automaton.

Therefore, in an environment like this, the value of an "arm" i changes each time step with a value sampled from a standard normal distribution $N(0, \sigma_u^2)$ and the feedback from an action is given from $N(\mu_i, \sigma_z^2)$. An example of how a random walk environment may change over time can be seen in figure 5.2. Each experiment is repeated 1000 times, where each repetition runs a different random walk environment over *n* time steps. Thus, this gives an indication of the KBLA's average performance on such randomly changing environments.

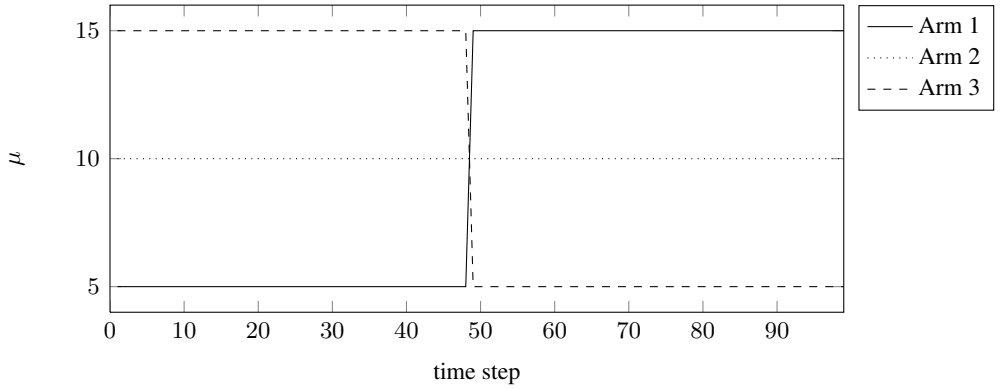
Figure 5.2: Example of random walk environment with 3 arms



Switching environment: These environments contain both stationary and dynamic elements. The reward distributions of each arm are constant. However, the arms interchange their values at

certain intervals as can be seen in figure 5.3. Switching environments are of interest because they involve both the ability to adapt to sudden changes and the ability to track a stationary best arm without to much performance loss.

Figure 5.3: Example of switching environment with 3 arms



5.3.2 Results random walk environments

Our test configurations for random walk environments consists of 2-armed, 10-armed and 50-armed problems with two different degrees of noise and two different degrees of update speed, as seen in table 5.2. Higher noise indicates a tougher environment because the arms are harder to distinguish when the value distributions of each arm are wider. A higher update speed on the other hand might both make the environment easier and harder. Easier because the difference between the mean values in the reward distributions will most likely be greater, but then again, a more rapidly changing environment can also increase the amount of times the index of the optimal arm changes.

Also note that in all these configurations the Kalman Bayesian Learning Automaton is given the correct parameters of the environment. Later, in the last part of the chapter, we will consider random walk environments where the parameters given to the KBLA are incorrect.

In table 5.3 we see the results for the test with just two arms and low noise. The Kalman Bayesian Learning Automaton is clearly superior in this type of environment, at least when given the correct parameters as

Table 5.2: Random walk configurations

Arms	Observation noise (Sensor variance) σ_z^2	Transition noise (Update variance) σ_u^2
2	1, 100	1, 100
10	1, 100	1, 100
50	1, 100	1, 100

here. None of the other schemes are able to reach the same accuracy. Many of them are hindered by their inertia when another arm becomes the best, but even dynamic schemes like AdaptivePursuit, GreedyEWA, SoftMaxEWA and AdaptEvE-MetaBandit does not come close to the performance of the KBLA.

Table 5.3: Results for two-armed environment with observation noise $\sigma_z^2 = 1.0$ and transition noise σ_u^2 = 1.0

Player / Iteration	**10**	**100**	**1000**	**2000**	**10000**
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 1.0$	0.74	0.9	0.97	0.97	0.99
BLA Normal Known σ_z^2	0.74	0.78	0.78	0.76	0.77
Poker	0.7	0.77	0.78	0.77	0.76
Pursuit 0.005	0.5	0.57	0.65	0.65	0.58
AdaptivePursuit α =0.8,β =0.8, P_{min} =0.01	0.55	0.64	0.80	0.83	0.85
AdaptivePursuit α =0.1,β =0.1, P_{min} =0.01	0.57	0.68	0.75	0.79	0.82
Greedy ϵ =0.05	0.62	0.7	0.71	0.74	0.71
Greedy ϵ =0.02	0.61	0.65	0.71	0.74	0.73
ϵ_n-Greedy $c = 1$, $d = 0.1$	0.52	0.5	0.67	0.7	0.68
GreedyEWA ϵ =0.05 α =0.3	0.66	0.74	0.79	0.84	0.84
GreedyEWA ϵ =0.01 α =0.3	0.62	0.61	0.75	0.78	0.83
SoftMaxEWA τ=0.5 α=0.8	0.70	0.82	0.83	0.84	0.83
SoftMaxEWA τ=0.1 α=0.9	0.78	0.85	0.86	0.88	0.89
UCBNormal	0.49	0.77	0.75	0.78	0.75
AEMetaBandit $\delta = 0.005$, $\lambda = 5.0$, $MT = 100$	0.62	0.78	0.86	0.87	0.88

The difference becomes even more apparent if we increase the number of arms. Table 5.4 shows the results for the case with 50 arms.

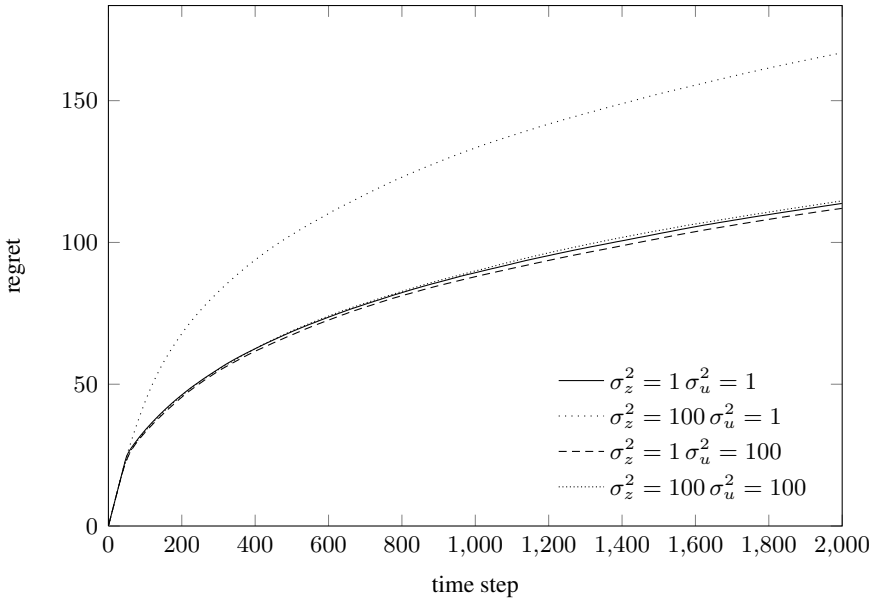
Table 5.4: Results for fifty-armed environment with observation noise $\sigma_z^2 = 1.0$ and transition noise σ_u^2 = 1.0

Player / Iteration	**10**	**100**	**1000**	**2000**	**10000**
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 1.0$	0.02	0.43	0.79	0.85	0.94
Poker	0.03	0.25	0.15	0.13	0.10
Pursuit 0.005	0.02	0.06	0.18	0.14	0.09
AdaptivePursuit α =0.8,β =0.8, P_{min} =0.01	0.04	0.17	0.40	0.43	0.48
AdaptivePursuit α =0.1,β =0.1, P_{min} =0.01	0.02	0.07	0.17	0.24	0.40
Greedy ϵ =0.1	0.02	0.20	0.24	0.27	0.32
Greedy ϵ =0.05	0.02	0.21	0.22	0.27	0.31
ϵ_n-Greedy $c = 4$, $d = 0.1$	0.02	0.02	0.21	0.22	0.19
GreedyEWA ϵ =0.1 α =0.9	0.02	0.28	0.47	0.59	0.78
GreedyEWA ϵ =0.05 α =0.9	0.02	0.26	0.36	0.49	0.76
GreedyEWA ϵ =0.01 α =0.3	0.02	0.17	0.13	0.14	0.28
SoftMaxEWA τ=0.5 α=0.3	0.02	0.04	0.06	0.05	0.05
SoftMaxEWA τ=0.2 α=0.9	0.02	0.15	0.17	0.16	0.14
UCBNormal	0.02	0.02	0.02	0.46	0.34
AEMetaBandit $\delta = 0.005$, $\lambda = 5.0$, $MT = 500$	0.01	0.02	0.26	0.34	0.38

The GreedyEWA scheme is the best of the other schemes by far. However, we should note that the AdaptivePursuit scheme also seems close to its maximum performance. Due to the way it works, its maximum performance drops with the number of actions, and from the description of the algorithm in section 2.3.3 we can see that $P_{max} = 1 - (K - 1)P_{min} = 1 - (50 - 1) * 0.01 = 0.51$. AEMetaBandit also performs decent. Even though it needs a change of reward in the negative direction, the low choice 5.0 for the λ parameter ensures a lot of exploration.

To see how the parameters of the environment affects the performance of the Kalman Bayesian Learning Automaton figure 5.4 shows normalized regret for four parameter combinations. We see that only when the sensor variance (observation noise) is high compared to the update variance (transition noise) the performance drops significantly. Both high sensor variance and high update variance does not influence the performance, thus it seems likely that a higher update variance parameter actually makes the environment somewhat easier due to the increased difference between the mean values of the reward distributions in the arms. The slightly lower regret on the environment with only high update variance also supports this.

Figure 5.4: Regret for KBLA in fifty armed random walks with varying difficulty



5.3.3 Results switching environments

The configurations for the switching environments are shown in table 5.5 and 5.6. In the first case the value of the best arm switches with an inferior arm, and thus favors algorithms designed to increase exploration when detecting rapid changes in the current exploited arm. In the second case the value of an inferior arm increases while the previously best arm does not change, and is thus harder to detect unless regular exploration is used.

Table 5.5: Switching environment configuration 1

Arm:	1	2	3	4	5	6	7	8	9	10
Mean value:	5	10	15	20	25	30	35	40	45	50
Comment:	Arm 5 and 10 interchange values									

Table 5.6: Switching environment configuration 2

Arm:	1	2	3	4	5	6	7	8	9	10
Mean value:	5	10	15	20	25	30	35	40	45	50
Comment:	Arm 5 alternates between 25 and 55									

In contrast to random walk environments, the update variance of a switching environment is unknown to the Kalman Bayesian learning automaton, and therefore several parameters have been tested in order to learn more about its behavior. The results for the first configuration are shown in table 5.7.

AdaptEvERestart is definitely superior in this test. Unsurprisingly, the change point detection mechanism is very strong in environments like this. Far behind AdaptEvERestart, but still performing adequately, we find the exponential recency weighted average schemes, and we see that especially the "aggressive" GreedyEWA performs well.

To understand how the schemes reacts to changes in the environment, figure 5.5 shows how the probability of selecting the optimal arm develops after the first change point ($t = 1000$).

Here we can clearly see the rapid convergence and quick adaption of AdaptEvERestart scheme. The other

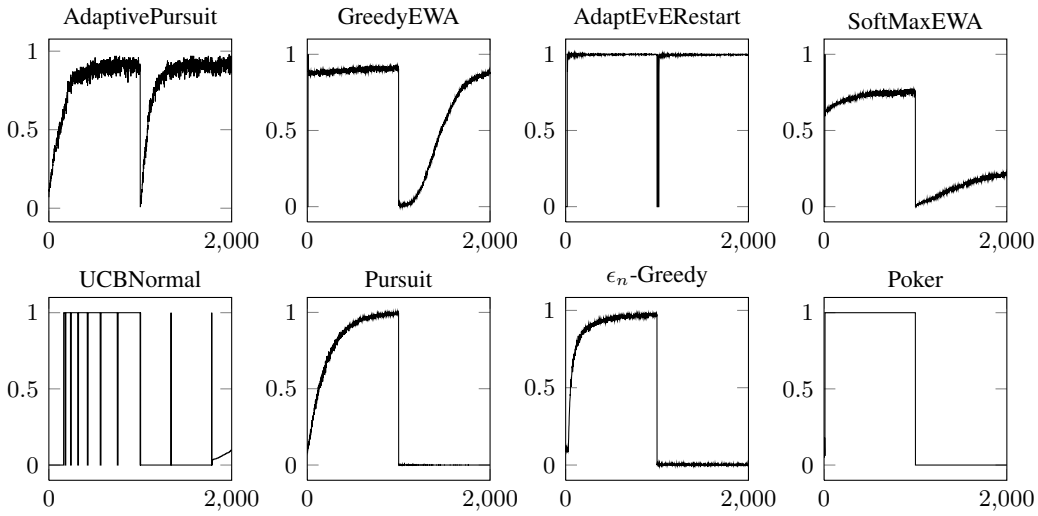
Table 5.7: Regret for switching environment configuration 1 with interchanging every 1000 time steps

Player / Iteration	**10**	**100**	**1000**	**2000**	**10000**
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.0$	4.27	5.11	5.12	235.74	1124.66
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.01$	4.27	5.12	5.45	119.28	821.28
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.1$	4.27	5.25	9.56	53.29	322.87
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.2$	4.24	5.39	14.26	42.08	270.35
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.5$	4.25	5.87	25.66	57.77	311.94
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 1.0$	4.28	6.65	39.7	82.32	423.61
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 2.0$	4.28	8.27	61.23	123.67	621.63
Poker	5.0	5.48	5.48	242.88	1131.82
Pursuit 0.005	4.97	40.14	101.5	390.32	1278.8
AdaptivePursuit α =0.8,β =0.8, P_{min} =0.05	4.01	27.79	252.96	505.05	2519.31
AdaptivePursuit α =0.3,β =0.3, P_{min} =0.02	4.07	27.37	134.0	255.2	1232.48
Greedy ϵ =0.1	5.0	9.51	54.23	308.01	1508.0
Greedy ϵ =0.05	5.0	7.22	29.72	272.19	1316.79
ϵ_n-Greedy $c = 1$, $d = 0.1$	4.99	49.81	499.6	885.36	2399.07
GreedyEWA ϵ =0.1 α=0.3	5.0	10.22	57.2	155.56	913.64
GreedyEWA ϵ =0.05 α =0.9	5.0	7.17	29.58	72.65	438.39
SoftMaxEWA τ=0.5 α=0.3	5.0	34.65	341.13	684.13	3429.28
SoftMaxEWA τ=0.1 α=0.3	5.0	10.18	44.24	185.56	915.52
UCBNormal	9.89	71.33	121.0	415.05	1200.07
AEγRestart $\delta = 0.005$, $\lambda = 5.0$	7.78	10.09	10.21	21.0	107.48

dynamic schemes is not able to get the same accuracy, and does not adapt as quickly. The stationary schemes identifies the best arm in the beginning, but then none of them see to overcome their inertia at all.

