Stochastic Optimization in Dynamic Environments

With applications in e-commerce

by

Olov Andersson and Spencer Bastani

Master's Thesis in Computing Science, 20 credits (30 ECTS credits)

Internal supervisors:
Thomas Hellström,
Department of Computing Science, Umeå University
Lars Eldén,
Department of Mathematics, Linköpings Universitet

External supervisor:
Daniel Bernholc,
TradeDoubler AB

Umeå February 2007

Abstract

In this thesis we address the problem of how to construct an optimal algorithm for displaying banners (i.e advertisements shown on web sites). The optimization is based on the revenue each banner generates, with the aim of selecting those banners which maximize future total revenue. Banner optimality is of major importance in the e-commerce industry, in particular on web sites with heavy traffic. The 'micropayments' from showing banners add up to substantial profits due to the large volumes involved. We provide a broad, up-to-date and primarily theoretical treatment of this global optimization problem. Through a synthesis of mathematical modeling, statistical methodology and computer science we construct a stochastic 'planning algorithm'. The superiority of our algorithm is based on empirical analysis conducted by us on real internet-data at TradeDoubler AB, as well as test results on a selection of stylized data sets. The algorithm is flexible and adapts well to new environments.

Keywords: global optimization, function models, MDP, Markov decision processes, dynamic programming, exploration vs. exploitation, banners, optimization, e-commerce.

Acknowledgements

We would like to thank our supervisor Daniel Bernholc at TradeDoubler AB and also Adam Corswant for helping us with the database system. We would also like to thank our examiners Lars Eldén and Thomas Hellström for providing supervision on the theoretical work. Finally, thanks to all the staff at TradeDoubler for their friendliness and role in providing a great work-environment.

Nomenclature

Most of the reoccurring abbreviations and symbols are described here.

Symbols

One-step Approach

$R(w)$	The revenue R as a function of the weight vector w.
r_k	The CPI of element k.
m_k	Number of impressions on element k (in some time interval).
C_k	Number of clicks on element k (in some time interval).

The Planning Approach

N	The length of the horizon.
T	The set of epochs, usually $T = \{1, 2, \ldots, N\}$.
S	The discrete set of states (outcomes).
a	An action, $a = j$ is the selection of ad-element j.
h_t	The vector of all ad-element selections and outcomes up to time t.
H_t	The space of all possible history vectors h_t.
$d_t(h_t)$	An action, when one wants to be specific of how it depends on h_t and t.
ξ	The reward function.
π	A policy. This is a vector of decisions.
v_N^π	The expected total reward over the whole horizon.
u_t^π	The expected total reward from epoch t and onward.

Statistics

Y_t	The Random Walk process (time series) at time t.
X_t	The noisy measurement of The Random Walk process at time t
σ_s	The sample error involved in the estimation of CPI.
σ_v	The volatility parameter in the Random Walk Model.
P_t	The Prediction Operator.
θ	The coefficient in the best linear predictor of X_t .

Abbreviations

BLUP	Best Linear Unbiased Predictor
DACE	Design and Analysis of Computer Experiments
EGO	Efficient Global Optimization
CPI	Clicks per N impressions.
MDP	Markov Decision Process
pdf	Probability density function
cdf	Cumulative distribution function

Contents

Chapter 1

Introduction

This text is written in 2006 as a master's final thesis as a collaboration between Olov Andersson (Umeå University) and Spencer Bastani (Linköping Institute of Technology) with Daniel Bernholc as supervisor at TradeDoubler AB. Examiners are Thomas Hellström [1] and Lars Eldén [2]. The thesis is based on research in dynamic content optimization conducted at TradeDoubler AB at their headquarters in Stockholm in 2006.
Founded in 1999, TradeDoubler is a European provider of online marketing and sales solutions. Industry leaders all over Europe have partnered with TradeDoubler to work with performance-based marketing and to manage online relationships, in order to increase sales and improve online business. TradeDoubler is headquartered in Sweden with a presence in 16 other markets, and customers include Apple Store, Dell, Telia Sonera, eBay and Kelkoo.[3]
In general terms, the problem dealt with in this thesis is that of optimizing an unknown function. The point of departure is the characterization of this function. By unknown function we here mean a functional relationship transforming input values into output values when the only information we have access to is input-output pairs of observations. In the literature this is referred to as 'black-box' optimization. This is in our view the most straightforward way of characterizing a functional relationship while keeping the assumptions regarding the functional relationship to a minimum. A theoretical account of black-box optimization consists of two parts. The first one is an approximation of the function, known as the 'surrogate' or 'meta-model'. The second part is finding the optimum of this approximation. We give an overview of some of the most popular function modeling techniques, in particular a recent function model known as DACE (Design and Analysis of Computer Experiments) which was originally constructed for the evaluation of computationally expensive computer simulations. In connection to this model we present the Efficient Global Optimization (EGO) algorithm which is a global optimization algorithm specifically designed for evaluation of functions which are expensive or impractical to evaluate. We believe that these two algorithms or approaches have a general applicability on a wide variety of problems similar to that dealt with in this thesis and should be of interest to anyone working with optimization in unknown environments.

[1] Department of Computing Science, Umeå University
[2] Scientific Computing, Department of Mathematics, Linköping University
[3] Text Source: www.tradedoubler.com Visit the web site for further information.

As our experience has grown working with the particular application treated in this thesis, the relative importance of optimization compared to function modeling has become clearer. Measuring profitability of banners on the internet is a difficult endeavor because of the large data-sets which have to be dealt with. These data-sets are often inflicted with many unpredictable changes because of the constant need to update, tailor and modify e-commerce products in this very fast-paced industry. However the most important difficulty, and the real essence of an effective banner algorithm, is the prediction of the preferences of web site visitors. These issues create what we refer to as the 'Dynamic Environment'. When optimizing in a dynamic environment, without sufficient data, it is not possible to reach any precision in a black-box optimization algorithm with a large parameter-space. In addition, traditional black-box optimization algorithms, focus exclusively on exploration of the parameter-space, and the actual exploitation of the results is carried out after this explorative stage. When optimizing in dynamic environments, exploration and exploitation must be carried out simultaneously. Given these insights it leads us to consider planning algorithms that can balance these two factors. In particular the planning algorithm offers a formulation which expresses optimality with respect to a planning horizon. This is particularly useful in our application since content shown on the internet is usually only used a fixed length of time before it becomes obsolete. Our model makes statistically precise how various parameter choices affect the uncertainty of the solution. The central component of the model is the predictor. The predictor is the instrument used to predict future profitability of banners, and we provide two distinct Best Linear Unbiased predictors, based on two different assumptions regarding the way the uncertainty enters into the model. The predictors are distinct in the sense that one of them incorporates measurement error in the observed values. The optimization is carried out by using Stochastic Dynamic Optimization. These types of problems are widely regarded as difficult to compute, however we show empirically and through informal reasoning that it is in fact possible to achieve good results within our application by using a few approximations. On a more technical level, when computing the scenario tree of dynamic programming we are able to reject some branches where the 'probability of improvement is small' by using a 'branch-and-bound' rule. Based on statistical analysis of the predictor models we also conclude that only a few steps of exploration and one step of recourse is needed in the stochastic optimization formulation, making the problem much more computationally tractable. As will be stressed in the thesis, the problem of selecting banners on websites is similar to a class of problems known as 'Bandit Problems'. In bandit problems the objective is to sequentially pull levers on hypothetical 'slot-machines', where each lever generates a reward from a probabilistic distribution. To the authors' knowledge there has been no attempt at solving the full finite horizon planning problem when the reward distributions are dynamic (changing over time). We contribute to the existing research by developing an approximate solution to this problem.