Surprisingly, even though the AdaptivePursuit is better at tracking changes in the environment, we see that it is almost outperformed by the Pursuit scheme in terms of total regret, even though Pursuit is much slower to adapt. If we think about the environment works, it makes perfect sense. The distance between the best and next best arm is relatively small and therefore choosing the next best arm has no drastic impact on regret. Thus, the Pursuit scheme probably ends up selecting the next best arm a while after the change point because this action received the next highest probability during the initial "learning"(before the first change point) phase.

Figure 5.5: Dynamic and stationary schemes in a switching environment



Looking at the results for the KBLA, we see that it performs best for update parameter 0.2, but shows great variation in performance for different values of the update parameter. How the update variance parameters affects the automaton in switching environment configuration 1 is shown in graph **??**. Clearly, the choice of update parameter is a compromise between accuracy and adapting speed. We see that if the update parameter is too small the automaton is not able to adapt when the optimal arm suddenly changes. However, a too high update parameter drops the maximum precision it is able to get.

Table 5.8 shows regret for the same environment with the switching frequency increased. Not surprisingly, the best update variance parameter is slightly higher when switching more frequently. That is, for 1000 the best is around 0.2 while for 250 it seems to be around 0.5.

We end of this section with the results for regret in switching environment configuration 2 in table 5.9. Only the adaptive schemes together with the KBLA is shown. Now that there is no change in the best option we see that the AdaptEvE scheme is not able to obtain the same performance as some configurations of the KBLA and GreedyEWA.

Figure 5.6: The compromise between learning speed and accuracy

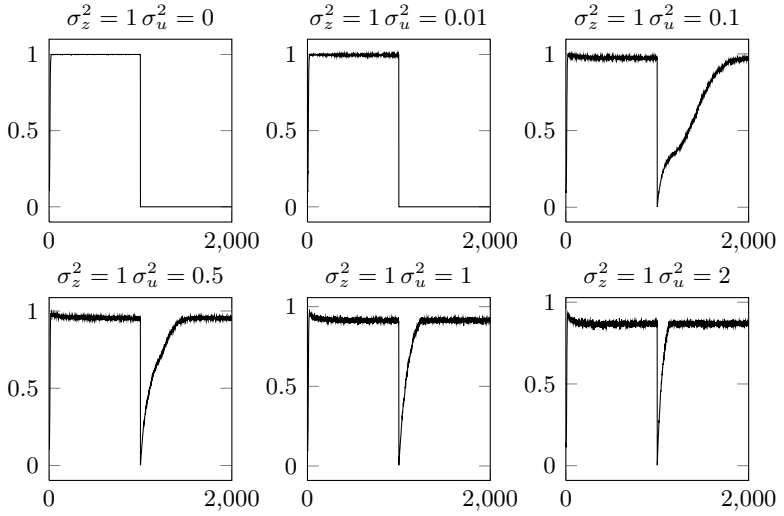


Table 5.8: Regret for switching environment configuration 1 with interchanging every 250 time steps

Player / Iteration	**10**	**100**	**1000**	**2000**	**10000**
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.0$	4.24	5.11	118.77	229.89	1118.79
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.01$	4.25	5.12	88.02	195.23	1022.85
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.1$	4.27	5.25	76.64	158.62	815.97
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.2$	4.25	5.38	56.92	126.76	679.97
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.5$	4.25	5.86	49.11	101.36	518.98
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 1.0$	4.29	6.68	56.97	116.91	598.6
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 2.0$	4.31	8.29	71.86	144.59	726.94

Table 5.9: Regret for switching environment configuration 2 with interchanging every 1000 time steps

Player / Iteration	**10**	**100**	**1000**	**2000**	**10000**
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.0$	5.0	5.0	5.01	105.01	505.01
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.01$	5.0	5.01	5.36	105.89	479.83
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.1$	5.0	5.12	9.41	65.96	326.83
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.2$	5.0	5.28	14.22	49.48	238.98
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.5$	5.0	5.75	25.52	58.59	286.3
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 1.0$	5.0	6.54	39.67	79.71	392.46
AdaptivePursuit α =0.8,β =0.8, P_{min} =0.05	4.13	27.89	252.51	497.45	2484.91
AdaptivePursuit α =0.3,β =0.3, P_{min} =0.02	3.9	27.36	132.96	250.19	1131.26
GreedyEWA ϵ =0.1 α =0.9	5.0	9.45	54.62	111.06	541.45
GreedyEWA ϵ =0.05 α =0.9	5.0	7.24	29.44	76.46	352.01
SoftMaxEWA τ=0.5 α=0.3	5.0	34.7	343.59	669.19	3341.27
SoftMaxEWA τ=0.1 α=0.3	5.0	11.47	58.62	193.7	903.93
AEγRestart $\delta = 0.005$, $\lambda = 5.0$	7.78	10.09	10.22	104.48	492.14
AEMetaBandit $\delta = 0.005$, $\lambda = 5.0$, $MT = 50$	7.78	10.1	10.27	107.46	502.53

5.4 Goore game

As we mentioned in section 4.0, where we introduced the Goore game, we have created both normally distributed and binary feedback functions for the Goore game. The functions have been adapted to accept a percentage of players that vote yes. This enables us to use the same function for different team sizes. Two properties of the function should affect the outcome, the shape and amplitude. Two shapes have been chosen, one flat and one narrow, while the amplitudes are set to 20 and 400. The complete set of feedback functions can be seen in table 5.10 while figure 5.7 shows the shape of the Gaussian functions. Note that the optimal percentage of yes votes is chosen to not be at the center of the graph because of the likeliness of 50% yes when selecting actions at random.

Figure 5.7: The Goore game functions used for normally distributed feedback

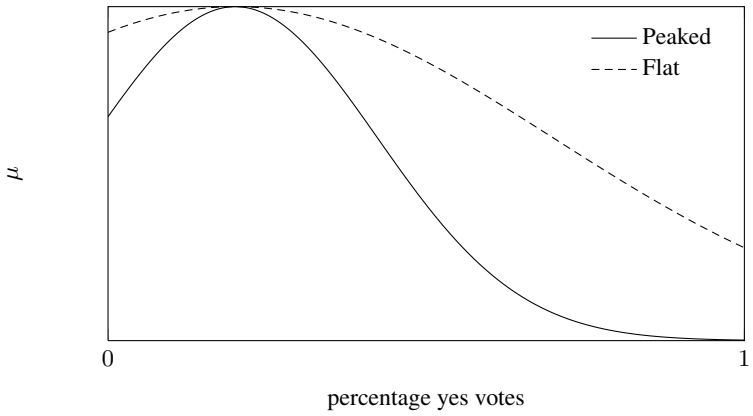


Table 5.10: Goore game configurations

Config	Feedback type	Function	Optimal yes	Team size
1	Gaussian with noise $\sigma^2 = 1.0$	Flat Gaussian	20%	10 and 100
2	Gaussian with noise $\sigma^2 = 1.0$	Peaked Gaussian	20%	10 and 100
3	Binary	Function from [IK03]	80%	10 and 100

5.4.1 Goore game results

The results for 10 player Goore game is shown in table 5.11.

Table 5.11: Results for 10 player Goore game on peaked Gaussian function with amplitude 20 and where 2 yes votes are optimal.

Team / Iteration	**10**	**100**	**1000**	**2000**	**10 000**	**100 000**
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 1.0$	2.64	2.77	2.67	2.67	2.64	2.64
ϵ_n-Greedy $c = 0.3$, $d = 0.1$	4.99	1.77	1.1	1.15	1.09	1.18
ϵ_n-Greedy $c = 0.6$, $d = 0.1$	4.81	3.16	0.34	1.12	1.04	1.02
Greedy ϵ =0.02	0.09	0.48	2.12	2.11	2.1	2.06
GreedyEWA ϵ =0.02 α =0.3	0.23	0.32	2.11	2.2	2.16	2.23
SoftMaxEWA τ=0.5 α=0.3	2.55	4.17	4.11	4.17	4.17	4.18
SoftMaxEWA τ=0.1 α=0.3	0.0	0.0	0.38	0.78	1.88	2.77
Pursuit 0.01	4.81	2.19	1.02	1.08	1.08	1.08
Pursuit 0.001	4.89	4.65	1.91	0.69	1.3	1.3
AdaptivePursuit α =0.8,β =0.8, P_{min} =0.01	4.25	2.65	2.28	2.41	2.36	2.38
UCBNormal	3.33	1.0	0.2	0.0	0.1	0.01
AEMetaBandit $\delta = 0.005$, $\lambda = 10$, $MT = 100$	0.0	0.12	1.32	1.81	2.36	2.45

None of the schemes are able to solve the game completely with this parameter configuration. The simple Greedy scheme is very close, and it is surprising that it slightly outperforms the GreedyEWA scheme which should be more suited for a dynamic environment like this. The SoftMaxEWA τ=0.5 obviously does too much exploration because it is far away from the solution, but even with lower exploration it is not able to get particularly close to the optimal solution. The KBLA quickly gets close to the optimal solution, but then it seems to stop, because it does not get any better over time.

Increasing the team size to 100 players makes the problem harder because it increases the amount of noise in the environment, in addition the distance between rewards are smaller. Table 5.12 contains the results for teams of 100 players on the peaked function.

We see that most schemes has problems when the number of players increases to 100. Only the ϵ_n-Greedy $c = 0.6$ scheme is close to the optimal solution, but the performance seems to be very sensitive to its parameter values. The simplest scheme of them all, the ϵ-Greedy scheme and the AdaptEvEMetaBandit scheme are also quite close after 100 000 iterations. The Kalman Bayesian Learning Automaton does not

Table 5.12: Results for 100 player Goore game on peaked Gaussian function with amplitude 20 and where 20 yes votes are optimal.

Team / Iteration	**10**	**100**	**1000**	**2000**	**10000**	**100000**
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 1.0$	43.34	41.25	41.81	41.93	41.92	41.83
ϵ_n-Greedy $c = 0.3$, $d = 0.1$	49.59	47.29	42.97	42.27	41.3	40.7
ϵ_n-Greedy $c = 0.6$, $d = 0.1$	50.84	38.96	22.66	21.34	20.68	20.5
ϵ_n-Greedy $c = 2$, $d = 0.1$	49.97	49.87	40.51	38.62	37.34	36.98
ϵ-Greedy ϵ =0.1	31.48	35.94	35.31	34.17	30.99	26.8
GreedyEWA ϵ =0.1 $\alpha = 0.3$	9.92	13.7	35.68	34.28	34.7	36.64
SoftMaxEWA τ=0.5 α=0.3	29.18	47.42	48.04	49.94	49.2	48.42
Pursuit 0.01	49.24	47.77	41.43	41.15	41.0	41.0
AdaptivePursuit α =0.8,β =0.8, P_{min} =0.05	45.84	41.08	40.74	40.66	40.9	41.48
Poker	45.04	43.31	43.22	43.22	43.22	43.22
UCBNormal	0.0	100.0	100.0	0.0	100.0	100.0
AEMetaBandit $\delta = 0.005$, $\lambda = 10$, $MT = 100$	0.0	1.74	5.14	6.74	9.8	16.32

perform well at all, it seems to converge quickly because there is no noticeable increase in performance over time.

We assumed that the early convergence of the KBLA in the 10 player game and its rather bad performance in the 100 player game was due to too high update variance and therefore ran the tests again with smaller values. The results of these tests can be seen in table 5.13 and 5.14 for the 10 and 100 player game respectively.

Table 5.13: KBLA results for varying values of the update variance parameter in 10 player Goore game on peaked Gaussian function with amplitude 20 and where 2 yes votes are optimal.

Team / Iteration	**10**	**100**	**1000**	**2000**	**10000**	**100000**
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.0$	2.48	2.45	2.45	2.45	2.45	2.45
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 1e-05$	2.73	2.7	2.7	2.7	2.65	2.6
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.0001$	3.07	2.85	2.85	2.85	2.8	2.15
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.001$	2.85	2.85	2.75	2.55	2.21	2.02
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.01$	2.5	2.35	2.17	2.12	2.07	2.07
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.1$	2.77	2.58	2.14	2.22	2.13	2.17

Table 5.14: KBLA results for varying values of the update variance parameter in 100 player Goore game on peaked Gaussian function with amplitude 20 and where 20 yes votes are optimal.

Team / Iteration	**10**	**100**	**1000**	**2000**	**10000**	**100000**
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.0$	42.15	40.3	40.12	40.08	40.02	39.64
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 1e-05$	42.51	40.76	40.52	40.42	40.24	37.36
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.0001$	42.66	40.38	40.19	40.02	37.98	23.57
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.001$	42.83	40.32	39.0	37.2	26.79	24.7
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.01$	41.95	39.21	30.41	28.7	28.38	28.03
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.1$	42.05	35.61	33.93	34.59	34.32	34.58

Indeed, for the 100 player game it seems clear that too much constant exploration was the problem. We also see that lowering the exploration in the 10 player game increases performance, but the KBLA is definitely not as sensitive to the update variance parameter here.

One might ask, what if the feedback range of the Goore game was larger, would that affect the parameter choice? Common reasoning implies that a larger feedback range gives greater distance between rewards and in a way the environment therefore "moves" faster and should require a higher update parameter. Table 5.15 shows the results for 100 KBLA players on the same peaked function, but with the amplitude increased to 400.

Table 5.15: KBLA results for varying values of the update variance parameter in 100 player Goore game on peaked Gaussian function with amplitude 400 and where 20 yes votes are optimal.

Team / Iteration	**10**	**100**	**1000**	**2000**	**10000**	**100000**
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.0$	43.93	38.66	34.86	34.75	34.24	34.24
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 1e-05$	44.87	38.57	35.24	34.27	32.78	31.83
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.0001$	45.51	38.96	34.46	33.17	31.25	29.93
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.001$	45.23	38.4	32.99	31.63	28.8	26.8
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.01$	44.93	37.52	31.3	29.55	25.79	23.99
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.1$	45.22	36.7	30.36	28.39	25.52	24.68

Here we see that we cannot expect to gain the same performance for the same parameter if the feedback range is different.

So far we have only considered the update parameters effect on the performance. However, the sensor parameter is also unknown in the Goore game. While the observation noise of the feedback function is

known, more noise is introduced by the other players. If all players had voted similarly each time, then the feedback would be the only source of noise and the sensor parameter had been known, but this is not the case. While the update variance parameter contributes to constant exploration, the sensor parameter can contribute to initial exploration, and thus might be more beneficial in environments like this where it is important to converge in the end.

Table 5.16 contains the results of increasing the sensor variance, and therefore the initial exploration, on the peaked Gaussian function with amplitude 20.

Table 5.16: The effect of increasing the sensor variance parameter in a 100 player Goore game on peaked Gaussian function with amplitude 20 and where 20 yes votes is optimal.

Team / Iteration	**10**	**100**	**1000**	**2000**	**10000**	**100000**
KBLA $\sigma_z^2 = 10.0$, $\sigma_u^2 = 0.0$	43.21	34.18	32.89	32.42	31.75	30.71
KBLA $\sigma_z^2 = 50.0$, $\sigma_u^2 = 0.0$	46.69	32.3	28.18	27.89	27.28	26.49
KBLA $\sigma_z^2 = 100.0$, $\sigma_u^2 = 0.0$	47.68	36.96	25.21	24.89	24.13	23.66
KBLA $\sigma_z^2 = 225$, $\sigma_u^2 = 0.0$	48.3	42.58	20.91	21.12	21.16	20.94
KBLA $\sigma_z^2 = 400.0$, $\sigma_u^2 = 0.0$	48.48	44.71	17.58	17.14	18.1	19.08

Now, can these parameters be extended to the function with amplitude 400? Table 5.17 has the answer. Unfortunately, we see that there is no silver bullet, the sensor variance parameter had to be increased significantly to obtain the same performance.

For comparison, the results for the other schemes are also shown. We see also many of these are very sensitive to their parameter configuration. Again, it seems that a properly configured ϵ_n-Greedy scheme is very strong in this game both in terms of convergence speed and accuracy. As before, AdaptEvEMeta-Bandit needs some time, but performs very well over time. It also surprising to see the basic Greedy scheme with as much as 10% exploration obtaining a very good score in the end.

Another consideration when increasing the sensor variance parameter in the KBLA, is the learning speed. More exploration in the start avoids early convergence, but may slow down the automaton significantly. We end of this section[2] with figure 5.8 where we show the learning speed for several choices of the sensor variance parameter on the peaked Gaussian function with amplitude 20.

[2]The results for the flat Gaussian function where omitted because they revealed nothing new.

Table 5.17: Results for 100 player Goore game on peaked Gaussian function with amplitude 400 and where 20 yes votes are optimal.

Team / Iteration	**10**	**100**	**1000**	**2000**	**10000**	**100000**
KBLA $\sigma_z^2 = 225$, $\sigma_u^2 = 0.0$	44.19	38.88	36.46	36.11	35.73	35.65
KBLA $\sigma_z^2 = 400.0$, $\sigma_u^2 = 0.0$	44.65	39.05	37.21	36.89	36.57	36.51
KBLA $\sigma_z^2 = 2000.0$, $\sigma_u^2 = 0.0$	44.47	38.26	36.69	36.38	35.71	35.45
KBLA $\sigma_z^2 = 10000$, $\sigma_u^2 = 0.0$	46.48	32.39	31.82	30.83	29.76	28.89
KBLA $\sigma_z^2 = 100000.0$, $\sigma_u^2 = 0.0$	49.87	37.58	18.81	18.89	19.23	20.02
ϵ_n-Greedy $c = 0.3$, $d = 0.1$	50.4	35.92	29.23	29.0	28.64	28.61
ϵ_n-Greedy $c = 0.6$, $d = 0.1$	49.8	38.18	19.52	18.45	17.8	17.62
ϵ_n-Greedy $c = 1.0$, $d = 0.1$	47.4	50.28	17.82	15.7	14.35	14.04
Greedy ϵ =0.1	5.53	4.52	28.58	28.84	28.69	21.22
Greedy ϵ =0.05	2.8	2.48	32.35	32.43	32.35	30.6
GreedyEWA ϵ =0.1 α =0.3	9.33	10.56	27.83	30.06	31.1	31.77
GreedyEWA ϵ =0.05 α =0.9	6.2	37.52	39.74	41.13	39.55	39.9
SoftMaxEWA τ=0.1 α=0.3	0.0	0.08	1.46	4.0	17.91	42.86
SoftMaxEWA τ=0.1 α=0.9	0.0	0.28	2.16	4.62	17.56	42.16
Pursuit 0.005	0.73	1.36	1.99	2.0	2.0	2.0
AdaptivePursuit α =0.8,β =0.8, P_{min} =0.05	1.13	1.44	1.88	1.88	1.89	1.89
AdaptivePursuit α =0.2,β =0.2, P_{min} =0.01	1.0	0.78	0.8	1.6	1.97	1.98
AEMetaBandit $\delta = 0.005$, $\lambda = 5$, $MT = 100$	0.0	0.0	0.04	1.0	19.95	22.49

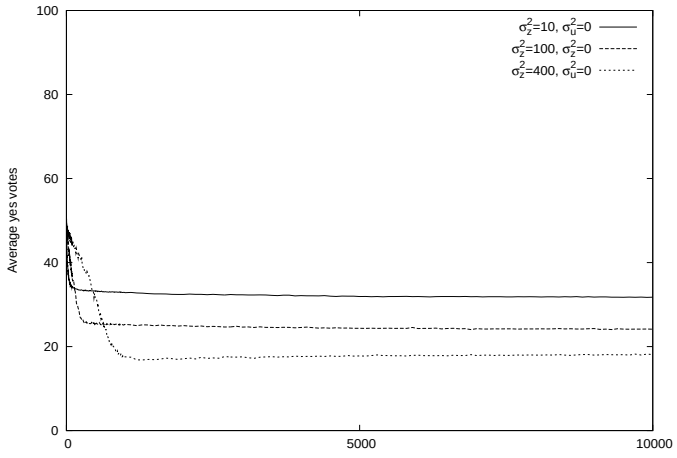


Figure 5.8: The impact on learning speed when the sensor variance parameter is increased.

5.5 Zero-sum games

As mentioned in chapter 4, zero-sum games serves as a benchmark for testing rationality in learning algorithms. These experiments are inspired by the analysis done in [NT89] and the game matrices have been chosen to cover different scenarios. In the game matrices, we show only the payoff to player one, since, because of the zero-sum property, the payoff to player two is easily determined by negating the payoff to player one.

The first game illustrates the case where both players have dominating strategies.

$$Game1 = \begin{bmatrix} 2 & 6 \\ -3 & 3 \end{bmatrix}$$

For the second matrix, only player one has dominating strategies. Game2 is also much harder than Game1 in the sense that the difference between feedback values are smaller.

$$Game2 = \begin{bmatrix} 2 & 1 \\ -2 & 1 \end{bmatrix}$$

The last type of zero sum games we have considered is the case of where no player has dominating strategies. Only mixed strategy equilibriums exist. The optimal solution to such zero-sum games is the minimax solution. Here we have chosen two games, one with 2 actions, the two-finger morra game, and one with 5 actions, an extended version of the rock, paper, scissors game.

$$Morra = \begin{bmatrix} 2 & -3 \\ -3 & 4 \end{bmatrix}$$

$$RPS_{EX} = \begin{bmatrix} 0 & -1 & 1 & -1 & 1 \\ 1 & 0 & -1 & -1 & 1 \\ -1 & 1 & 0 & -1 & 1 \\ 1 & 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 1 & 0 \end{bmatrix}$$

As we saw in section 4.3.3 the optimal mixed strategy for the Two-finger Morra game is $(a_1 : 7/12, a_2 : 5/12)$, while in the extended rock, paper, scissors the optimal mixed strategy is $(a_1 : 1/9, a_2 : 1/9, a_3 : 1/9, a_4 : 1/3, a_5 : 1/3)$. Because of symmetry, this holds for both players.

For the games with pure strategy equilibria we only put the schemes up against an instance of themselves to observe how fast they are able to identify the best choice. However, in the mixed strategy games we run a tournament too. The reason for this is to observe which scheme is best at exploiting weaknesses in other schemes.

5.5.1 Results Game1

Table 5.18 contains the results for the Game1. The first number in each column is player one's probability of selecting the dominating strategy, while the second number is player two's probability of selecting the dominating strategy.

Table 5.18: Probability of selecting the dominating strategy in Game1

Player / Iteration	**10**	**100**	**1000**	**10000**
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.0$	0.97, 0.99	1.00, 1.00	1.00, 1.00	1.00, 1.00
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.001$	0.97, 0.98	1.00, 1.00	1.00, 1.00	1.00, 1.00
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.01$	0.97, 0.98	1.00, 0.99	1.00, 1.00	1.00, 1.00
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.1$	0.96, 0.97	0.98, 0.98	0.98, 0.98	0.98, 0.98
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.5$	0.96, 0.92	0.95, 0.94	0.95, 0.94	0.96, 0.94
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 1.0$	0.93, 0.91	0.93, 0.92	0.93, 0.92	0.93, 0.92
ϵ_n-Greedy $c = 0.3$, $d = 0.1$	0.51, 0.51	0.84, 0.84	0.99, 0.98	1.00, 1.00
Greedy ϵ =0.02	0.99, 0.99	0.99, 0.99	0.99, 0.99	0.99, 0.99
GreedyEWA ϵ =0.02 α =0.3	0.99, 0.99	0.99, 0.99	0.99, 0.99	0.99, 0.99
SoftMaxEWA τ=0.1 α=0.3	1.00, 0.88	1.00, 1.00	1.00, 1.00	1.00, 1.00
Poker	0.82, 0.98	0.99, 0.99	0.99, 0.99	0.99, 0.99
Pursuit 0.01	0.52, 0.53	0.81, 0.80	1.00, 1.00	1.00, 1.00
AdaptivePursuit α =0.8,β =0.8, P_{min} =0.01	0.99, 0.99	0.99, 0.99	0.99, 0.99	0.99, 0.99
UCBNormal	0.67, 0.67	0.86, 0.90	0.99, 0.98	1.00, 0.99
AEMetaBandit $\delta = 0.005$, $\lambda = 5.0$, $MT = 100$	0.67, 0.67	0.86, 0.90	0.99, 0.98	1.00, 0.99

Perhaps not surprisingly, most schemes are able to quickly identify the dominating strategy. For this

game, it seems that an update parameter close to zero gives the best performance of the Kalman Bayesian Learning Automaton. In terms of convergence speed GreedyEWA, SoftMaxEWA, AdaptivePursuit and the KBLA players with small update variance is the fastest schemes to find a near-optimal solution.

5.5.2 Results Game2

Table 5.19 shows the results for Game2. Again, the reported probabilities are versus an instance of itself, and the first number is player one's probability of playing the optimal action, while the second number is player two's probability of playing the optimal action. Only player one has a dominating strategy (a_1), but player two's optimal choice is a_2 because it minimizes player one's payoff and therefore also its loss.

Table 5.19: Probability of playing the optimal action in Game2

Player / Iteration	**10**	**100**	**1000**	**10000**
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.0$	0.98, 0.70	1.00, 0.98	1.00, 1.00	1.00, 1.00
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.001$	0.98, 0.69	1.00, 0.96	1.00, 0.98	1.00, 0.98
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.01$	0.97, 0.69	0.99, 0.91	0.98, 0.92	0.98, 0.92
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.1$	0.95, 0.70	0.94, 0.80	0.94, 0.80	0.94, 0.80
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.5$	0.90, 0.69	0.89, 0.70	0.89, 0.71	0.89, 0.70
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 1.0$	0.88, 0.65	0.86, 0.66	0.86, 0.67	0.86, 0.67
ϵ_n-Greedy $c = 0.3$, $d = 0.1$	0.50, 0.51	0.84, 0.17	0.98, 0.02	1.00, 0.00
Greedy ϵ =0.02	0.99, 0.03	0.99, 0.01	0.99, 0.01	0.99, 0.01
GreedyEWA ϵ =0.02 α =0.3	0.99, 0.02	0.99, 0.01	0.99, 0.01	0.99, 0.01
SoftMaxEWA τ=0.1 α=0.3	1.00, 0.76	1.00, 0.97	1.00, 0.99	1.00, 1.00
Poker	0.98, 0.65	1.00, 0.93	1.00, 0.94	1.00, 0.94
Pursuit 0.01	0.54, 0.48	0.81, 0.30	1.00, 0.00	1.00, 0.00
AdaptivePursuit α =0.8,β =0.8, P_{min} =0.01	0.99, 0.01	0.99, 0.01	0.99, 0.01	0.99, 0.01
UCBNormal	0.67, 0.33	0.90, 0.77	0.98, 0.02	0.99, 0.00
AEMetaBandit $\delta = 0.005$, $\lambda = 5.0$, $MT = 100$	0.67, 0.33	0.90, 0.77	0.98, 0.02	0.99, 0.00

Again, we note the rapid convergence rate of the KBLA to its equilibrium point. Just after 100 iterations, almost all results are unchanged. However a larger update variance parameter does not seem to increase the speed of the convergence rate, and it definitely seems that assuming a stationary environment ($\sigma_u^2 = 0$) is the best here due to its superior accuracy.

SoftMax is again a top performer as in Game1, and is together with Poker the only of the other schemes that does not converge to the wrong solution.

5.5.3 Results two-finger Morra

The results for the two-finger Morra game when playing versus an instance of itself can be seen in table 5.20. The probability of a_1 is reported for player one, while the probability of a_2 is reported for player 2.

Table 5.20: Results for the two-finger Morra game where the optimal mixed strategy is (0.58, 0.42)

Player / Iteration	**10**	**100**	**1000**	**10000**
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.0$	0.91, 0.95	0.74, 0.73	0.71, 0.71	0.70, 0.69
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.001$	0.61, 0.31	0.62, 0.38	0.58, 0.41	0.57, 0.41
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.01$	0.60, 0.30	0.58, 0.37	0.57, 0.40	0.57, 0.40
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.1$	0.61, 0.29	0.57, 0.39	0.57, 0.40	0.57, 0.40
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.5$	0.62, 0.31	0.58, 0.40	0.57, 0.39	0.57, 0.39
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 1.0$	0.60, 0.37	0.56, 0.40	0.58, 0.39	0.58, 0.39
ϵ_n-Greedy $c = 0.3$, $d = 0.1$	0.49, 0.49	0.74, 0.65	0.63, 0.49	0.66, 0.52
Greedy ϵ =0.02	0.99, 0.99	0.95, 0.85	0.62, 0.58	0.62, 0.60
GreedyEWA ϵ =0.02 α =0.3	0.98, 0.96	0.98, 0.74	0.98, 0.47	0.98, 0.50
SoftMax-EWA-N τ=0.1 α=0.3	0.62, 0.44	0.86, 0.87	0.97, 0.98	0.93, 0.94
Poker	0.54, 0.75	0.75, 0.76	0.75, 0.77	0.75, 0.76
Pursuit 0.01	0.52, 0.53	0.56, 0.67	0.95, 0.67	1.00, 0.67
AdaptivePursuit α =0.8,β =0.8, P_{min} =0.01	0.99, 0.69	0.98, 0.53	0.98, 0.51	0.98, 0.50
UCBNormal	0.67, 0.67	0.46, 0.64	0.57, 0.64	0.59, 0.62
AEMetaBandit $\delta = 0.005$, $\lambda = 5.0$, $MT = 100$	0.67, 0.67	0.45, 0.62	0.57, 0.64	0.59, 0.62

We see that none of the schemes are able to identify the optimal mixed strategy. Some of the KBLA schemes and UCBNormal are very close. This time the KBLA with σ_u^2=0.0 falls behind.

However, being able to play a good mixed strategy against an instance of itself does not mean that it can do so versus a variety of opponents. Table 5.21 contains the scores for the tournament where every player played against all others as both player one and player two.