This thesis covers a broad number of topics, therefore we have provided a short summary of each chapter in the following section. Hopefully this will ease navigation through the text. Consulting this overview now and again can aid a reader who does not wish to take part in all mathematical details. Section 1.2.2 is required reading and lays out the foundations of the problem considered. Chapter 2 and 3 can be read independently, these chapters are more theoretical, hence some readers might find it natural to start reading chapter 4 instead and then

consult chapter 2 and 3 as needed. But other readers may find these chapters necessary to put the problem treated in chapter 4 in the right context.

1.1 Topics covered and short summaries

There are ten chapters (including this introduction) and two appendices. Main topics dealt with are:

Chapter 1: This chapter contains this introduction and some background on the e-commerce theory as well as the important definition of CPI.

Chapter 2: A brief introduction to optimization is provided with emphasis on global optimization techniques. After some preliminary definitions, the principles of optimization algorithms suited for unknown or 'black-box' functions are outlined. In particular we consider the EGO-algorithm which is based on optimization with respect to a response surface, a 'surrogate' model for an unknown function. We also provide some technical details on this algorithm which are included primarily for the interested reader. Also see the bibliography for some influential papers on this subject. This chapter can be read independently. The important point of this chapter is to convey the difficulties involved when trying to model and optimize unknown functions.

Chapter 3: This is a stand-alone chapter which provides a survey of some function modeling techniques. When modeling real-world phenomena it is important to be able to characterize the type of function one is dealing with and to distinguish between different approaches. The aim of this chapter is to provide a division of models into categories such as deterministic, stochastic, local and global and to help tell them apart. A useful read is section 3.3.4 on time series which may aid the understanding of Chapter 7.

Chapter 4: In this chapter we model the weight selection problem as the optimization of a multivariate function, drawing upon the ideas of chapter 2 and 3. At the end of this chapter, in section 4.2 we discuss various problems inherent in such a description. Generally we call this the 'one-step' approach to distinguish it from the planning algorithm which builds on a different concept.

Chapter 5: Provides a discussion of the more general trade-off between one step optimization and data acquisition. In section 5.2 the ε-type algorithms are described, these are important because they provide a straightforward and concrete way of setting weights. In chapter 8 we test the performance of such an algorithm and compare it to the one we have constructed.

Chapter 6: This is a central chapter where we introduce the planning algorithm, which is the focus of most of this thesis. We describe 'planning' problems and relate these to 'bandit'-problems and provide some theoretical background and a literature overview of related research.

Chapter 7: This chapter is the beginning of the formalized treatment of our algorithm. The chapter lays out the statistical model (the random walk) which we use throughout the rest of the thesis. A discussion of sampling,

errors and variances is also provided. Section 7.4 is an extension of the basic model which incorporates uncertainty in the measurements of the mean value (CPI) of the random walk model. This section requires some familiarity with time series analysis or stochastic processes. The extension is interesting in its own right but can be skipped without loss of continuity.

Chapter 8: This chapter is the heart of the thesis and is in short a formalization of dynamic programming with uncertainty, a form of discrete dynamic stochastic programming. The modeling language is the Markov Decision Process which is a general framework used to describe different planning problems. The approach is general with regard to the probabilistic model incorporated. Although notation may be confusing at first, we have tried to be as exact as possible. The reader familiar with the theory will most certainly recognize the well-known Bellman Optimality Principle of section 8.1. Also the examples of section 8.2 should be helpful.

Chapter 9: This chapter outlines the MATLAB implementation of the algorithm and provides results from simulations with graphs. This chapter applies theory and illuminates many important issues. We also show the superiority of our algorithm compared to the algorithms of Chapter 5.

Chapter 10: A concluding chapter which provides a more general discussion of the simulation results and our proposed solution method. We also elaborate on some extensions of this work suitable for future research.

Appendix A: Gives an overview of different least squares problems.

Appendix B: Parenthetical on the data-aggregation issue.

1.2 Background

1.2.1 TradeDoubler

[4] TradeDoubler is a European provider of online marketing and sales solutions. Industry leaders all over Europe have partnered with TradeDoubler to work with performance-based marketing and to manage online relationships in order to increase sales and improve online business.

General facts

- Founded in 1999
- Operations in 17 European markets
- 256 employees throughout Europe
- Head quarters in Stockholm
- Local offices throughout Europe

Business facts

- Offers online marketing and sales solutions
- Solutions to B2B and B2C companies
- Performance-based marketing model
- Proprietary technology platform

Financial facts

- 117 million Euros in turnover 2005
- 82% increase in turnover from 2004 to 2005
- Break even reached in Q3 2002 on a group level
- Major share holders are TowerBrook Capital Partners (formerly Soros Private Equity Partner), Arctic Ventures and Prosper Capital

Company vision

> To be the preferred partner driving profitability through online marketing

Company mission

> TradeDoubler creates profitable partnerships with major advertisers, publishers and agencies based on its proprietary technology and comprehensive knowledge within online marketing.

[4] Source: http://www.tradedoubler.com

1.2.2 The Application

Definitions

A frequent form of advertising is showing 'banners' or ads on web pages. We will refer to the creators of such advertisements as 'advertisers'. A display of an ad on a website is called an 'impression'. The advertisements are in the form of graphical images promoting a product or service offered by the advertiser. TradeDoubler helps the advertiser to put these images on web pages. The owners of these web pages are called 'affiliates'. TradeDoubler thus works as an intermediary between the affiliate and the advertiser. Through this process value is generated for all three parties.
A special type of advertisement occurs when ads are put together in the form of a 'pool'. A pool is a single object that can be put on a web page which contains several banners or ad-elements, from on or several advertisers. A pool can be thought of as a ready made package which contains several (allegedly) interesting ad-elements. When a pool is shown on a web page, one of the ad-elements in the pool is shown. Which element to be selected is the decision of interest in this thesis.
The technical mechanism behind the pool is a set of frequencies or weights determining the probability of each advert being shown. These weights together with the banner elements in the pool is what we call an *ad-rotation*.
The reason ads are shown is two-fold. To get people to click on them, and to induce revenue generating actions following as a direct result of these clicks[5]. Revenue generating actions are

- Clicks (The visitor clicks on a banner that is shown.)
- Leads (The visitor clicks on a banner and then signs up for some service on the advertisers web site.)
- Sales (After clicking the banner advertisement the visitor subsequently buys some product from the advertisers web site.)

Problem: How should weights in pools be set?

Our task is to find a system which assigns optimal weights. Optimal weights are such that the revenue of the pool is maximized. The pool should consequently show those elements which result in those revenue generating actions with the highest payoff.
It is in practice difficult to keep track of which particular banner impressions lead to which revenue generating events such as a 'lead' or 'sale'. However, preceding every event there is a click and in the absence of other information every *click* is considered to have an equal probability of generating a *lead or a sale*. This suggests that we measure revenue in terms of *number of clicks*. Thus clicks will be the relevant determinant of whether an element is considered good or not, relying on the assumption that clicks cause the other types of events. As an example, consider a web site visitor clicking on an element which *leads* her to a page where she fills out a form, applying for

[5]Of course there is inherent value in exposing the visitor to the product regardless of whether the visitor clicks on the image or not but we are only interested in effects we can observe.

membership for a particular service. If the application is successfully completed revenue is generated. It is worth pointing out that after a person clicks on an element, it is up to the advertiser to ensure that the content the visitor is presented with is sufficiently interesting to induce a lead or a sale (the visitor in our example was induced to apply for a membership service). Our task can be summarized

> Maximize the number of clicks generated from all banners shown in a pool on a given web site during a certain time period.

The whole process we have discussed here provides value for three parties. These parties are the advertiser, the affiliate and TradeDoubler. The transactions involved in generating 'pool revenue' generates value for each individual part. It is clear that it is highly desirable for all parties that the pools being shown on the web pages are fully optimized.

CPI

An important measure in e-commerce is *Clicks Per Impression* (CPI). The CPI is calculated by dividing the number of clicks on an element by the number times it was shown during a time interval, or in other words the number of impressions, say N on this banner in this time interval. Consequently if we let

$$c_i = \begin{cases} 1, & \text{if impression } i \text{ produced a click;} \\ 0, & \text{otherwise.} \end{cases}$$

then we can define our performance measure as

$$\text{CPI} = \frac{1}{N} \sum_{1}^{N} c_i$$

Chapter 2

Global Optimization

In this chapter we will present some theory on global optimization. The theoretical underpinnings of this introduction can be found in [24] and [22].

2.1 Introduction to Global Optimization

We consider a real valued n-dimensional function $f(x)$. The problem of finding the maximum of this function subject to a constraint $g(x) = 0$ can be written

$$\max_{x \in \mathbb{R}^n} \; f(x) \tag{2.1.1}$$

$$\text{subj to} \quad g(x) = 0 \tag{2.1.2}$$

This is a *constrained* global optimization problem. In general a constrained optimization problem may have many different inequality and equality constraints. Without any constraints we say that we have a (global) *unconstrained* optimization problem. The constraint defines the set of *feasible* solutions or what is called the feasible region. $f(x)$ is usually called the *objective function*. The problem consists of finding a feasible solution, that is a solution x which satisfies $g(x) = 0$ and yields the largest value of $f(x)$. This is called an optimal solution. In general optimal solutions are not unique, in the sense that there may be several equally good solutions.
General optimization problems are very seldom solved analytically but instead solved by *optimization algorithms*. They usually proceed with some initial guess of the optimal solution and then iteratively search the feasible region for directions where the function value improves. Algorithms most commonly use the information contained in the constraint g, the expression of f and the first and second order derivatives of f.

2.1.1 Local vs Global Optimization

To find a solution to (2.1.1) is in general difficult. Most algorithms find a series of *locally* optimal solutions instead. A local maximum is a point z such that $f(z)$ is greater than $f(x)$ for all points x in a vicinity of z. Local maximums may or may not be equal to the global maximum depending on the situation. In economical and financial applications such as ours we are primarily concerned with finding global maximums.

2.1.2 Gradient Based Optimization Algorithms

The *gradient* of a multivariate function f is the vector of the first order partial derivatives of f. This is referred to as *first order information*. It may also be known that the *Hessian* for a function f is the matrix of all second order partial derivatives[1]. This is referred to as *second order information*. This information together can be used to form a local second order approximation of f (usually called the multivariate Taylor Expansion). Many optimization algorithms use this information to search for local optimums. For a maximization problem the algorithm looks for *directions of ascent* provided by the first order information and then uses the second order information to assert the *curvature* of the function and thus decide if it may be a local optimum. These algorithms trace out a trajectory in the feasible set with the aim of locating a local maximum of f.

2.1.3 Stochastic Optimization

Sometimes some components of (2.1.1) may be uncertain. For example suppose f is some form of a production function, and the constraint represents the amount of available resources for production. We may know by certainty exactly how resources are transformed to outputs through the function f but we may be uncertain how much available resources we have. Then we will have to base our optimization on an estimate of the resources available. In situations where randomness occurs we might have to use the probabilistic concept of an *expected* model. The optimization is then performed on the expected resources available. In some cases it is permissible to disregard the probabilistic nature of the application and treat the problem as deterministic.

2.2 Response Surfaces in Global Optimization

A constrained optimization problem of particular interest in this thesis is

$$\begin{aligned} &\max_{x\in\mathbb{R}^n} \; f(x_1, x_2, \ldots, x_n) \\ &\sum x_i = 1 \\ &x_i \geq 0 \quad i = 1, \ldots, n \end{aligned}$$

2.2.1 Lack of Analytical Expression

Sometimes we have no information on $f(x)$ other than samples of function values. In this case the optimization problem takes the form of what is called a 'black-box' optimization problem. These problems arise when there is no analytical expression nor any gradient information available. These functions are referred to as *black-box functions*. It is still possible to model such functions by constructing a *surrogate model* or *meta-model*. A simple example of a meta-model in 2 dimensions is a simple regression line fitted to points scattered in a 2d-graph. This surrogate, i.e the regression line can be used to predict the response y of x. The purpose of the regression line is to provide a

[1] That is the matrix of all $\frac{\partial^2 f}{\partial x_i x_j}$ for all i, j

model of the functional relationship between y and x. This model can be used to predict function values at an arbitrary point in the domain where the model is valid. The quality of the prediction is determined by the model uncertainty. In several dimensions we call such a model a *Response Surface Model*. This is simply a surface fitted to observed input-output pairs of observations, usually together with some statistical measure of the uncertainty of the approximation. Response Surfaces are used to model the output of systems in a domain where we have limited or no observability of the function. A regression line for example can be used to make predictions into the future. Response surfaces are practical to model functions which are considered 'expensive' to evaluate. The expense may arise in computer simulations where every function evaluation takes very long time. More importantly for our purposes, in real-life economical applications there may be a significant cost attached to evaluating the function. This is true in particular for optimization applications where we want to minimize the number of function evaluations at non-optimal input parameters.
Obviously, there are benefits from evaluating the function through a surrogate instead of computing the real (costly) function. More details on function models are provided in the next chapter. Right now we are interested in a particular type of optimization which arises when we use a surrogate model to find the optimum. We are going to present one such method here. The idea behind this method is to balance optimization on the surrogate with the need to improve the surrogate in order to reduce the uncertainty of this approximation [25].

2.2.2 Efficient Global Optimization

The principle behind Efficient Global Optimization (EGO) is to first fit a surface to data (input-output pairs). This is the model for $f(x)$. The model provides means of predicting function values at unknown data points. The EGO algorithm is then applied to this model and provides a mechanism for updating and improving the surface model as determined by the information summarized in the *Expected Improvement (EI)* function. This function will be explained in more detail shortly. The EI function serves as the objective function in the EGO optimization algorithm. The benefit of EGO is that it weighs together the uncertainty of the model we use to represent f together with the potential of improving the function value at a given point. EGO is labeled a global optimization algorithm and is specifically designed for situations where the number of function evaluations is severely limited by time or cost. It allows us to 'jump to conclusions' instead of following a trajectory like the gradient based optimization algorithms [15], [25].

> The key to using response surfaces for global optimization lies in balancing the need to exploit the approximating surface (by sampling where it is minimized) with the need to improve the approximation (by sampling where prediction error may be high).[15]

Any model can in principle be used to represent f. The only requirement for it to be used with EGO is that it provides predictions of f at any point together with some measure of uncertainty of the prediction. The predictor uncertainty

is the *model uncertainty*, it is *not* measurement error or noise. One regards samples of $f(x)$ at a point x as the true response of f at x. Thus EGO in its original form supports only deterministic function models that *interpolate* the data (See chapter 3). This implies that the uncertainty at a point which has previously been sampled is zero. The result of this is that the EGO algorithm never performs repeated samples at a point. In a 2005 paper by D.Hung et al [1] a modified Expected Improvement is suggested which incorporates measurement error or noise in the response values. Thus in their context it may be desirable to make repeated samples at a point in order to reduce the measurement uncertainty at this point. However, this modification still assumes that the underlying function is fixed. Thus there is measurement error in the samples of the underlying fixed model in this approach. For an excellent overview and comparison of several different global optimization algorithms that use response surfaces see [16].