The tournament are dominated by the adaptive schemes which are definitely better than the converging

Table 5.21: Average total score in the two-finger Morra tournament

Player / Iteration	**10**	**100**	**1000**	**2000**	**10000**
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.5$	7.96	164.62	2103.63	4573.32	26687.17
ϵ_n-Greedy $c = 0.3$, $d = 0.1$	-0.76	25.23	-1173.61	-2765.12	-17747.85
Greedy ϵ =0.05	5.4	-101.18	-1034.34	-2026.99	-9672.5
Greedy ϵ =0.1	5.67	-38.3	-506.16	-957.17	-3959.81
GreedyEWA ϵ =0.1 α =0.3	6.54	-14.45	375.53	1053.48	8892.19
SoftMaxEWA τ=0.5 α=0.3	9.05	122.41	1539.75	3374.92	19835.35
Pursuit 0.01	-0.09	28.69	-1633.8	-4924.86	-36485.31
AdaptivePursuit α =0.8,β =0.8, P_{min} =0.01	-9.67	-230.34	-1856.02	-3157.77	-10267.81
Poker	8.02	-105.8	-2033.55	-4316.47	-23575.75
UCBNormal	-16.11	74.16	2110.19	4572.18	23160.89
AEMetaBandit $\delta = 0.005$, $\lambda = 10$, $MT = 100$	-16.02	74.96	2108.37	4574.49	23133.44

schemes at playing mixed strategies versus a wide range of contenders. The individual arm probabilities and the expected reward for the KBLA versus the different opponents are shown in table 5.22. The reported probabilities are all for action one at time step 10000. In the leftmost column with probabilities the KBLA is player one while in the rightmost column with probabilities the KBLA is player two. Again, the optimal probability for action one is 0.58 for both players, but player two has a slight advantage in this game in terms of expected reward (ER). The ER is the average score of the game if it were repeated infinitely.

Table 5.22: Tournament results for KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.5$ against each individual player

Adversary	$P1_{a_1}$, $P2_{a_1}$	**ER(P1)**	$P1_{a_1}$, $P2_{a_1}$	**ER(P2)**
ϵ_n-Greedy $c = 0.3$, $d = 0.1$	0.95, 0.99	1.71	0.99, 0.05	2.69
Greedy ϵ =0.1	0.82, 0.84	0.65	0.84, 0.24	1.14
GreedyEWA ϵ =0.1 α =0.3	0.82, 0.85	0.67	0.86, 0.24	1.22
SoftMaxEWA τ=0.5 α=0.3	0.6, 0.63	-0.07	0.58, 0.61	0.08
Pursuit	0.96, 1.0	1.8	1.0, 0.04	2.80
AdaptivePursuit α =0.8,β =0.8, P_{min} =0.01	0.92, 0.96	1.48	0.97, 0.09	2.37
Poker	0.94, 0.98	1.64	0.56, 0.47	0.05
UCBNormal	0.6, 0.63	-0.07	0.57, 0.56	0.08
AEMetaBandit $\delta = 0.005$, $\lambda = 10$, $MT = 100$	0.6, 0.63	-0.07	0.57, 0.56	0.08

We see that even though the KBLA does not achieve a positive score when playing as player one against some schemes, for instance UCBNormal, it is able to get a slightly higher total expected reward against every scheme. Thus, it makes sense that it was able to achieve the top score in the tournament.

The KBLA was also tested against a perfect opponent. Table 5.23 contains the results of this experiment. Here KBLA has only played as player one, which has a maximum expected payoff off $-1/12 = -0.0833$. In contrast to the tournament score, where the total score is shown, the average score is reported here. We see that the performance gets slightly worse over time, but the KBLA are still very close to the maximum attainable payoff.

Table 5.23: The KBLA versus a perfect opponent in two-finger Morra

Player / Iteration	**10**	**100**	**1000**	**2000**	**10000**
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.5$	-0.08481	-0.08576	-0.08549	-0.09074	-0.09341
FixedProbPlayer [0.583, 0.416]	0.08481	0.08576	0.08549	0.09074	0.09341
KBLA $P(a_1)$, FPP $P(a_2)$	0.58, 0.58	0.59, 0.58	0.59, 0.58	0.59, 0.58	0.59, 0.58

5.5.4 Results Rock, paper, scissors extended

In table 5.24 we see the effect on performance in zero-sum games when increasing the number of actions. Only the results for $t = 10000$ is shown.

While sensitive to the update variance parameter, the KBLA is able to get very close to the optimal solution with a small update variance parameter. Most of the non-KBLA schemes are far away from the optimal strategy, except UCBNormal which in fact is quite close. SoftMaxEWA also performs adequate and could likely get closer with further tuning.

Table 5.24: Arm probabilities when playing versus an instance of itself in Rock, paper, scissors extended

Player / Iteration	**10000**
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.0$	0.11, 0.11, 0.12, 0.34, 0.32
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 1e-06$	0.11, 0.11, 0.11, 0.35, 0.31
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 1e-05$	0.11, 0.11, 0.12, 0.36, 0.30
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.0001$	0.12, 0.12, 0.11, 0.37, 0.28
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.001$	0.13, 0.11, 0.13, 0.39, 0.24
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.01$	0.14, 0.14, 0.14, 0.37, 0.20
Poker	0.10, 0.12, 0.12, 0.52, 0.14
ϵ_n-Greedy $c = 0.3$, $d = 0.1$	0.27, 0.12, 0.01, 0.46, 0.13
ϵ_n-Greedy $c = 0.6$, $d = 0.1$	0.24, 0.11, 0.01, 0.45, 0.19
Greedy ϵ =0.1	0.10, 0.11, 0.07, 0.45, 0.27
Greedy ϵ =0.02	0.15, 0.13, 0.05, 0.45, 0.23
GreedyEWA ϵ =0.1 α =0.3	0.55, 0.14, 0.02, 0.26, 0.03
GreedyEWA ϵ =0.02 α =0.3	0.67, 0.13, 0.00, 0.19, 0.01
SoftMaxEWA τ=0.5 α=0.3	0.17, 0.16, 0.17, 0.33, 0.16
SoftMaxEWA τ=0.1 α=0.3	0.13, 0.11, 0.12, 0.47, 0.17
Pursuit 0.01	0.56, 0.04, 0.00, 0.40, 0.01
AdaptivePursuit α =0.8,β =0.8, P_{min} =0.01	0.82, 0.08, 0.01, 0.08, 0.01
AdaptivePursuit α =0.8,β =0.8, P_{min} =0.1	0.33, 0.17, 0.14, 0.23, 0.13
UCBNormal	0.11, 0.12, 0.11, 0.38, 0.29
AEMetaBandit $\delta = 0.005$, $\lambda = 80$, $MT = 50$	0.11, 0.12, 0.11, 0.38, 0.29

5.6 Prisoner's Dilemma

The Prisoner's Dilemma is motivated by its special properties and the fact that it is not zero-sum. We use a normalized version where the payoffs can be seen in the following matrix. For instance, if both players adopts a rational behavior and testifies, then they both receive 0.4 as payoff.

$$PD = \begin{bmatrix} 0.8, 0.8 & 0.0, 1.0 \\ 1.0, 0.0 & 0.4, 0.4 \end{bmatrix}$$

5.6.1 Results

Table 5.25 contains the score obtained when playing versus an instance of itself in the Prisoners Dilemma.

Table 5.25: Scores vs self in the iterative Prisoner's Dilemma

Player / Iteration	**10**	**100**	**1000**	**2000**	**10000**
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.0$	0.60	0.47	0.40	0.40	0.40
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.001$	0.53	0.45	0.46	0.42	0.42
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.01$	0.54	0.45	0.48	0.46	0.48
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.1$	0.56	0.49	0.50	0.51	0.50
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.5$	0.52	0.52	0.51	0.52	0.51
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 1.0$	0.55	0.51	0.53	0.53	0.53
Poker	0.54	0.48	0.44	0.44	0.43
ϵ_n-Greedy $c = 0.6$, $d = 0.1$	0.56	0.48	0.40	0.40	0.40
Greedy ϵ =0.05	0.60	0.51	0.43	0.43	0.40
GreedyEWA ϵ =0.1 α =0.3	0.51	0.55	0.58	0.58	0.54
GreedyEWA ϵ =0.05 α =0.9	0.65	0.76	0.72	0.71	0.72
SoftMaxEWA τ=0.1 α=0.3	0.60	0.47	0.43	0.42	0.43
SoftMaxEWA τ=0.3 α=0.3	0.56	0.45	0.46	0.46	0.48
Pursuit 0.01	0.51	0.47	0.40	0.40	0.40
AdaptivePursuit α =0.8,β =0.8, P_{min} =0.01	0.73	0.72	0.75	0.76	0.72
AdaptivePursuit α =0.8,β =0.8, P_{min} =0.1	0.63	0.62	0.59	0.61	0.60
UCBNormal	0.67	0.51	0.49	0.59	0.71

We see that the KBLA players seems to converge quickly to the rational behavior. It is clear that a higher

update variance parameter causes the automaton to deviate somewhat from a completely rational behavior, but with the constant exploration applied this is expected.

It seems that cooperating is more common for the dynamic schemes, especially one configuration of GreedyEWA and AdaptivePursuit have nearly determined this non-rational, but optimal solution. UCB-Normal is the only of the stationary schemes that also approaches this behavior.

The most rational KBLA player ($\sigma_u^2 = 0$) has been applied in a tournament setting with the rest of the schemes. The result of this experiment can be seen in table 5.26.

Table 5.26: Prisoner's Dilemma tournament scores

Player / Iteration	**10**	**100**	**1000**	**2000**	**10000**
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.0$	12.57	122.45	1170.77	2305.67	11248.49
ϵ_n-Greedy $c = 0.3$, $d = 0.1$	11.78	113.41	1162.67	2293.29	11195.43
Greedy ϵ =0.02	11.79	108.69	1038.89	2129.19	10993.24
GreedyEWA ϵ =0.02 α =0.3	12.06	87.65	495.65	845.11	3175.92
SoftMaxEWA τ=0.1 α=0.3	12.89	125.44	1130.78	2206.11	10712.26
Pursuit 0.01	11.66	110.12	1161.17	2293.38	11179.47
AdaptivePursuit α =0.8,β =0.8, P_{min} =0.01	10.19	67.37	419.88	710.93	2510.85
Poker	12.18	119.28	1114.21	2192.53	10708.57
UCBNormal	8.5	111.59	1029.64	2076.38	10776.73

While the KBLA obtains the top score, the scores are so similar that it seems like most of the schemes plays rationally in this game. In fact, only the GreedyEWA and AdaptivePursuit schemes seems to deviate from the rational strategy in this tournament, because they are so far behind and probably tries to cooperate way to often, which we know is not beneficial against a rational opponent. Also, it is rather surprising to see that UCBNormal has a high score, since it did not play rationally versus an instance of itself, but here we see that it is clearly able to play rationally versus other opponents.

To see exactly how rational the KBLA plays, table 5.27 shows the testify probabilities at time step 10000 against each individual opponent. The results for UCBNormal are also included for comparison.

While there are small differences in testify probabilities, we see that the KBLA is even more eager to testify than the UCBNormal scheme and should consequently earn some extra points here and there.

Table 5.27: Probability of testifying (T) for the KBLA and UCBNormal versus each individual opponent in the Prisoner's Dilemma tournament.

Adversary	**T(KBLA), T(Adv)**	**T(UCBNormal),T(Adv)**
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.0$	x	0.98, 1.00
ϵ_n-Greedy $c = 0.3$, $d = 0.1$	1.00, 1.00	0.98, 1.00
Greedy ϵ =0.02	1.00, 0.99	0.96, 0.98
GreedyEWA ϵ =0.02 α =0.3	1.00, 0.05	0.95, 0.06
SoftMaxEWA τ=0.1 α=0.3	1.00, 0.93	0.90, 0.87
Pursuit 0.01	1.00, 1.00	0.98, 1.0
AdaptivePursuit α =0.8,β =0.8, P_{min} =0.01	1.00, 0.04	0.94, 0.04
Poker	1.00, 0.90	0.95, 0.96
UCBNormal	1.00, 0.98	x

5.7 Sensitivity

The tested parameters are in factors of standard deviations from the true parameter. For instance, a factor range $\{1, 2, 3, 4, 5\}$ for a true standard deviation value of 50 gives the following test range parameters: $\{10, 12.5, 25, 50, 75, 100, 150, 200, 250\}$.

5.7.1 Sensitivity to incorrectly specified update variance parameter

The regret at $t = 2000$ for a two-armed random walk environment with true update variance $\sigma_u = 50$ is shown in table 5.28, while the corresponding arm probabilities resides in table 5.29.

Table 5.28: Regret at $t = 2000$ for incorrectly specified update parameter in two-armed random walk environment with true σ_u =50.

Belief	$\frac{1}{5}\sigma_u$	$\frac{1}{4}\sigma_u$	$\frac{1}{3}\sigma_u$	$\frac{1}{2}\sigma_u$	$\boldsymbol{\sigma_u}$	$2\sigma_u$	$3\sigma_u$	$4\sigma_u$	$5\sigma_u$
Regret	200.95	174.75	149.83	110.25	81.81	111.48	146.56	178.28	206.93

The results are as expected, specifying a too high or too low update parameter decreases performance and the performance loss only increases with values further away from the true value.

However, it is quite interesting to observe that specifying a too high update parameter gives a higher probability of selecting the optimal arm than a too low update parameter, but in terms of regret it definitely

Table 5.29: Probability of selecting the optimal arm for incorrectly specified update variance parameter in two-armed random walk environment with true σ_u =50.

Player / Iteration	**10**	**100**	**1000**	**2000**
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 100.0$	0.80	0.87	0.89	0.91
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 156.25$	0.80	0.88	0.91	0.92
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 277.777777778$	0.80	0.90	0.93	0.93
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 625.0$	0.80	0.90	0.95	0.96
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 2500.0$	0.78	0.92	0.97	0.98
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 10000.0$	0.72	0.89	0.96	0.97
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 22500.0$	0.70	0.86	0.94	0.95
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 40000.0$	0.67	0.82	0.93	0.94
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 62500.0$	0.63	0.81	0.92	0.93

Table 5.30: Regret at $t = 2000$ for incorrectly specified update parameter in fifty-armed random walk environment with true σ_u =50.

Belief	$\frac{1}{5}\sigma_u$	$\frac{1}{4}\sigma_u$	$\frac{1}{3}\sigma_u$	$\frac{1}{2}\sigma_u$	$\boldsymbol{\sigma_u}$	$2\sigma_u$	$3\sigma_u$	$4\sigma_u$	$5\sigma_u$
Regret	343.86	300.36	246.62	147.77	110.9	261.43	401.33	516.41	606.83

Table 5.31: Probability of selecting the optimal arm for incorrectly specified update parameter in fifty-armed random walk environment with true σ_u =50.

Player / Iteration	**10**	**100**	**1000**	**2000**
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 100.0$	0.06	0.16	0.15	0.17
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 156.25$	0.07	0.20	0.18	0.21
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 277.777777778$	0.05	0.22	0.24	0.27
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 625.0$	0.06	0.31	0.46	0.51
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 2500.0$	0.03	0.43	0.78	0.84
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 10000.0$	0.02	0.14	0.64	0.74
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 22500.0$	0.02	0.06	0.46	0.61
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 40000.0$	0.02	0.04	0.32	0.48
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 62500.0$	0.02	0.03	0.22	0.37

seems better to specify a lower parameter. This is even more evident if we increase the problem size to fifty arms, for which the regret and optimal arm probabilities can be seen in table 5.30 and 5.31 respectively.

Clearly, incorrectly specifying the parameters drops both the regret and optimal arm probability a lot more in fifty-armed environments than it does in two-armed environments.