The Expected Improvement Function

The idea behind EGO is probably best understood by studying the Expected Improvement Function. This function balances local and global search. Searching the prediction error function for the point where the uncertainty is highest amounts to global search of the parameter space, whereas searching the predictor amounts to local search. We first define the Improvement Function and then take the expected value to obtain the Expected Improvement, the definitions follow [25] chapter 6 and [15]. To make the exposition more standard it is henceforth assumed that the aim is to find a global minimum.[2]

Definition 2.2.1 *Let $Y(\mathbf{x})$ model our uncertainty about the response $y(\mathbf{x})$ at the point $\mathbf{x}$. Let $y_{min} = min\{y^{(1)}, y^{(2)}, \ldots, y^{(N)}\}$ denote the best function value from the sample obtained so far, then the Improvement is defined as*

$$I(\mathbf{x}) = \max\{y_{min} - Y(\mathbf{x}), 0\}$$

Since y_{min} is the best (smallest) value found sofar, a large observation of the difference, $y_{\min} - y(\mathbf{x})$ entails that $\mathbf{x}$ is such that *the predicted value* $y(\mathbf{x})$ is smaller (thus better) than y_{min}. Making these kind of arguments, what we are really interested in is the expectation of $I(x)$. Here $Y(x)$ is the response model predictor, $Y(x) \sim \mathcal{N}(\hat{y}, s^2)$ where $\hat{y}$ is the predicted value at x and s is the standard error at x. [3]

Definition 2.2.2 *The Expected Improvement is*

$$E[I(\mathbf{x})] = \int I(\mathbf{x})\, dF(\mathbf{x})$$

where F is the distribution of $I(\mathbf{x})$.

The derivation of the expectation of $I(x)$ is quite complex and we state it as a theorem without proof

[2] Note that the problem $\max_x f(x)$ is equivalent to $\min_x -f(x)$.

[3] A response model predictor yields two outputs. The prediction made by the surrogate model $\hat{y}$ of the unknown function f at a point x, as well as a measure of the uncertainty of the prediction made at this point s .

Theorem 2.2.3

$$E(I) = \begin{cases} (y_{min} - \hat{y})\Psi\left(\frac{y_{min}-\hat{y}}{s}\right) + s\psi\left(\frac{y_{min}-\hat{y}}{s}\right), & s > 0 \\ 0, & s = 0 \end{cases}$$

where Ψ is the standard normal distribution function and ψ is the standard normal density function[4]. This function gives the 'value' of sampling at a point. $s = 0$ corresponds to a point already sampled (so there is no uncertainty about the value at this point) and hence our surface interpolates the data at this point. This implies in EGO that there is no value in sampling at such a point thus the expected improvement is zero.

The EGO Optimization Algorithm

We will here give a very brief description of how EGO works, for details refer to [15]. The principle behind the algorithm is to iteratively sample the function $f(x)$ creating a surrogate for f in search for a global maximum.[5] In every iteration the expected improvement function, EI, is maximized. The point where the EI function attains its highest value will become the next point to sample f. [16] states that the algorithm will converge to the global maximum but this may take very long time. In practice the algorithm is terminated when the expected improvement is sufficiently small for most values of x. A significant part of the computing time in this algorithm is devoted to actually solving the so called 'subproblems' in which the EI is maximized. Jones et al have developed a branch-and-bound type optimization algorithm for these problems.

[4]Note the distinction here,

$$\Psi(x) = \Pr\{X \leq x\}$$

where $X \sim \text{normal}(0, 1)$ and $\psi = \frac{d\Psi}{dx}$

[5]In practice before the algorithm is initiated a *space-filling design* is usually employed to get an initial model of how f looks like to reduce the standard errors

Chapter 3

Function Modeling

3.1 Overview

Function modeling or approximation has a venerable history of research in several different academic fields such as machine learning, mathematics, systems theory, signal processing, and statistics. The need for modeling function behavior from data commonly arises when one lacks a suitable analytic model of some real world phenomena and has, to some extent, consider it a black box. In practice that is more often than not the case. The goal of this chapter is to give a brief but holistic view of the problem and the different approaches used in solving it. We will then use this as a motivation for our choices of function models in the latter chapters. To ease readability we have chosen to structure the methods into distinct groups according to the main features of the approaches, rather than on a field by field basis. It is important to point out that this is not meant to be an exhaustive treatment of the subject, but rather a brief overview.

3.2 Deterministic Function Models

Consider that we want to model the response $\mathbf{y}$ from an unknown function f of $\mathbf{x}$:

$$\mathbf{y} = f(\mathbf{x}) \tag{3.2.1}$$

where $\mathbf{x}$ and $\mathbf{y}$ may be vectors. The function under consideration may thus be multivariate and vector valued. If the domain of the function is $\mathbf{X}$ and the codomain is $\mathbf{Y}$, then the permissible input is all $\mathbf{x} \in \mathbf{X}$ and the possible output is $\mathbf{y} \in \mathbf{Y}$. A natural way of providing a description of this function is to sample the function for different values of the input variable $\mathbf{x}$ and construct a table of the resulting input-output pairs $(\mathbf{x}, \mathbf{y})$. Formally this can be characterized by an ordered list of distinct variable values $\mathbf{x}_1, \ldots, \mathbf{x}_n$ and a list of matching results $\mathbf{y}_1, \ldots, \mathbf{y}_n$ such that $\mathbf{y}_k = f(\mathbf{x}_k)$. Henceforth a pair of values $(\mathbf{x}_k, \mathbf{y}_k)$ will be referred to as a data point or a sample.
Unless the domain of the function, $\mathbf{X}$, is finite, which rarely happens in practice, it is impossible to list the output for every $\mathbf{x} \in \mathbf{X}$. However it is

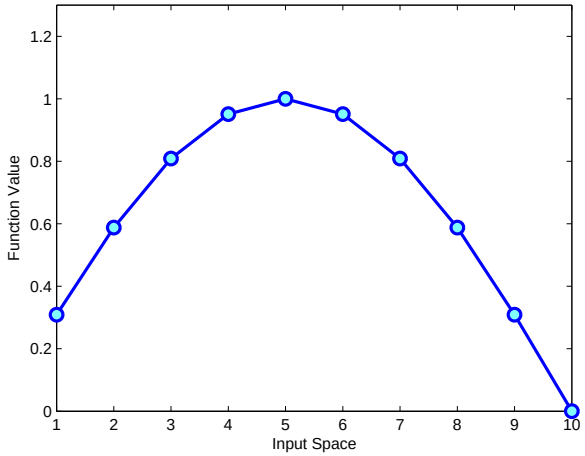

Figure 3.1: A model of an unknown univariate function based on linear interpolation between known data points.

possible to use some suitably general parameterized[1] function model and deduce the parameters of this model from a table of known values. The practice of using a model of this kind to evaluate a function within a range of known values is commonly referred to as *interpolation*. When used to evaluate the function outside a range of known values it is instead called *extrapolation*. For most practical purposes the methods are however conceptually identical. Good overviews of interpolation methods can be found in [8] and [10]. Since the function is assumed to be deterministic, the fitted model should pass exactly through all known data points, which in practice means that the number of parameters in the model tends to be close to the number of known data points. Generally, the closer the chosen model is to the target function, the fewer parameters are needed. Since the model has to be able to pass through any future data points, some strong assumptions about the shape the function are effectively made by using fewer parameters than data points. This requires considerable care and is rarely practical.
One of the oldest methods of modeling deterministic functions is to fit an nth-order polynomial to all data points. However this may result in wildly oscillating behavior between the data points for higher order polynomials, see [10] pp. 106-107 for a discussion. This insight leads to an approach where models are fitted locally to a limited number of known data points in the vicinity around each point of interest, a so called *local interpolation*. These models may be lower order polynomials or more complex constructs. The simplest case is linear interpolation where a straight line is fitted locally between the two neighboring points. An example of this can be seen in figure 3.1. If the function of interest is smooth to some extent, a more accurate model than just linear interpolation can be constructed. For smooth functions, several neighboring points can be used, creating a local but more sophisticated

[1]Since a parameterized model can vary in a mathematically precise fashion with each parameter, each parameter represents a *degree of freedom* in the model.

approximation which also approximates the curvature of the underlying function. One example of this is *spline* interpolation. These are piecewise continuous polynomial constructions, differentiable up to some specified order. Another smooth deterministic function model with a statistical interpretation is the kriging approximation model treated briefly in section 3.4. It is worth noting that since any interpolation model grows more detailed as the number of data points grows, any of these simple interpolation schemes can yield arbitrary precision of a deterministic relation by simply adding more data points.

3.3 Stochastic Function Models

In real world applications, data is seldom purely deterministic. Errors in measurement, and inaccuracies due to a multitude of uncontrolled factors impact the accuracy of sample values of real-world functional relationships. The challenge is finding a function model for the underlying relationship without incorporating such 'noise' in the model. By analogy of the deterministic formulation in (3.2.1), we can just add a stochastic error term, a random variable $\varepsilon_{\mathbf{x}}$ with zero mean and finite variance. It may be dependent on the input, but for simplicity we will in this overview only consider the case where the errors are i.i.d, that is, independent of the input and identically distributed.

$$\mathbf{y} = f(\mathbf{x}) + \varepsilon \tag{3.3.1}$$

With statistical terminology this is called a *homoscedastic* error, as opposed to the *heteroscedastic* case with dependency in the errors.
There are primarily two types of strategies available for keeping noise out of the model. Either we can express a preference for simple models by using a global function model with a limited number of parameters, or we can employ some kind of local smoothing model. In the end these approaches are largely equivalent and can be seen as a smoothing of the data to avoid incorporating what is assumed to be noise into the model. For an interesting comparison of parameterized models and smoothing see section 2.9 of [11]. When selecting the number of parameters in global models, or adjusting smoothing parameters of local models one is in fact making explicit or implicit assumptions about the level of noise in the data. This can be done either by prior knowledge or empirically, which will be briefly discussed in the section on model validation. The model in equation (3.3.1) represents a relaxation of the requirement that the model should perfectly fit all data points. When taking noise into consideration, the estimation of the model parameters has to be done by minimizing an 'error measure'. This allows for a broader class of models than just the interpolating models, and goes under several different names depending on in which field the model is applied. Curve fitting, machine learning, regression analysis and system identification are some of them.

3.3.1 Global Models

The simplest single response parametric model is a *linear* combination of the input variables

$$y(\mathbf{x}) = \boldsymbol{\beta}^T\mathbf{x} + \varepsilon = \beta_1 x_1 + \cdots + \beta_p x_p + \varepsilon \tag{3.3.2}$$

where $\boldsymbol{\beta}$ is a vector of parameters, $\mathbf{x}$ is the vector of input variables and y is the response value. This linear model is the essence of the statistical method of linear regression[2]. Linear regression traditionally finds an estimate of β in the 'least squares sense' by minimizing the sum of squared errors. There are many efficient algorithms available for solving least squares problems which perhaps is the strongest reason for the popularity of the linear model. The solution of the general least squares problem is given in appendix A.
The ε term in (3.3.2) is the random quantity called the model error. In practice ε represents the residue of the response variable after the fit of the linear combination of input variables has been accounted for. The residuals are used to statistically estimate the variance of the model error. The variance of the model error can then be compared to the variance of the response variable to get an expression for how large proportion of the variation of the response variable that the model can account for. This measure is often reported by statistical packages, and is called the (sample) multiple correlation coefficient (R). Assuming that the model is correct, the variance of the model error should represent the variance of the noise, the true sample error in (3.3.1).
The linear model (3.3.2) is called linear because it is *linear in the parameters* i.e the components of $\boldsymbol{\beta}$. The following model is also a linear model, but the input variable $\mathbf{x}$ enters the model through the p basis functions $f_1, f_2, \ldots, f_p$

$$y(\mathbf{x}) = \beta_1 f_1(\mathbf{x}) + \cdots + \beta_p f_p(\mathbf{x}) + \varepsilon \tag{3.3.3}$$

where the basis for example could consist of polynomials, such as when each f_i represents a power of i, yielding a polynomial representation of order p. An example of such a global polynomial model is displayed in figure 3.2. The rationale for a polynomial model can perhaps best be illustrated by an analogy to the Taylor series expansion. The famous theorem in Mathematics known as Taylor's theorem states that a deterministic analytic multivariate function satisfying certain differentiability criteria can be represented by a convergent series of polynomial terms. Linear regression with polynomial terms is just one example of modeling functions by linear expansions of basis functions. Admittedly, a polynomial basis is perhaps the most common, but it might not always be suitable for computational reasons, or it might require too many basis functions which is problematic when one often face restrictions on the number of parameters that can be employed within the model. Other examples of basis functions for f_i besides polynomials are fourier expansions and wavelets where the latter is more suitable for functions that do not have a natural interpretation in the frequency domain. There exists a vast diversity of basis functions that are used in practice. Common to all these basis function approaches is that in view of model (3.3.3), the series should converge in the presence of the error term (noise). This entails a further formidable discussion which we leave out here. This is an interesting subject however, but a thorough treatment of these methods is beyond the scope of this thesis, see for example chapter 5 of [11].

[2] see section 7.7 of [13] for an in-depth treatment

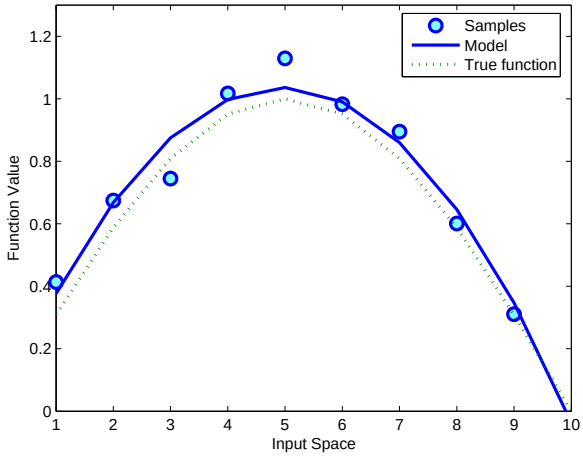

Figure 3.2: A simple second degree polynomial function model of an unknown univariate function (the same as in the previous figure) based on samples with measurement error. The model does not pass through the data points to avoid the noise, still as can be seen from the match with the dotted line, the errors cause a slight misalignment with the true function.

Non-Linear Models

Non-linear models understandably encompass a much broader set of function models than their linear counterparts. There is no limit to the number of possible non-linear constructs to use in a model. This means that in practice you choose a suitable subset, or family, of non-linear parameterized models. These are then fitted to data by selecting the parameters that minimize some error function. As in the linear case the most common practice is by minimizing the squared error of the model fit. Since non-linear models are not as restricted as linear models they can yield much better predictive power, but since there are more parameters to estimate, these models are more vulnerable to over-fitting, brought up in the model validation section which follows. The non-linearity also means that minimizing the error rarely reduces to just solving a simple linear system. Usually one has to employ some kind of gradient based hill climbing optimization to estimate the model parameters. These methods vary a lot in time and space complexity.