5.7.2 Sensitivity to incorrectly specified sensor variance parameter

The results in terms of regret and optimal arm probabilities for incorrectly specifying the sensor variance parameter in a two-armed environment with true $\sigma_z = 50$ are shown in table 5.32 and 5.33.

Table 5.32: Regret at $t = 2000$ for incorrectly specified sensor variance parameter in two-armed random walk environment with true σ_z =50.

Belief	$\frac{1}{5}\sigma_z$	$\frac{1}{4}\sigma_z$	$\frac{1}{3}\sigma_z$	$\frac{1}{2}\sigma_z$	$\boldsymbol{\sigma_z}$	$2\sigma_z$	$3\sigma_z$	$4\sigma_z$	$5\sigma_z$
Regret	372.92	357.22	343.41	319.46	333.15	416.31	479.76	535.48	575.73

Table 5.33: Probability of selecting the optimal arm for incorrectly specified sensor variance parameter in two-armed random walk environment with true σ_z =50.

Player / Iteration	**10**	**100**	**1000**	**2000**
KBLA $\sigma_z^2 = 100.0$, $\sigma_u^2 = 1.0$	0.50	0.60	0.84	0.90
KBLA $\sigma_z^2 = 156.25$, $\sigma_u^2 = 1.0$	0.52	0.60	0.86	0.90
KBLA $\sigma_z^2 = 277.777777778$, $\sigma_u^2 = 1.0$	0.53	0.60	0.86	0.91
KBLA $\sigma_z^2 = 625.0$, $\sigma_u^2 = 1.0$	0.52	0.62	0.88	0.92
KBLA $\sigma_z^2 = 2500.0$, $\sigma_u^2 = 1.0$	0.50	0.63	0.87	0.91
KBLA $\sigma_z^2 = 10000.0$, $\sigma_u^2 = 1.0$	0.50	0.59	0.82	0.88
KBLA $\sigma_z^2 = 22500.0$, $\sigma_u^2 = 1.0$	0.48	0.57	0.79	0.85
KBLA $\sigma_z^2 = 40000.0$, $\sigma_u^2 = 1.0$	0.52	0.55	0.76	0.83
KBLA $\sigma_z^2 = 62500.0$, $\sigma_u^2 = 1.0$	0.50	0.54	0.73	0.81

Surprisingly, the true sensor parameter does not seem to be the best one! We actually see that values slightly below the true value gives the best results. Also, it is clear from the results that it is better to choose a small sensor value rather than a large value if unsure about the true parameter.

When increasing the number of arms to fifty we again see the pattern with lower regret in table 5.34, but also lower optimal arm probability when the sensor decreases as seen in table 5.35.

Table 5.34: Regret at $t = 2000$ for incorrectly specified sensor parameter in fifty-armed random walk environment with true σ_z =50.

Belief	$\frac{1}{5}\sigma_z$	$\frac{1}{4}\sigma_z$	$\frac{1}{3}\sigma_z$	$\frac{1}{2}\sigma_z$	$\boldsymbol{\sigma_z}$	$2\sigma_z$	$3\sigma_z$	$4\sigma_z$	$5\sigma_z$
Regret	305.76	299.15	300.41	316.87	400.62	572.61	695.1	778.28	830.99

Table 5.35: Probability of selecting the optimal arm for incorrectly specified sensor parameter in fifty-armed random walk environment with σ_z =50.

Player / Iteration	**10**	**100**	**1000**	**2000**
KBLA $\sigma_z^2 = 100.0$, $\sigma_u^2 = 1.0$	0.02	0.04	0.37	0.51
KBLA $\sigma_z^2 = 156.25$, $\sigma_u^2 = 1.0$	0.02	0.04	0.42	0.54
KBLA $\sigma_z^2 = 277.777777778$, $\sigma_u^2 = 1.0$	0.02	0.04	0.44	0.55
KBLA $\sigma_z^2 = 625.0$, $\sigma_u^2 = 1.0$	0.02	0.03	0.46	0.56
KBLA $\sigma_z^2 = 2500.0$, $\sigma_u^2 = 1.0$	0.02	0.03	0.40	0.56
KBLA $\sigma_z^2 = 10000.0$, $\sigma_u^2 = 1.0$	0.02	0.02	0.23	0.45
KBLA $\sigma_z^2 = 22500.0$, $\sigma_u^2 = 1.0$	0.03	0.02	0.11	0.33
KBLA $\sigma_z^2 = 40000.0$, $\sigma_u^2 = 1.0$	0.02	0.02	0.07	0.23
KBLA $\sigma_z^2 = 62500.0$, $\sigma_u^2 = 1.0$	0.02	0.02	0.05	0.16

5.7.3 Sensitivity to incorrectly specified sensor variance parameter and update variance parameter

These experiments involves random walk environments where both the sensor variance parameter and update variance parameter are incorrectly specified by the KBLA. From the experiments on random walk environments, section 5.3.2, we saw that identical environment parameters constitutes to easy random walk environments, while a high sensor variance parameter compared to a high update variance parameter proved to create difficult random walk environments. Table 5.36 shows the results for incorrectly specifying both parameters wrong in easy random walk environments ($\sigma_z = 50$,$\sigma_u = 50$), while table 5.37 contains the results for the hard random walk environments ($\sigma_z = 50$, $\sigma_u = 1$). Only the results for the fifty-armed configurations are shown, the results for the two-armed configurations are in the appendix.

For easy random walk environments, we see that in terms of regret the row with the true update parameter gives the best results. Also, a small update variance parameter drops the performance less than large values. An incorrect sensor variance parameter on the other hand, does not seem to have a great impact

Table 5.36: Regret at $t = 2000$ for incorrectly specified sensor variance parameter and update variance parameter in fifty-armed random walk environment with true $\sigma_z^2 = 50$ and true $\sigma_u^2 = 50$.

Belief	$\frac{1}{5}\sigma_z$	$\frac{1}{4}\sigma_z$	$\frac{1}{3}\sigma_z$	$\frac{1}{2}\sigma_z$	σ_z	$2\sigma_z$	$3\sigma_z$	$4\sigma_z$	$5\sigma_z$
$\frac{1}{5}\sigma_u$	334.67	342.81	341.79	350.3	339.18	346.05	343.59	357.1	354.2
$\frac{1}{4}\sigma_u$	297.88	304.81	308.56	302.16	307.36	309.62	312.34	310.14	315.34
$\frac{1}{3}\sigma_u$	246.31	239.89	246.98	252.39	246.97	248.89	247.84	240.62	251.92
$\frac{1}{2}\sigma_u$	150.05	149.03	147.47	145.66	147.63	151.18	151.24	151.85	157.13
σ_u	111.84	111.89	112.22	112.23	111.85	114.94	118.39	122.69	127.93
$2\sigma_u$	262.28	262.23	262.13	262.41	262.39	263.99	266.22	269.7	273.67
$3\sigma_u$	401.59	401.61	401.63	401.36	401.58	402.17	403.02	405.43	407.62
$4\sigma_u$	515.95	515.43	515.53	515.25	515.55	515.55	516.76	517.32	518.39
$5\sigma_u$	605.73	605.73	605.48	605.93	606.03	606.01	606.08	606.8	607.23

Table 5.37: Regret at $t = 2000$ for incorrectly specified sensor variance parameter and update variance parameter in fifty-armed random walk environment with true $\sigma_z^2 = 50$ and true $\sigma_u^2 = 1$.

Belief	$\frac{1}{5}\sigma_z$	$\frac{1}{4}\sigma_z$	$\frac{1}{3}\sigma_z$	$\frac{1}{2}\sigma_z$	σ_z	$2\sigma_z$	$3\sigma_z$	$4\sigma_z$	$5\sigma_z$
$\frac{1}{5}\sigma_u$	384.93	379.44	366.35	346.68	357.92	495.48	638.3	742.63	810.19
$\frac{1}{4}\sigma_u$	360.15	366.84	349.72	336.61	354.32	496.39	641.02	744.14	810.59
$\frac{1}{3}\sigma_u$	344.11	349.12	327.73	321.57	346.52	501.9	643.59	745.45	810.57
$\frac{1}{2}\sigma_u$	324.23	307.04	301.0	306.57	349.35	512.35	651.39	749.49	813.84
σ_u	298.93	301.18	302.17	315.44	398.84	566.13	689.74	772.27	825.02
$2\sigma_u$	402.26	400.26	405.51	428.36	525.78	674.12	757.5	811.98	848.51
$3\sigma_u$	513.97	512.43	515.26	531.64	613.4	729.81	796.62	837.21	865.37
$4\sigma_u$	600.85	596.94	596.8	608.53	673.3	767.78	820.19	853.28	877.73
$5\sigma_u$	667.94	664.05	660.98	670.65	717.84	792.99	836.86	865.67	885.93

on performance.

In hard random walk environments however, the sensor variance parameter has a major influence on performance, and again the pattern of increased performance with lower values arises. In fact, this seems true for every combination of parameters. The update variance parameter is also influential here, but again you lose less in terms of regret with small values.

5.8 Robustness to non-Gaussian feedback

This last part of the chapter contains the results for the experiments where the Kalman Bayesian Learning Automaton has not received the feedback it assumes, that is, normally distributed feedback.

5.8.1 Binary experiments

Binary zero-sum

For binary zero-sum we again used a mixed strategy game. The game matrix used is the same as in the analysis done by [NT89].

$$Binary = \begin{bmatrix} 0.8 & 0.2 \\ -0.4 & 0.6 \end{bmatrix}$$

The optimal strategies here is (0.25, 0.75) for player one and (0.5, 0.5) for player two.

We see that only very small update parameters gives good performance for the KBLA here. Still no parameter choices seems to beat the binary BLA. Among the non-Bayesian schemes, some schemes are very close to the optimal solution after 10 000 time steps, especially SoftMaxEWA(but seems to have stopped), ϵ_n-greedy, UCBTuned and the meta UCB scheme. It has been proven by [NT89] that $L_{R-\epsilon P}$ $\alpha = 0.002, \beta = 1e-05$ eventually converges to the optimal solution in this game. However, we see that only its second player has been able to do so within this time frame.

Binary goore game

The parameters of the KBLA are motivated by the earlier results with high sensor variance parameter on the Goore game and the very low update variance parameter that seems to be necessary in binary environments.

Table 5.39 and 5.40 contains the results for the 10 and 100 player binary Goore game.

While no scheme outperforms the Tsetlin automaton in the 10 player Goore Game, some are very close, both in terms of convergence speed and accuracy. The KBLA does not seem to require much explo-

Table 5.38: Results for binary zero-sum game where the optimal action one probabilities are 0.25 for player one and 0.5 for player two.

Player / Iteration	**10**	**100**	**1000**	**10000**
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.0$	0.47, 0.35	0.20, 0.37	0.25, 0.47	0.23, 0.49
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 1e-06$	0.46, 0.36	0.22, 0.37	0.27, 0.47	0.26, 0.47
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 1e-05$	0.45, 0.35	0.22, 0.37	0.26, 0.46	0.25, 0.46
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.0001$	0.46, 0.37	0.22, 0.38	0.26, 0.42	0.28, 0.43
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.001$	0.47, 0.37	0.26, 0.38	0.31, 0.40	0.31, 0.39
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.01$	0.47, 0.38	0.36, 0.39	0.37, 0.39	0.37, 0.37
BLA-Bernoulli	0.47, 0.36	0.19, 0.38	0.27, 0.50	0.25, 0.49
Poker	0.47, 0.35	0.20, 0.34	0.12, 0.42	0.09, 0.45
ϵ_n-greedy c =0.3 d =0.1	0.50, 0.50	0.27, 0.22	0.18, 0.59	0.22, 0.48
GreedyEWA ϵ =0.1 α =0.3	0.82, 0.67	0.91, 0.62	0.92, 0.64	0.92, 0.65
SoftMaxEWA τ=0.1 α=0.3	0.41, 0.31	0.29, 0.42	0.25, 0.45	0.25, 0.45
Pursuit 0.01	0.52, 0.50	0.46, 0.35	0.06, 0.42	0.04, 0.64
AdaptivePursuit α =0.8,β =0.8, P_{min} =0.01	0.98, 0.85	0.98, 0.94	0.98, 0.94	0.98, 0.94
$L_{R-\epsilon P}$ $\alpha = 0.002, \beta = 1e-05$	0.50, 0.50	0.49, 0.49	0.48, 0.41	0.10, 0.50
L_{R-I} a =0.005	0.49, 0.50	0.50, 0.48	0.40, 0.30	0.31, 0.34
Tsetlin	0.39, 0.40	0.40, 0.39	0.40, 0.40	0.40, 0.40
UCBTuned	0.43, 0.27	0.22, 0.36	0.23, 0.44	0.22, 0.45
AEMetaBandit $\delta = 0.005$, $\lambda = 5.0$, $MT = 50$	0.43, 0.26	0.21, 0.36	0.22, 0.43	0.24, 0.47

ration here and quickly identifies a near-optimal solution, but lacks slightly in accuracy compared to BLA Bernoulli and Tsetlin.

In the 100 player game, we see that the KBLA actually performs quite good compared to many other binary schemes. UCBTuned is definitely fastest to a good solution, but seems to lack slightly in determining the optimal point. L_{R-I} on the other hand needs some time, but performs very well over time and seems to converge towards the optimal solution.

Table 5.39: Results for 10 player Goore game on binary function where 8 yes votes is optimal.

Team / Iteration	**10**	**100**	**1000**	**2000**	**10000**	**100000**
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.0$	5.65	7.6	7.92	7.97	7.99	8.0
KBLA $\sigma_z^2 = 2.5$, $\sigma_u^2 = 0.0$	5.49	6.81	7.89	7.96	8.05	8.0
KBLA $\sigma_z^2 = 5.0$, $\sigma_u^2 = 0.0$	5.59	6.34	7.79	7.87	8.05	8.0
KBLA $\sigma_z^2 = 10.0$, $\sigma_u^2 = 0.0$	5.17	5.98	7.55	7.77	7.95	8.11
BLA-Bernoulli	6.29	7.95	8.0	8.0	8.0	8.0
ϵ_n-greedy $c = 0.6$,$d = 0.1$	5.07	6.58	8.72	8.8	8.87	8.1
ϵ_n-greedy $c = 0.1$,$d = 0.1$	5.05	7.65	7.99	7.94	7.94	7.88
SoftMaxEWA τ=0.1 α=0.3	7.25	7.75	7.96	7.96	7.9	7.88
GreedyEWA ϵ =0.02 α =0.3	5.13	6.14	7.66	7.67	7.72	7.73
Pursuit 0.01	5.05	6.29	8.97	8.94	8.94	8.94
AdaptivePursuit α =0.8,β =0.8, P_{min} =0.01	2.72	3.65	7.51	7.72	7.66	7.6
Poker	6.12	7.45	7.68	7.64	7.66	7.66
$L_{R-\epsilon P}$ $\alpha = 0.1, \beta = 0.1$	5.77	6.04	5.84	5.78	5.91	5.95
L_{R-I} a =0.005	5.09	5.11	6.75	7.74	8.11	8.0
Tsetlin N=1	6.96	8.0	8.0	8.0	8.0	8.0
UCBTuned	7.65	7.75	7.94	7.96	7.98	8.0
AEMetaBandit $\delta = 0.005$, $\lambda = 5.0$, $MT = 50$	7.58	7.79	7.93	7.98	7.99	8.0

The EvE challenge

The EvE challenge [AHC$^+$06] was hosted in 2006 as a contest for handling the exploration vs exploitation dilemma. The challenge includes six environments which intend to represent user behavior in probabilities in terms of link clicking on the Internet. Every environment has five arms, which represents the link probabilities.