3.3.2 Local Models

There are, as mentioned in the introduction to stochastic function modeling, primarily two approaches to dealing with the noise. The first one relies on fitting a global model restricted to a limited number of parameters[3], which was treated above. The second approach described in this section uses all known samples as parameters but avoids the noise by utilizing some kind of

[3]Remember that the number of parameters decide the degrees of freedom in the model.

local smoothing, or averaging of samples. This can be formalized as

$$f(\mathbf{x}) = \sum_{i=1}^{n} g(\mathbf{y}_i, d(\mathbf{x}, \mathbf{x}_i)) + \varepsilon$$

where $(\mathbf{x}_i, \mathbf{y}_i)$ are the known samples, $d(\mathbf{x}, \mathbf{x}_i)$ is a distance metric and g is a weighted averaging function. In short, the value of f near a point $\mathbf{x}$ will be a weighted average of the sample data points that are 'close' according to some metric. Local models are conceptually very simple and similar to the local interpolation models treated earlier. This makes them attractive, but it also means that they use all the data available in the modeling process similar to the way interpolation models do. A classic example is the nearest neighbor method that for any point use a function of the k nearest samples to construct the response of the function, basically taking a local average. This model also offers an intuitive interpretation of the averaging; relating the degree of smoothing needed, to the size of the noise present in the observed samples. For a treatment of these methods, see chapter 6 of [11].
Generally, local methods and non-linear global methods are employed when models with good predictive power are needed for complex relationships, while linear global alternatives are preferred when a certain simple analytic relationship is believed to be present, or when a simple model is otherwise preferred. Local models also have the advantage that if samples are iteratively added the model accuracy will scale, while the accuracy of parametric models tends to be inherently limited by the initial choice of degrees of freedom.

3.3.3 Nominal Models

A special subset of function modeling algorithms arise when the response variable takes values in a discrete set $\mathbf{Y}$ (a discrete co-domain). This is the case when response data $\mathbf{y}_i$ consists of unordered categories like $\mathbf{Y} = \{orange, apple, banana\}$. With statistical terminology this type of discrete data is often called *nominal* data, contrary to *ordinal* data where order or magnitude is relevant. In nominal models, magnitudes are irrelevant, there are no assumptions of smoothness, and interpolating between categories does not make sense. This area of function modeling is usually referred to as supervised learning, classification, pattern recognition or discriminant analysis. These problems are for example studied in machine learning. An excellent resource on this topic is the book Pattern Classification by Duda, Hart and Stork [7]. A nominal model is based on the idea of measuring how well a point in the input space fits to each category. There are mainly two approaches in providing this 'matching'. The first one is statistical (such as in parametric statistics and in non-parametric density estimation), where the 'matching category' is determined by the population which maximizes a probabilistic measure as a function of the input point. The second approach is by approximating boundaries of 'decision regions' which best separate the populations or categories (a non-parametric method in statistical terminology). This defines 'surfaces' in the input space known as *discriminant functions* which make it possible to discriminate, on the basis of these decision regions, which category a point is most likely to belong[4].

[4]For details see section 5.2.2 of [7].

Nominal models represent a mapping of input variables to a response, that is, different ways to model a functional relationship. Although nominal models differ in one aspect. In a nominal model, output is assigned by the classifier f, which selects the category corresponding to the discriminant function g producing the highest value as a function of the input point. Further, the discriminant functions are by themselves also function models. The task of each discriminant function is to describe a category by providing a measure on how likely it is that a given input x belongs to this category. The model can be illustrated as

$$f(\mathbf{x}) = \arg\max_{\mathbf{y}\in\mathbf{Y}}\{g_{\mathbf{y}}(\mathbf{x}) + \varepsilon\} \tag{3.3.4}$$

for points $\mathbf{x}$ in the input space $\mathbf{X}$. The functions $g_{\mathbf{y}}$ are either likelihood functions (probabilistic measures) or discriminant functions, determining the 'likeliness' of belonging to the population or category $\mathbf{y}$ respectively[5].
When using linear discriminant functions the input space is said to be divided into categories by *cutting hyperplanes*. But discriminant functions may also be non-linear (such as in classical artificial neural networks[6]) or linear only when the input space first is transformed into a higher dimensional input space[7] (as is the case of general linear machines and support vector machines). 'Support vector machines' in particular has enjoyed an increased interest in the last couple of years[8].
A very simple and widely used (local) method for classification is the k-Nearest Neighbor method. This is a nominal model, but is really more of a heuristical method, since it does not explicitly construct discriminant functions such as in (3.3.4). Instead, it attempts to model f directly by assigning points in the input space to categories based on the proximity of the points to each other.
Finally there are also nominal models where the domain of f is also a set of nominal data (i.e both the domain and co-domain are discrete). These models include popular data-mining methods such as decision trees and association rules for classification[9].

3.3.4 Time Series Models

Temporal Models

Temporal models are function models where the relation between the response variable and the inputs is described by an expansion of a sequence of 'lagged variables' in time. The purpose of such models is to describe relationships in time as well as predict future response values from past response values. A linear temporal model of order n can be written

$$Y(t) = \beta_1 \cdot Y(t-1) + ... + \beta_n \cdot Y(t-n) + \varepsilon_t \tag{3.3.5}$$

[5] In some cases the likelihood approach and the discriminant function coincide as in Fisher's Linear Discriminant Function and the maximum likelihood of two normally distributed populations with equal variance (for a discussion see section 11.5 of [13]).

[6] Neural networks are not inherently nominal models. In the common formulation they give vector valued output $(y_1, \ldots, y_n)$ where each component is constructed to model the discriminant function for a specific category.

[7] Such as when adding the variables x^2 and x^3 into the simple linear regression model $y = \beta x + \varepsilon$, where x^2 and x^3 are just functional variations of the input x.

[8] For a thorough treatment of support vector machines see [6] [23].

[9] See chapter 8 of [7] and [18] respectively

where the same variable Y_t is studied at different points in time indexed by t. In some sense Y_t is both the input and the response of this model. This linear relationship is a special case of the linear regression model explained previously. For our purposes it is interesting to note that when this model is applied iteratively to make predictions, the errors will build up in an additive fashion, which decreases the prediction accuracy.

Stochastic Process Models

The statistical field of time series analysis has the most elaborate theoretic framework for working with temporal models. Since our problem is inherently temporal we will give a couple of definitions from time series analysis that will be used in chapter 7. All definitions will follow [4], one of the most widely recognized books on the subject. Although these definitions are crucial for section 7.4, this section is not necessary for understanding the rest of the thesis and may be skipped at first and just used for reference as needed. Temporal models studied in time series analysis are linear stochastic processes defined as linear combinations of 'white noise'. A sequence of random variables are said to be white noise if the terms have zero mean and are uncorrelated. When the terms are normally distributed we say that we have *Gaussian White Noise*. For example we write $\{Z_t\} \sim \text{WN}(0, \sigma^2)$ to express that the sequence $\{Z_t\}$ is normally distributed white noise, which is equivalent to stating that $Z_t \sim \mathcal{N}(0, \sigma^2)$ and $\mathbf{E}(Z_i Z_j) = 0$ for all $i \neq j$. We proceed by defining the *linear process*

$$X_t = \sum_{j=-\infty}^{\infty} \psi_j Z_{t-j} \tag{3.3.6}$$

where $\{Z_t\} \sim \text{WN}(0, \sigma^2)$ and $\{\psi_j\}$ is a sequence of constants such that

$$\sum_{j=-\infty}^{\infty} |\psi_j| < \infty$$

The coefficients ψ_j constitute a *linear filter* applied to the white noise sequence $\{Z_t\}$. When the process is observed, and the observed values are plotted in a graph we obtain the *sample path* of the stochastic process. Linear processes of this sort are *stationary*[10] in time, stationarity can be intuitively understood as a process which neither will increase nor decrease indefinitely. While the sample paths of stationary processes may lean in either a positive or negative direction for some time, the sample path tend to *regress* back towards the mean. This implies that the mean and covariance of the terms is invariant with respect to time (that is the covariance does not depend on t) and it is possible to formulate the *autocovariance*

$$\gamma(t, t+h) = \text{Cov}(X_t, X_{t+h})$$

as

$$\gamma(t, t+h) = \gamma(h)$$

[10] Technically the condition we use is *weak* stationarity, these terms are often used interchangeably in the literature.

that is, as function of the lag h alone. It is possible to prove that any process is stationary by formulating the covariance function and showing that this expression does not depend on t. Time-invariance is an attractive property of stationary series, simplifying analysis and makes it possible to prove many useful theorems in the theory of random processes.
Equation (3.3.6) is the *(linear) filter representation* of time series X_t. With this definition we can study a useful class of linear processes known as the ARMA(p, q) process. An ARMA(p, q) process is defined as

$$X_t - \phi_1 X_{t-1} - \ldots - \phi_p X_{t-p} = Z_t + \theta_1 Z_{t-1} + \ldots + \theta_q Z_{t-q} \tag{3.3.7}$$

or more succinctly as

$$X_t \phi(B) = Z_t \theta(B) \tag{3.3.8}$$

where $\{Z_t\} \sim$WN$(0, \sigma^2)$. Here $\phi(B)$ and $\theta(B)$ are *filter polynomials*. The ARMA representation is only valid if these polynomials do not have any common zeroes. A filter polynomial is just a polynomial of the backshift operator B defined as

$$BX_t = X_{t-1}.$$

The operator can be applied several times, for example

$$BBX_t = B^2 X_t = X_{t-2}$$

Thus a polynomial in B , multiplied with X_t is a linear expansion of different lagged versions of X_t. For example if $\phi(B) = 1 + B^2$ then

$$\phi(B)X_t = (1 + B^2)X_t = X_t + X_{t-2}$$

The ARMA process in equation (3.3.7) is actually a composition of an AR (autoregressive) and an MA (moving average) process where $\phi(B)$ and $\theta(B)$ are denoted the AR and MA filter polynomials respectively. The MA and AR processes can be directly derived as special cases of the ARMA process (3.3.7), where AR(p)=ARMA$(p,0)$ and MA(q)=ARMA$(0,q)$. For example, this implies a characterization of the initial linear model (3.3.5) as an ARMA$(n, 0)$ or AR(n) given that Y_t has a filter representation such as (3.3.6), or in other words if the series Y_t is stationary.
An ARMA(p,q) process X_t is said to be *invertible* if there exists constants π_j such that $\sum_{j=0}^{\infty} |\pi_{t-j}| < \infty$ and

$$Z_t = \sum_{j=0}^{\infty} \pi_j X_{t-j} \text{ for all } t. \tag{3.3.9}$$

This requirement is equivalent to the condition

$$\theta(B) = 1 + \theta_1 B + \ldots + \theta_q B^q \neq 0 \text{ for all } |z| \leq 1.$$

because then the representation (3.3.9) can be derived by dividing equation (3.3.8) with $\theta(B)$ yielding

$$Z_t = \frac{\phi(B)}{\theta(B)} X_t \tag{3.3.10}$$

where $\frac{\phi(B)}{\theta(B)}$ is a polynomial in B. That is, an ARMA process is invertible if there exist a linear combination of (lagged) process variables $\{X_t\}$ which can be expressed as a white noise process.

3.3.5 Model Selection and Validation

The fit of a model to known data points can be made arbitrary close in many parametric models by having a model with at least as many parameters as data points, in effect reducing it to an interpolation problem. The stochastic global polynomial models reducing to global polynomial interpolation is an example of this. In a local smoothing model the equivalent is to let the smoothing parameter approach zero. On the other hand, in both cases this means that the expected noise will end up in the model too and the model will not generalize well to new data and have limited use in future prediction. The selection of the number of parameters or the selection of smoothing factor is in this sense an implicit assumption regarding the level of noise in the samples. When the level of noise is underestimated one speaks of *over-fitting* or *overtraining*. Training is simply another word for fitting a model to data, but when there is overtraining the model is 'trained' to mimic the data too closely which is not desirable since this includes the noise into the model. A solution to this problem could be to exploit prior knowledge of the modeled process, or to make simplifying assumptions about the population as well as performing empirical tests to gain a deeper understanding of how noise arises in the samples. There are various measures of over-fitting, some completely heuristic inspired by Occam's razor[11] and some more statistical such as the Akaike information criterion (AIC) and the Bayesian information criterion (BIC). The by far most popular way of empirically testing for over-fitting is cross-validation. In cross-validation, data is divided into two groups where the first group is used for model fitting and the second group is used for model validation. After the model has been fitted, the model is tested on the second group of data, providing an indication of how the model would perform on 'real-world' data. It is in this way possible to implicitly estimate the level of noise in the samples by finding the model that generalizes best to the validation data set (and to new real-world data). Dividing the data set into two groups like this, where only one group is used for model fitting, does mean that there will be less data available for estimation of model parameters. It is therefore not always possible to cross-validate, because the sample size of the data material may not allow it. This emphasizes another important trade-off, that between parameter estimation and validation.

Although there has been an immense interest in function modeling for decades, there is really no consensus on which methods yield the 'best' results. Some models are simply more suited to certain tasks than others and this is sometimes called the *inductive bias* of the model. It is indeed quite natural that it is possible to achieve better performance when making assumptions regarding the structure of the underlying function. If the assumptions of such a model are wrong, performance will likewise suffer. Demonstration of this can be found in the literature, specifically illustrated by the various 'No Free Lunch Theorems' in the fields of learning, search and optimization[12]. Besides model accuracy there are however other concerns, such as time and space complexity. The local methods considered have no initial computational cost

[11] Occam's razor was commonly expressed in Latin as *"Entia non sunt multiplicanda praeter necessitatem"* which can be paraphrased as *"Simple solutions are preferred"*

[12] See section 9.2 of [7] for an overview and [33] for a treatment in the context of black-box optimization.

but every function evaluation has a computational complexity of $\mathcal{O}(N)$, since local models need to keep track of potentially all samples N the space complexity is also $\mathcal{O}(N)$. Linear regression models while far more restricted, can initially be computationally costly having a complexity of $\mathcal{O}(N \cdot p^2)$ which stems from solving a linear system with p parameters using a QR factorization. On the other hand it only has a time and space complexity of $\mathcal{O}(p)$ when the model is evaluated (p is the number of parameters). Needless to say, a parametric model with a small number of parameters can be very manageable *after* the model has been fitted.

3.4 The DACE function model and EGO

DACE stands for Design and Analysis of Computer Experiments. This is a response surface model or surrogate which can be used with the EGO algorithm described in section 2.2.2. When fitting a DACE model, the observations themselves are not assumed to be subject to error as the model *interpolates* the data-points, it is instead the response values at *untried* sites (i.e sites where we have no observations) which the model error describes. DACE consequently makes an assumption that the underlying function it tries to model is deterministic in the respect that running the system for the same input parameters (weights) several times will yield the same output.

> Regression is about estimating regression coefficients that (together with the assumed functional form) completely describe *what the function is*. DACE is about estimating correlation parameters that describe *how the function typically behaves*.[15]

The appeal of the DACE approach lies in its way of providing a model and how it models 'on average' what behavior that can be expected when we move in the parameter space (or how a revenue function typically behaves as we move in some direction of the input space). The particular DACE implementation that we have looked at is the *kriging approximation model*. The kriging approximation is a local interpolation technique which by means of a simple model makes prediction of function values between data points statistically precise. The predictor in the model employs a spatial correlation structure and does not in its original form allow for errors in the the response values. It is possible however to build upon such deterministic models, creating stochastic models, adapted to work in environements where there is measurement error or noise, see for example [1].
The DACE modeling can be described as a two step procedure:

1. The first step is fitting an initial surrogate model. For example historical data on the system could be used for this, or an experimental design can be employed to make a more systematic initial exploration of the search space. Thus establishing a model which very roughly should depict how the function which is modeled behaves in different parts of the parameter space.