The environments are very hard in the sense that the maximum success probability is just over 18 percent and the arm probabilities are very close. The environments (referred to as "visitors" in the challenge) are as follows [AHC$^+$06][3]:

- VisitorConstant - Stationary environment

[3]There were actually one more visitor in the original challenge, called VisitorWeeklyCloseVariation, but we were not able to

Table 5.40: Results for 100 player Goore game on binary function where 80 yes votes is optimal.

Team / Iteration	**10**	**100**	**1000**	**2000**	**10000**	**100000**
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.0$	56.49	55.07	69.38	71.58	72.0	72.3
KBLA $\sigma_z^2 = 5.0$, $\sigma_u^2 = 0.0$	52.4	52.09	56.35	61.64	82.74	82.67
KBLA $\sigma_z^2 = 10.0$, $\sigma_u^2 = 0.0$	52.31	51.7	54.01	56.41	88.37	87.19
BLA-Bernoulli	52.63	55.99	66.4	67.26	67.39	67.89
Poker	50.78	53.52	59.92	62.11	63.37	63.65
ϵ_n-greedy c:2,d:0.1	49.58	50.08	63.19	64.97	66.25	66.61
ϵ_n-greedy c:0.6,d:0.1	49.83	51.91	61.67	62.15	62.61	62.83
GreedyEWA ϵ =0.05 α =0.3	52.13	52.9	55.7	54.65	54.04	54.44
GreedyEWA ϵ =0.1 α =0.3	50.47	52.98	53.54	53.16	52.94	53.65
SoftMax-EWA τ=0.1 α=0.3	66.04	64.39	56.58	55.35	54.19	54.34
SoftMax-EWA τ=0.5 α=0.3	54.2	50.93	50.59	50.65	50.62	50.86
L_{R-I} a =0.005	50.33	50.13	52.06	54.44	74.55	79.84
L_{R-I} a =0.01	49.7	50.14	54.44	60.05	78.89	79.74
Tsetlin N=5	52.14	55.71	58.12	59.24	61.01	63.54
UCBTuned	78.67	74.0	75.99	75.91	77.68	77.6
AEMetaBandit $\delta = 0.005$, $\lambda = 5.0$, $MT = 50$	79.49	74.05	75.37	76.62	77.9	77.82

- VisitorFrequentSwap - Switching environment where the best option changes frequently.
- VisitorDailyVariation - Two sinusoidal components determines the probability, the component with the shortest period is dominant.
- VisitorWeeklyVariation - Same as the daily variation visitor except that the longest period is dominant, and thus the best option changes less frequently than the daily variation.
- VisitorLongGaussian - The best option changes after a long period of time.

Motivated by the earlier results showing good performance in binary experiments with update variances close to zero, we tested the KBLA in these environments with update parameters from the range $0, 10^{-9}, 10^{-8}, \ldots, 1$. The performance differences were huge. In table 5.41 we show the best performing KBLA ($\sigma_u^2 = 0.0001$) together with $\sigma_u^2 = 0$ and $\sigma_u^2 = 1.0$. The results are compared to the results in

determine the implementation details for that visitor and it has therefore not been included in this test.

[HBG+07], where they describe and evaluate 3 variations of the AdaptEvE scheme, were the best one also won the tournament.

Table 5.41: Regret in the EvE challenge for 10^6 time steps averaged over 100 repetitions. Values are $\times 10^3$.

Env/Scheme	**Restart**	**Meta**	**Meta-p**	**KBLA 0.0001**	**KBLA 0.0**	**KBLA 1.0**
Constant	0.4 ± 0.02	2.5 ± 0.5	3.2 ± 0.3	9.9 ± 0.6	0.22 ± 4.3	20.0 ± 0.09
FrequentSwap	12.1 ± 0.1	14.0 ± 1.9	10.6 ± 1.3	15.4 ± 2.2	35.1 ± 2.1	48.6 ± 0.1
DailyVariation	6.9 ± 0.6	6.2 ± 0.7	6.1 ± 0.7	4.8 ± 0.5	63.7 ± 0.0	56.9 ± 0.1
WeeklyVariation	7.3 ± 0.2	4.8 ± 0.8	5.1 ± 0.9	5.0 ± 0.6	64.0 ± 0.0	57.2 ± 0.2
LongGaussians	7.4 ± 0.4	4.8 ± 1.6	4.3 ± 1.4	4.3 ± 0.7	120.3 ± 6.4	130.5 ± 0.2
Total	34.1	32.3	29.3	39.4	283.0	312.7

We see that the best KBLA seems very strong on the changing environments, especially on the "DailyVariationVisitor", but in terms of total regret we see that it cannot compare to either of the AdaptEvE schemes due to too much loss on the visitors with constant elements, the Constant and FrequentSwap vistors. The KBLA with zero as the update variance parameter handles the Constant visitor, although with high standard deviation.

Chapter 6

Discussion

In this chapter we discuss the results with the intention of understanding the behavior of the Kalman Bayesian Learning Automaton.

6.1 Performance in random walk environments

We argued earlier that random walk environments are the kind of environments the Kalman Bayesian Learning Automaton is intended for. The tracking capabilities of the Kalman filter is created exactly for such probalistic reasoning over time under uncertainty. Therefore it was perhaps not surprising that the performance of the KBLA turned out to be superior to all other tested schemes in this experiment.

When the noise of the environment was increased, we observed that the performance of the KBLA dropped, however increasing the speed of the environment did not seem to influence the performance of the automaton, but rather made it easier. When operating in an environment with higher noise the KBLA will consider each arm more uncertain, and therefore the variances of each arm will be larger which in turn leads to a wider range of possible values during sampling. Wider sampling value ranges drops the performance somewhat due to increased amount of overlapping values, but in a more rapidly changing environment the mean value of the reward distributions of the arms are likely to be further away from each other and thus reduces the amount of overlapping values.

6.2 KBLA in switching environments

In switching environments the performance of the KBLA was very dependent on its parameter configuration. Unlike random walk environments, the update variance parameter had to be 'tuned', either from knowledge of the environment or simple guessing. We observed that the configurations with the update variance parameter (σ_u^2) close to 0 had to little exploration to adapt when the arms switched. However, too large values of σ_u^2 resulted in too much exploration and the accuracy dropped significantly. There was definitely a compromise between adaption speed and accuracy. We also observed that increasing the frequency of the switching environment favored higher choices of σ_u^2.

It seems clear that finding the optimal σ_u^2 value in switching environments like this require good knowledge of the environment and its behavior. Still, with the right values of σ_u^2, which happened to be between 0.1 and 1.0 in our tests, the KBLA obtained adequate results. In fact, it was only surpassed by the AdaptEvE scheme.

The AdaptEvE scheme applies a change point detection mechanism that we saw proved very effective in identifying the new best arm after a changepoint. A KBLA with high σ_u^2 should also find the new best arm fast, but the constant exploration would hinder it from gaining the same accuracy as AdaptEvE. From another perspective, the changepoint mechanism in the AdaptEvE scheme is dependent on the value of the best arm to change. The KBLA however, does not have this weakness, even if there is no change in the current best option, it would still adapt if another arm 'silently' becomes the best.

An interesting scheme for handling switching environments would be the KBLA with $\sigma_u^2 = 0$ with the changepoint mechanism in AdaptEvE, and then restarting the KBLA at each changepoint. This would not handle the case of a non-changing optimal arm though.

6.3 Performance in the Goore game

The Goore game results revealed that small teams were easy for the KBLA. With the right parameters it converged quickly and obtained good accuracy. However when the number of players was increased to hundred, the automaton had to be tuned to reduce the amount of noise and to stop early convergence.

The parameter configuration was sensitive the amplitude and shape of the function used. We can understand this in terms of the arm selection process in the KBLA. When the amplitude of the function increases

the values between rewards increases, and in turn causes the distance between the mean values of the reward distributions of the automatons arms to increase and therefore a greater variance is necessary in the sampling process to explore the other arm sufficiently. For instance, if the update variance parameter is too small then it would take a too long time before the other arm is chosen in the sampling process.

Similarly for the shape of the function, if the function is flat then the distance between rewards are smaller and as such should also affect the parameter choice.

We also saw that it was possible to gain good solutions by increasing the sensor parameter. However, this had a certain effect on learning speed. From the parameter analysis, we know that when the sensor variance parameter is increased it takes more time before the arm variance reaches its fixed point and thus increases the amount of exploration initially. More initial exploration makes the KBLA less aggressive in the start and should therefore lower the learning(convergence) speed.

It was also clear from the results that the magnitude of the sensor variance parameter, the amount of initial exploration needed, was dependent on the amplitude of the function. Similarly to the update parameter, this property can be understood by the need for higher variance when the distance between rewards are bigger.

We found no perfect parameter configuration in our experiments and it is clear that this game needs further investigation in order to see whether it is possible to determine offline or online what the parameters of the KBLA should be for optimal performance.

6.4 Performance in zero-sum and non zero-sum games

The zero-sum game results was in general very favorable for the Kalman Bayesian Learning Automaton and clearly demonstrated its potential as a rational decision making unit. Specifically, in Game1, the KBLA players converged quickly to a solution, and even the optimal solution if the update variance was sufficiently low. A higher update variance parameter did not seem to influence the learning speed in this game and only dropped the maximum performance due to increased exploration.

Game2 showed rather similar results for the KBLA players, common for all was the rapid convergence rate, but also in this game the KBLA players with low update variance was the top performers. The other schemes however, with the exception of Poker and SoftMaxEWA, had trouble finding the optimal action for player 2.

In the mixed strategy game the KBLA players outperformed all other tested schemes and played very close to the optimal strategy. This time the KBLA with $\sigma_u^2 = 0$ was slightly behind, which makes sense, as converging towards a specific action is suboptimal because both arms should be played. The KBLA was not able find the true optimal strategy versus a perfect player, but the score was very close to the maximum attainable payoff.

The only non zero-sum game we tested, the Prisoner's Dilemma, did not reveal anything particularly new in terms of behavior, it rather just strengthened the view of the KBLA as a rational agent when configured properly.

6.5 Scalability

In random walk environments, scaleability did not seem like a big problem. Increasing the number of arms to 50 decreased the performance, but with such amount of randomness involved this was expected and it is hard to imagine how an algorithm can perform much better on average than the KBLA does in such environments.

The KBLA also approached the optimal mixed strategy when we applied it in a zero-sum game with five actions.

The scaleability aspect was also tested in the Goore game, in a different context. It was not the number of arms that increased, but rather the number of automatons operating together. Here, it became evident that the automaton became much more sensitive to its parameters in order to find the best solution when the team size increased.

6.6 Robustness

6.6.1 Sensitivity

Incorrect update variance parameter in random walks

It was clear that the update parameter influenced the performance when incorrectly specified, and this was more noticable when the number of arms increased. We also saw that a high update parameter gave good

results on the probability of selecting the optimal arm while a low update parameter was good in terms of regret. This seems reasonable, because when specifying a too low update parameter, noticing changes becomes harder and the KBLA stays longer on one arm before exploring others. On the other hand when the update variance parameter is specified too high more exploration is done and the KBLA will identify the optimal arm more often, thus the probability of selecting the optimal arm becomes higher, but the regret increases more rapidly because of the regular exploration of very bad arms.

For environments with high update variance, the performance for an incorrect parameter might be acceptable, because missing by a factor of 2 on the standard deviation on such an environment like this with true variance 2500 means incorrectly specifying as much as $2500-625 = 1875$ below or $10000-2500 = 7500$ above. If the environment instead had a true update variance of 1, then missing by a factor of standard deviation 2 would only be $1 - \frac{1}{4} = \frac{1}{3}$ below and $4 - 1 = 3$ above.

Incorrect sensor variance parameter in random walks

Incorrectly specified sensor variance in random walk environments gave rather surprising results. The true value did not give the best result. Instead, a slightly lower parameter gave the best result, and it was clear that it was better to specify a too small parameter than a too large.

When the value is too low, the automaton thinks the observations are more accurate than they really are and two things happen. First, the variance of the automatons arms decreases faster and the fixed variance point becomes smaller, which leads to overall less exploration. Still, since the update parameter is correct, it tracks environmental changes as it should, and thus the only difference is a slightly smaller variance on the best arm and it should therefore be a little more "aggressive" in the arm selection process. We can see this improve performance in certain cases, especially in two-armed environments where it might be beneficial to stay longer on one arm.

However, we saw that even in fifty-armed environments a smaller sensor parameter improved performance, but this was only in terms of regret, the probability of selecting the optimal arm was best when the true parameter was used. If we think about it, it makes sense, a KBLA with smaller sensor stays longer on what it thinks is the best arm, and even though it may not be the best arm, it does not waste too much time exploring very bad arms which has a drastic impact on regret.

Both wrong update variance parameter and sensor variance parameter in random walks

We again saw the importance of supplying the KBLA with a rather accurate update variance parameter. The accuracy of the sensor variance parameter did not have much influence in the easy environments, but in hard environments the performance was highly dependent on this value.

The reason for the sensor variance parameter not being important in easy random walk environments is likely due to the fact that the update variance parameter has greater impact on the amount of exploration as we saw in section 5.2, the parameter analysis. And therefore, since it is relatively simple for the KBLA to distinguish arms in easy random walk environments, the amount of noise it assumes do not have a great impact.

When the environment is hard however, distinguishing the arms needs a finer balance between exploration and exploitation, and thus when specifying the sensor variance parameter too high there is likely too much noise in the arm selection process to be able to track good arms. For low values of the sensor variance parameter, we can use the same reasoning as in the previous subsection, that is, the KBLA is more "aggressive" due to lower exploration, and stays longer on good arms.

6.6.2 Binary feedback

While it was clear from the results that the Kalman Bayesian Learning Automaton was very sensitive to its parameter configuration when receiving binary feedback, the binary experiments also showed that it was in fact able to solve also binary zero-sum games and the binary Goore game with high degrees of accuracy. Even in the EvE challenge it was able to perform very good on some environments. Unfortunately though, it was very evident that obtaining adequate performance in binary environments relied on specific parameter configurations.

Because of the KBLA's assumption of normally distributed feedback, we were not to surprised over its sensitivity to the reward/penalty paradigm. Binary environments contains another kind of noise, the uncertainty caused by the probability of receiving a zero or one. The KBLA is likely extra sensitive to such noise because the Kalman filtering estimate is rather aggressive and if a few "unlucky" zeros is received as feedback on a good arm it will quickly regard this as a bad arm. This seems to indicate that it is worthwhile to reduce the amount of exploration in binary environments, to stay longer on each arm to obtain a reasonable estimate, while still do some exploration to follow environmental changes. It is clearly a very

fine line, and we would hesitate to apply the KBLA any such cases unless the dynamics of the environment is sufficiently regular to allow for offline parameter tuning.

6.7 Impressions on tuning the parameters of the KBLA compared to other adaptive schemes

It is clear that the performance of the KBLA is highly dependent on the values of its parameters. However, all adaptive schemes we have used in our comparison need parameter tuning, and also influences their performance. The question is then whether it is easier or harder to tune the parameters of the KBLA.

If we first consider the schemes that applies the exponential recency-weighted average (EWA) method, that is, the GreedyEWA, SoftMaxEWA and AdaptivePursuit schemes we know that a single parameter determines their "response rate". That parameter, here denoted α, determines how quickly it forgets older rewards, and thus puts more emphasize on the most recent rewards. There is no direct equivalent parameter in the KBLA, put Kalman filtering in itself puts strong emphasize on recent rewards as long as it believes the measurements to be sufficiently accurate, that is, when it is supplied with a rather small sensor variance parameter. So, it is certainly possible to make the KBLA more "aggressive" to recent rewards, but some may argue that this is harder to tune than the α parameter in the EWA schemes due to the fact that the sensor variance parameter also influences other factors such as initial exploration in the KBLA.

Furthermore, the α parameter is not the only parameter to determine the performance of the EWA schemes, the GreedyEWA scheme has an exploration rate parameter ϵ, which determines the probability of exploring other actions. The SoftMaxEWA scheme has τ which also determines the exploration rate, though in a completely different manner than GreedyEWA, while AdaptivePursuit has two other parameters, β and P_min where both affect the exploration amount. Empirically, we found it much easier to tune the ϵ parameter compared to the τ parameter[1]. For AdaptivePursuit we tested a wide range of parameters and found in to be most sensitive to its P_{min} value, which is not surprising since it ensures that even the worst action is played with a certain probability.

With some knowledge of the environment we would argue that the KBLA is easier to tune than both AdaptivePursuit and SoftMaxEWA, due to the update variance parameter having a more clear constant

[1]This seems similar to the impressions in [SB98], where they argue that SoftMax methods are hard to tune.

impact on exploration. On the contrary, compared to the GreedyEWA scheme, it may be harder, since the ϵ parameter has a very clear purpose. If we want 10% exploration then we set it to 0.10, nice and easy, while achieving a rate of 10% exploration in the KBLA with just parameter tuning is nontrivial because then one would also have to consider a, perhaps dynamic, feedback range and choose the update parameter accordingly.

Considering the rest, that is, AdaptEvERestart and AdapEvEMetaBandit, we feel that, since most our environments were not completely suited for their mechanics, it would be unfair to call them "hard to tune" from trying to make them work in environments where they really don't fit. Still, some of our environments, like the first switching configuration, did match their strengths, and we were left with some impressions. First, tuning for the change point mechanism in a normally distributed switching environment did not prove as challenging as we initially thought. We thought perhaps the noise in feedback could trigger the change point mechanism, but we did not see any such deviation in our ensembles even with rather low values of λ, the limit parameter. For the meta scheme, the meta length parameter MT did not seem to have a large impact on performance as long as we used values over 50 to let the meta phase build up a reasonable average.

Overall, we feel the strength of the parameter configuration in the KBLA is due its flexibility and not being overly sensitive. If only initial exploration is needed, for instance in a environment where it is important to converge in the end, it is just a matter of increasing the sensor variance parameter. On the other hand, should there be a need for a more aggressive arm selection process, then lowering the sensor variance parameter would do the job, with the effect that it stays longer one arm due to less exploration but also puts more trust in recent rewards.

The downside and most notable weakness in the parameter configuration as we see it is definitely the fact that one has to consider the feedback range when tuning parameters. We especially saw this in the Goore game, but this would be true for most of the other experiments as well if we increased or decreased the feedback ranges there. This is not the case for schemes following a "probabilistic" approach, as the Greedy variants, SoftMax and AdaptivePursuit, where the amount of exploration is far less dependent on the actual value range of the feedback.

6.8 Application domains

While possible application domains for the KBLA is undoubtedly an interesting area for further work, several impressions arises from the strengths and weaknesses we have revealed.

Its strongest point is definitely tracking an environment with multiple Gaussian processes. Considering that Gaussian processes is a frequent recurring pattern in nature, science and engineering this seems like no small feat. Some examples of randomly changing environments according to [Gar79][ch7] is the random movement of molecules in liquids and gas (brownian motion), the diffusion of heat through materials, the spread of rumours and the spread of disease. Of course, for the problem to be valid for the KBLA it most also somehow involve decision making, e.g in a case where an agent has observe randomly moving objects and at some points determine the nearest one.

The KBLA also obtained very good results in zero-sum games. Immediately, it might be hard to see exactly where this is beneficial to a system, except from proving the rationality of an agent. However, [NT89] contains several examples of applications where the theoretical and empirical results from zero-sum games are used as the motivation factor. They claim that learning automatons first and foremost are applicable in large systems with a small number of actions with examples including traffic routing, computer scheduling and decision making in economic networks. Specifically, they considered the case of routing where several automatons worked together with only two actions and showed that automatons[2] outperformed fixed rules because they were able to adapt to the dynamics of the network and play a mixed strategy.

Thus, since a tuned KBLA were close to the optimal mixed strategy in all our tested zero-sum games, we believe it definitely has potential in applications where it is important to adapt and play a mixed strategy. However, since it is unlikely that any such applications where multiple automatons interact can be viewed in terms of a Gaussian process, we see the same challenges as before occurring. That is, if it is not possible to decide the values of reinforcement, tuning the parameters for optimal performance may prove nontrivial. Still, systems can often be tested and validated rigorously in advance before being applied in a critical setting, so in many cases this might not be a huge problem.

[2]They used the L_{R-I} and L_{R-P} learning automatons.

6.9 Thoughts on the Kalman filtering approach

Kalman filtering is able to provide an optimal estimate for linear Gaussian models with simple recursive calculations. When applied in a bandit setting, the estimates are a bit more noisy since only one arm can be observed each time step, but it was clear that the KBLA succesfully managed to track multiple arms with high accuracy in random walk environments, which are described by a linear Gaussian state-space model. The other environments however, don't follow a linear Gaussian model and as such the parameters of the KBLA has to be tuned to balance the amount of exploration needed.

For non-Gaussian and indeed, general case models, there also exist filtering approaches for estimating distribution parameters. The question is whether these other filtering algorithms also can be extended to work in a bandit problem setting. Examples of these approaches include the Extended Kalman filter, the Unscented Kalman filter and Particle filters. Especially particle filters, perhaps better known as sequential Monte Carlo methods, has received a lot of attention lately, because they allow for recursive state parameter estimation of virtually any dynamic model[DD06]. We leave this question for further work, but we fear that there is a trade-off between higher accuracy and computational simplicity.

Chapter 7

Conclusion

7.1 Conclusion

In this thesis we have investigated a new interesting approach to non-stationary multi-armed bandit problems, the Kalman Bayesian Learning Automaton(KBLA). Our empirical study included testing the KBLA in randomly changing environments, switching environments, the Goore game, zero-sum and non-zero-sum games. We also looked at the robustness and scalability of the approach. We revealed several strengths and weaknesses of the automaton, the results can be summed up in the following.

- The KBLA performed excellent in randomly changing environments. No existing solutions we know of were able to obtain even nearly the same performance in such environments. In switching environments where it was possible to detect a change in the best option, the KBLA was not able to outperform AdaptEvE, but still obtained adequate performance with tuning. Also, in switching environments where no change in the best option occurred, the KBLA was the top performer.
- In most tested multiplayer games the KBLA was able to solve the game with the right parameter configuration. In the Goore game, the need for the right update variance parameter configuration became very evident, and it was clear that there was a fine line between too little and too much exploration. We also saw that it was possible for the KBLA to solve this game by only increasing the initial exploration, unfortunately this solution was also dependent on the shape and amplitude of the feedback function. Still, it should be noted that all tested schemes had problems with this

game , and as such, we don't necessary consider the KBLA suboptimal to any of the other tested schemes in this game.

In zero-sum and non-zero-sum games the KBLA approached an almost optimal rational behavior in all tested games. It found both pure strategy and mixed strategy solutions, with rapid learning rate.

- As for the scalability aspect we observed that in multi-armed bandit problems the KBLA scaled well with the number of actions, even a problem with 50 arms was tackled with relatively high accuracy. We also saw that it was able to get near the optimal mixed strategy when the number of actions were increased in a two-player zero-sum game .

  In the Goore game however, it was clear that increasing the players in the team significantly hindered the performance of the KBLA. Parameters had to be finely tuned in order to balance the line between too much exploration and early convergence.

- In randomly changing environments the KBLA was surprisingly robust to wrong parameter configuration. Some configurations of the sensor variance parameter even yielded better results than using the true sensor variance parameter of the environment. The performance was more dependent on the update variance parameter, and we saw that all incorrect values of this parameter dropped performance.

  In all the other experiments the parameters of the environments were unknown, and it was clear the parameter configuration of the KBLA definitely had an impact on the maximum performance obtainable, but this was also the case for all tested adaptive schemes. Tuning the parameters of the KBLA was in general a simple task, except when playing the Goore game with a large team.

  The KBLA was also able to obtain very good results when applied to binary environments. In fact, it was among the top performers in both binary zero-sum games and binary Goore game. Even to the wide range of environments in the EvE challenge it performed adequately. However, with such feedback the need for correct tuning of parameters became very evident.

We believe this says a lot about the strength of a Kalman filtering tracking approach in multi-armed bandit problems. Clearly, it is able to track a wide range of environments with high accuracy. Its main weakness is its dependence on the parameter configuration, which we have seen may prove hard to tune in some environments like the Goore game and binary environments. However, in spite of the apparent parameter tuning weakness we consider the KBLA a strong approach to the bandit problem and undoubtedly has a lot of potential in artificial intelligence applications where rational decisions must be made under uncertainty.

7.2 Further Work

We have seen that in some environments the KBLA needs extensive tuning to perform well, but combining ideas from other learning schemes might overcome these limitations. Several interesting possibilities for extending the KBLA comes to mind:

- When it is important to converge in the end, a possibility would be a KBLA scheme using a constant like ϵ_n-greedy to reduce the sensor parameter and/or update parameter over time and therefore only be adaptive initially.
- We have seen that the KBLA with zero update variance ($\sigma_u^2 = 0$) is very good at quickly converging towards an optimal arm, but often lacks enough exploration to track sudden changes in the environment. A possible extension would be to combine this configuration with the change point mechanism from the AdaptEvE schemes.
- The update parameter leads to constant exploration and should not be too high if the environment is slowly moving or stationary. Is it possible to determine online if the value should be increased or decreased? A meta scheme is a one possible candidate for the job. A Meta KBLA scheme would be interesting, but then there is the question of how the sensor and update variance parameters of the meta KBLA should be determined. Perhaps the parameterless UCBNormal (or UCBTuned in binary environments) would be more suited as a meta scheme for the KBLA.
- An alternative to a meta scheme is "meta learn" the parameters. This has recently been done for the SoftMax algorithm in [Sik08], and may provide a starting point.

Thinking back at our approach and what we have done, many thoughts arise regarding areas of further work:

- Another approach should be considered to really understand the behavior of the KBLA in the Goore game. Is it possible to calculate what the parameters should be for optimal performance?
- Since, in general, all tested schemes were very sensitive to its parameter configuration in the normally distributed Goore game and, we have to ask ourselves whether it was right to use this game in our benchmark. Perhaps our way of approaching this game was wrong, but we feel that it did not

say too much about the true performance of the KBLA in large team settings, and as such we see testing the KBLA in different team settings as important further work. Especially real-time systems comes to mind, because they are often decentralized and calls for adaptive decision making under uncertainty. In addition, delays and distortions in feedback are common so in a way this would test several aspects of the KBLA at the same time.

- Sinusoidal patterns are frequently recurring in nature and science. We have only briefly tested the KBLA in sinusoidal environments (see the EvE challenge and appendix A.1.1) and therefore must be studied further for a comprehensive understanding of the KBLA in these environments.
- A huge advantage of applying the Kalman filtering approach as a state estimation technique is its computational simplicity. However, it is really only linear Gaussian environments that is a "perfect" match for this filter. It would be interesting to see whether more versatile state estimation techniques like particle filtering could work in a bandit setting.

Bibliography

[ACF02] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235256, 2002.

[ACFS98] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and R. E Schapire. Gambling in a rigged casino: The adversarial multi-armed bandit problem. *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, 1998.

[ACFS03] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):4877, 2003.

[AHC+06] P. Auer, Z. Hussain, N. Cesa-Bianchi, L. Newnham, and J. Shawe-Taylor. Exploration vs. exploitation challenge. http://pascallin.ecs.soton.ac.uk/Challenges/EEC/, 2006.

[BN09] T. Br\aadland and T. Norheim. Empirical evaluation of the bayesian learning automaton family. Master's thesis, University of Agder, 2009.

[Cho07] Graham Kendall; Xin Yao; Siang Yew Chong. *The Iterated Prisoners' Dilemma: 20 Years on*. World Scientific Publishing Company, illustrated edition edition, May 2007.

[DD06] Corentin Dubois and Manuel Davy. Joint detection and tracking of Time-Varying harmonic components: a flexible bayesian approach. *IN "IEEE TRANSACTIONS ON SPEECH, AUDIO AND LANGUAGE PROCESSING*, 2006.

[Gar79] Martin Gardner. *Mathematical Circus: More Games, Puzzles, Paradoxes, and Other Mathematical Entertainments.* Random House Childrens Books, 1st edition, September 1979.

[GB10] O. C Granmo and Stian Berg. Solving Non-Stationary bandit problems by random sampling from sibling kalman filters. In *To Appear in Proceedings of The Twenty Third International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems IEA-AIE 2010*. Springer-Verlag, June 2010. To Appear in Proceedings of the.

[GCSR03] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis*. Chapman & Hall, 2 edition, July 2003.

[Gra08] O.-C. Granmo. A bayesian learning automaton for solving Two-Armed bernoulli bandit problems. In *Machine Learning and Applications, 2008. ICMLA '08. Seventh International Conference on*, pages 23–30, 2008.

[HBG+07] C. Hartland, N. Baskiotis, S. Gelly, M. Sebag, and O. Teytaud. Change point detection and meta-bandits for online learning in dynamic environments. *CAp*, pages 237–250, 2007.

[HNP+03] S. Ho, H. Nassef, N. Pornsinsirirak, Y. C Tai, and C. M Ho. Unsteady aerodynamics and flow control for flapping wing flyers. *Progress in Aerospace Sciences*, 39(8):635681, 2003.

[HV04] Shaun Hargreaves-Heap and Yanis Varoufakis. *Game Theory: A Critical Introduction*. Routledge, 2 edition, April 2004.

[IK03] R. Iyer and L. Kleinrock. QoS control for sensor networks. In *Communications, 2003. ICC '03. IEEE International Conference on*, volume 1, pages 517–521 vol.1, 2003.

[May79] Peter S Maybeck. *Stochastic models, estimation, and control*, volume 141 of *Mathematics in Science and Engineering*. Academic Press, 1979.

[NRTV07] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, September 2007.

[NT89] Kumpati S. Narendra and Mandayam A. L. Thathachar. *Learning Automata: An Introduction*. Prentice Hall, May 1989.

[Nyb] Svein Olav Nyberg. Statistikk en bayesiansk tilnaerming. Unpublished.

[RN02] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2 edition, December 2002.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, March 1998.

[Sik08] R Sikora. Meta-learning optimal parameter values in non-stationary environments. *Knowledge-Based Systems*, 21(8):800–806, 2008.

[SL08] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, December 2008.

[Thi05] D. Thierens. An adaptive pursuit strategy for allocating operator probabilities. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, page 1546, 2005.

[TK93] Brian Tung and Leonard Kleinrock. Distributed control methods. *IN PROCEEDINGS OF THE 2ND INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE DISTRIBUTED COMPUTING*, 206:206—215, 1993.

[Tse74] M.L. Tsetlin. *Automaton Theory and Modelling of Biological Systems (Mathematics in Science & Engineering)*. Academic Press Inc.,U.S., 1974.

[VM05] J. Vermorel and M. Mohri. Multi-armed bandit algorithms and empirical evaluation. *Lecture notes in computer science*, 3720:437, 2005.

Appendices

Appendix A

Results that were left out of the main report

A.1 Multi-armed bandit problems

A.1.1 Sinusoidal environments

Sinusoidal environments are a frequently recurring pattern in nature. The experiments with sinusoidal environments was left out of the main report because only one sinusoidal environment has been tested and we therefore feel the coverage is not as comprehensive as it should be. This environment can be seen in figure A.1.

The regret on this environment is listed in table A.1.

Figure A.1: Sinusoidal environment with 3 arms

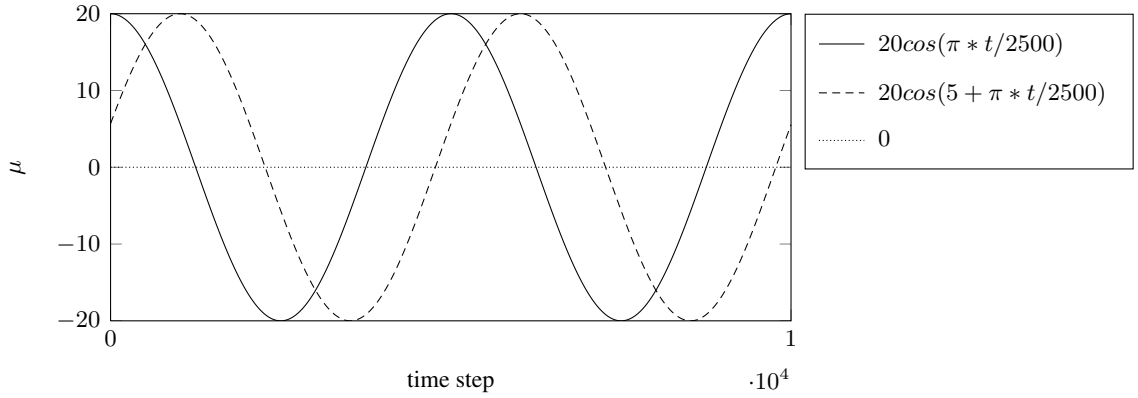


Table A.1: Results for regret on the sinusoidal environment in figure A.1.

Player / Iteration	**10**	**100**	**1000**	**2000**	**10000**
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.0$	1.72	1.72	183.05	833.98	4525.64
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.001$	1.72	1.72	143.25	145.54	2888.11
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.01$	1.72	1.72	46.94	48.59	1738.6
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.1$	1.72	1.72	6.91	11.6	55.82
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.5$	1.72	2.03	17.4	29.97	142.3
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 2.0$	1.72	4.29	39.24	69.98	338.94
Poker	2.31	2.31	183.36	556.66	4214.21
Pursuit 0.05	5.19	12.98	194.39	1025.97	4963.33
AdaptivePursuit α =0.1,β =0.8, P_{min} =0.05	4.91	27.0	103.55	187.91	834.44
AdaptivePursuit α =0.8,β =0.8, P_{min} =0.01	5.05	31.38	56.74	73.32	241.82
Greedy ϵ =0.05	1.87	4.16	189.92	262.88	3790.66
Greedy ϵ =0.02	1.81	2.79	181.95	278.29	3804.93
GreedyEWA ϵ =0.05 α =0.3	1.95	4.29	35.29	64.27	328.0
GreedyEWA ϵ =0.02 α =0.9	1.77	2.69	14.84	26.43	198.87
SoftMaxEWA τ=0.5 α=0.3	3.35	22.89	183.45	398.05	1914.4
SoftMaxEWA τ=0.1 α=0.3	1.72	1.72	171.79	251.44	2597.55
UCBNormal	9.71	27.98	189.71	196.73	3778.17
AEγRestart $\delta = 0.005$, $\lambda = 5.0$	9.71	27.98	60.41	231.89	1864.41
AEMetaBanditC $\delta = 0.005$, $\lambda = 5.0$, $MT = 100$	3.43	3.43	12.54	29.92	160.43

A.2 Goore game

A.2.1 Results for the flat Gaussian function

The results for the 100 player Goore game on the flat Gaussian function with amplitude 20 can be seen in table A.2.

Table A.2: Results for 100 player Goore game on flat Gaussian function with amplitude 20 and where 20 yes votes are optimal.

Team / Iteration	**10**	**100**	**1000**	**2000**	**10000**	**100000**
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.0$	43.27	36.74	34.2	33.49	32.01	29.82
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 1e-05$	46.27	36.94	34.41	34.04	32.78	23.8
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.0001$	39.8	35.06	32.04	31.77	25.39	27.87
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.001$	44.4	37.58	31.37	31.46	31.86	32.52
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.01$	46.73	41.56	36.34	37.78	38.0	38.85
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.0$	43.67	37.44	35.46	34.89	33.28	31.6
KBLA $\sigma_z^2 = 5.0$, $\sigma_u^2 = 0.0$	49.53	37.74	30.61	30.06	29.64	28.26
KBLA $\sigma_z^2 = 10.0$, $\sigma_u^2 = 0.0$	43.67	38.52	30.36	29.02	27.78	26.13
KBLA $\sigma_z^2 = 50.0$, $\sigma_u^2 = 0.0$	48.33	45.48	30.82	23.29	23.13	22.52
ϵ_n-Greedy $c = 0.3$, $d = 0.1$	51.67	36.12	31.06	30.64	30.48	30.41
ϵ_n-Greedy $c = 0.6$, $d = 0.1$	49.93	42.9	27.07	26.17	25.55	25.42
ϵ_n-Greedy $c = 1.0$, $d = 0.1$	51.2	50.92	25.91	24.46	23.49	23.23
Greedy ϵ =0.1	5.73	5.22	17.46	26.97	28.58	23.92
Greedy ϵ =0.2	11.0	10.5	25.31	26.69	24.38	20.14
GreedyEWA ϵ =0.1 α =0.3	9.13	10.34	42.83	39.54	39.34	39.86
GreedyEWA ϵ =0.05 α =0.3	4.73	5.32	40.76	37.04	38.73	39.05
SoftMaxEWA τ=0.5 α=0.3	35.27	50.26	49.74	50.01	49.62	49.69
SoftMaxEWA τ=0.1 α=0.9	1.07	12.14	44.23	47.65	48.4	48.52
Pursuit 0.005	1.2	1.08	1.99	2.0	2.0	2.0
AdaptivePursuit α =0.8,β =0.8, p_{min} =0.05	1.07	1.34	0.97	1.22	1.01	1.08
AdaptivePursuit α =0.8,β =0.8, p_{min} =0.01	0.8	0.7	0.83	1.19	1.37	1.2
Poker	45.13	40.98	38.62	38.4	38.4	38.4
UCBNormal	33.33	10.0	2.0	0.0	1.0	0.1

A.3 Zero-sum games

A.3.1 Tournament results for Rock, paper, scissors extended

The results for this tournament are shown in table A.3

Table A.3: Results for the RPS extended tournament

Player / Iteration	**10**	**100**	**1000**	**2000**	**10000**
KBLA $\sigma_z^2 = 1.0$, $\sigma_u^2 = 0.01$	1.45	61.0	975.74	2120.06	11576.49
ϵ_n-Greedy $c = 0.3$, $d = 0.1$	-0.54	16.87	21.05	-153.43	-1862.9
Greedy ϵ =0.05	1.77	-7.19	-232.83	-335.93	-9.39
Greedy ϵ =0.1	1.37	7.93	20.41	105.5	1636.88
GreedyEWA ϵ =0.1 α =0.3	1.44	-18.14	-513.96	-1073.0	-5515.36
SoftMax-EWA-N τ=0.5 α=0.3	1.01	38.04	541.63	1179.55	6828.99
Pursuit 0.01	-0.42	7.54	10.65	-515.05	-7245.43
AdaptivePursuit α =0.8,β =0.8, P_{min} =0.01	-2.12	-89.05	-1334.14	-2739.18	-14030.32
Poker	0.27	4.88	-258.18	-557.74	-3240.84
UCBNormal	-4.24	-21.87	769.62	1969.22	11861.87

A.4 Sensitivity

A.4.1 Sensitivity to incorrectly specified sensor variance parameter and update variance parameter

The results for the easy and hard two-armed random walk environments are included here in table A.4.

Table A.4: Regret at $t = 2000$ for incorrectly specified sensor variance parameter and update variance parameter in two-armed random walk environment with true $\sigma_z^2 = 50$ and true $\sigma_u^2 = 50$.

Belief	$\frac{1}{5}\sigma_z$	$\frac{1}{4}\sigma_z$	$\frac{1}{3}\sigma_z$	$\frac{1}{2}\sigma_z$	σ_z	$2\sigma_z$	$3\sigma_z$	$4\sigma_z$	$5\sigma_z$
$\frac{1}{5}\sigma_u$	188.66	190.88	186.89	187.21	182.11	175.71	177.89	177.67	183.92
$\frac{1}{4}\sigma_u$	171.36	172.19	176.81	171.51	165.79	162.57	163.01	162.16	166.25
$\frac{1}{3}\sigma_u$	149.13	146.21	143.47	141.94	142.24	137.73	139.56	146.47	152.43
$\frac{1}{2}\sigma_u$	106.3	110.86	108.49	106.38	110.54	109.95	117.28	121.08	126.42
σ_u	82.09	82.07	82.35	83.37	85.59	90.85	100.04	108.82	117.31
$2\sigma_u$	108.71	109.5	109.5	109.33	111.6	119.05	127.74	136.37	145.65
$3\sigma_u$	141.33	141.85	141.65	142.23	144.17	149.71	158.21	167.05	175.53
$4\sigma_u$	171.18	171.41	171.61	172.29	173.26	179.06	186.63	194.28	202.99
$5\sigma_u$	199.42	199.27	199.33	200.1	200.87	205.5	212.19	219.16	226.77

Table A.5: Regret at $t = 2000$ for incorrectly specified sensor variance parameter and update variance parameter in two-armed random walk environment with true $\sigma_z^2 = 50$ and true $\sigma_u^2 = 1$.

Belief	$\frac{1}{5}\sigma_z$	$\frac{1}{4}\sigma_z$	$\frac{1}{3}\sigma_z$	$\frac{1}{2}\sigma_z$	σ_z	$2\sigma_z$	$3\sigma_z$	$4\sigma_z$	$5\sigma_z$
$\frac{1}{5}\sigma_u$	565.58	555.92	497.59	433.79	405.08	445.49	499.55	535.41	572.77
$\frac{1}{4}\sigma_u$	563.81	506.49	465.8	420.93	385.21	441.46	493.82	535.59	569.73
$\frac{1}{3}\sigma_u$	504.67	484.98	458.23	401.4	363.95	432.64	485.89	533.74	572.27
$\frac{1}{2}\sigma_u$	476.86	459.95	422.47	378.7	347.81	421.96	485.1	532.26	570.51
σ_u	385.6	367.52	350.17	322.0	343.62	427.32	490.45	538.25	579.26
$2\sigma_u$	362.6	347.08	333.53	332.15	375.53	463.58	523.27	569.51	607.61
$3\sigma_u$	378.98	369.09	357.54	358.91	407.62	496.63	557.16	599.89	633.14
$4\sigma_u$	402.81	392.11	383.61	381.93	437.63	524.42	582.39	622.5	656.76
$5\sigma_u$	430.04	419.84	408.17	409.01	461.1	546.06	603.1	644.79	675.13

Appendix B

Missing algorithms from chapter 2

B.1 ϵ_n-Greedy

Algorithm 11 Pseudocode for ϵ_n-Greedy

Parameters: $c \in \mathbb{R}^+$
Init: $Q_i = 0$ for $i \in 1 \dots K$
loop
  $\epsilon = \frac{cK}{d^2 n}$
  if $Random \in \mathbb{R}[0,1] < \epsilon$ **then**
    $i = Random \in \mathbb{Z}[1, K]$
  else
    i = $\arg\max \frac{Q_j}{n_j}$ where$j \in \{1..K\}$
  end if
  n_j = n_j + 1
  r = feedback from action a_i
  Q_i = $Q_i + r$
end loop

B.2 UCBTuned

Algorithm 12 Pseudocode for UCBTuned

Init: Play each action once
loop
  i = $\arg\max_j \; m_j + \sqrt{\frac{\ln n}{n_j} \min\{\frac{1}{4}, V_j(n_j)\}}$ for $j \in \{1 \dots K\}$
  r = feedback from action i
  Update mean m_i from r and increment both n_i and n
end loop

B.3 Poker

[BN09]'s interpretation of the Poker scheme described in [VM05] is shown in listing 13.

Algorithm 13 Pseudocode for Poker

Init: $n_j = 0$, $r_j = 0$, $r_j^2 = 0$ for $j \in \{1 \dots K\}$
Init: Select two random actions at least twice and update the corresponding r_j, r_j^2, n_j
for $t \in \{1 \dots T\}$ **do**
  $q = |\{i, r_i > 0\}|$
  $i_0 = \arg\max \mu_i$
  $i_1 = j$ such that $|i, u_i > u_j| = \sqrt{q}$
  $\delta_\mu = (\mu_{i_0} - \mu_{i_1}/\sqrt{q}$
  $\mu^* = \arg\max \mu_i$
  $p_max = -\infty$
  $i = undefined$
  for $j \in \{1 \dots K\}$ **do**
    if $n_j > 0$ **then**
      $\mu = u_j$
    else
      $\mu = \hat{E}_{k,n[k]>0[\mu[k]]}$
    end if
    if $n_j > 1$ **then**
      $\sigma = \sigma_j$
    else
      $\sigma = \hat{E}_{k,n[k]>0[\mu[k]]}$
    end if
    $p = \mu + \delta_\mu(T - t) \int_{\mu* + \delta_\mu}^{\infty} N(x, \mu_j, \frac{\sigma_j}{\sqrt{n_j}} dx$
    if $p > p_max$ **then**
      $p_max = p, i = j$
    end if
  end for
  $x =$ reward from action i_{max}
  $r_i = r_{i_{max}} + x$
  $r_i^2 = r_{i_{max}}^2 + x_i^2$
  $n_i = n_i + 1$
end for