2. Secondly, the approximation is improved intelligently by sampling according to EGO. That is where the chance of improvement is high, and where the prediction error is high.

The relative significance of the first and second step depends on the application at hand.
Following Jones et al [15] it is possible to provide an intuitive explanation of how a typical kriging predictor is formed. Assume we have n observations of the form (y, x) of an unknown function f. The kriging predictor assumes that these points are generated by a stochastic process model relating different response values y to points in space x. A prediction amounts to predicting the response y^* at point x^* which has not been previously sampled. The predictor can be derived by considering the augmented sample of $n + 1$ observations where the added observation is a pseudo-observation. In other words, an additional site x^* is added to the sample with unknown response y^*. For every value of y^*, we can compute the likelihood of the augmented sample consisting of $n + 1$ observations, which is just the original set of n observations (y, x) and an extra observation (y^*, x^*) added. The likelihood is computed by using the fact that the observations are assumed to be generated by a gaussian stochastic process model, much like the gaussian process models employed in time series analysis that was covered earlier. Now it turns out that for any x^*, the value of y^* that maximizes this likelihood is the best linear predictor derived. Thus the kriging predictor is a Maximum Likelihood Estimator (MLE) based on the statistical description of the underlying stochastic function model relating points in space.
For a full treatment of the DACE model and the mathematics of the predictor see Nielsen et al [19] which also involves a toolbox implementation in MATLAB. For some numerical aspects see [20].

Chapter 4

The One-Step Approach

This chapter attempts to solve the pool optimization problem as defined in chapter 1 by drawing upon the theory of optimization and function modeling in chapters 2 and 3 respectively.

4.1 The Continuous Weight Selection Problem

In order to specify the probability (or frequency) by which each ad-element is being shown we define a distribution of probabilities called *weights* in each pool. Let $w = (w_1, \ldots, w_n)$ be the vector of weights for a pool being considered, where w_i is the probability of ad-element i being selected and n the number of elements in the pool. Formally we require that $w \in \mathcal{W}$ where $\mathcal{W}$ stands for the weight-space defined as the unit sphere in $\mathbb{R}^n$ with $\mathcal{L}_\prime$ norm. So for each $w \in \mathcal{W}$ we have $\sum w_i = 1$, $0 \leq w_i \leq 1$ and we measure distance between two points in space by

$$d(w^{(1)}, w^{(2)}) = \sum |w_i^{(1)} - w_i^{(2)}|$$

Assume a relationship between the weights w and total number of clicks R such as $R = R(w)$. The database supplies us with pairs of observations (R, w). So by changing the weight vector used on the web page we observe the resulting total number of clicks in the pool. We will initially treat our optimization problem as a traditional global black-box optimization problem. This formulation is 'one-step' in the sense that it doesn't take future decisions into account. It also means that we will treat $R(w)$ as a deterministic function which is unknown to us. By sampling from the system (database) we gather information on this functional relationship. The one step formulation will at first be very general. Our investigations of the one-step formulation will lead us to some important issues. Dealing with these issues will later materialize into something we call the 'planning' approach or simply the planning algorithm. This approach will be covered in the second part of the thesis. For now, the task is to decide which setup of weights w is best. What we want to do is get the highest CPI possible for the pool. The CPI is calculated as follows. If every evaluation of $R(w)$ uses a fixed number of impressions, $R(w)$ can then be divided by this amount to obtain CPI. Lets illustrate this with an example

Example

Let us say we start with a weight vector $w^{(0)}$ insert this into $R(w)$, that is we let $w^{(0)}$ be the banner frequencies during some period. If we define this period by say 10000 impressions[1], then element k was shown $10000 \cdot w_k$ number of times. The total pool CPI will be

$$R(w^{(0)})/10000.$$

Using this information we can then decide on a weight vector $w^{(1)}$ and calculate $R(w^{(1)})$ and so on. This results in a set of values

$$\{R(w^{(0)}), R(w^{(1)}), R(w^{(2)}), \ldots\}$$

This information can be used with some optimization algorithm with the aim of solving

$$\max_{w \in \mathcal{W}} R(w)$$

This is the one-step *continuous* weight selection optimization problem. $R(w)$ is an unknown function to us that we have sampled at some points in the weight-space. The weight selection is called continuous because the weight vector w can take on any value on the unit sphere.
A continuous algorithm is motivated when maximal CPI is believed to be achieved by determining the right 'mix' of elements and an optimal weight vector is determined by maximizing some objective function. This objective is a function of the weights alone and the purpose of the function is to capture *weight dependencies*. Weight dependencies are believed to occur on web sites with high traffic (many visitors) combined with in particular many *returning* visitors. The effects are also reinforced if the advert has a central placement on its hosting web site. As an example, by showing a single banner very intensively during a lengthier time period, it is possible that CPI will decline as a result of people getting 'tired' of it. In such cases the element has simply been shown too many times. When weight dependencies exist, $R(w)$ is assumed to be a *non-linear* function of w.

4.1.1 Evaluating R

We now show how we can use the special structure of our n-dimensional function to express the number of clicks generated during a time interval. We can write

$$R(w) = C_1(w_1) + C_2(w_2) + \cdots + C_n(w_n)$$

where C_i is number of clicks coming from element i in the pool when weight w_i is assigned to this element. We focus on one single time period so we observe our ad-rotation algorithm during a time period which results in a fixed number of impressions N. If we assume that all weights have been constant during this time period then element k was shown $m_k = N \cdot w_k$ times. Hence,

$$R_N(w) = C_1(m_1) + C_2(m_2) + \cdots + C_n(m_n)$$

[1] Time is measured in terms of impressions.

where we emphasize that the value of R is calculated using N total impressions, distributed over each ad-element. The rationale for looking at it this way is simply that this is how we evaluate $R(w)$ in practice. We know how many clicks each individual element has generated in the past. So when we want to 'evaluate' the n-dimensional R function at some weight w we use data on the number of clicks for each individual element and add them together. Now we can write

$$\begin{aligned} R^{\text{CPI}}(w) &= \frac{R_N(w)}{N} \\ &= w_1 \frac{C_1(m_1)}{m_1} + w_2 \frac{C_2(m_2)}{m_2} + \cdots + w_p \frac{C_n(m_n)}{m_n} \\ &= w_1 r_1(m_1) + w_2 r_2(m_2) + \cdots + w_n r_n(m_n) \end{aligned}$$

This is the CPI for the pool in terms of the CPI of the individual elements r_k.

4.2 Why the One-Step Approach is Unappropriate

The one-step approach has some major difficulties in providing us with a satisfying model of our system. Most importantly, it can be difficult to evaluate $R(w)$ properly at some points. This is directly related to how data is gathered on the individual ad-elements. In short, elements that have low weights are seldom shown and as the weights decrease on these elements the data diminish. This is a famous academic dilemma known as the *Exploration vs Exploitation* trade-off and is the topic of Chapter 5.
Secondly, the relationships we wish to capture change with time. We initially said that $R(w)$ was a deterministic function we wish to sample, but if $R(w)$ changes over time, that is if $R(w) = R(w,t)$, then every sample will be taken from a different function. For example if we sample during time instances $t = t_0, t_1, t_2, \ldots t_n$ we obtain a sample on $R(w,t)$ looking like

$$\{R(w^0,t_0), R(w^0,t_1), \ldots, R(w^0,t_n)\}$$

These samples will not be equal. R is not deterministic but rather *stochastic*. It may be possible to get good results by treating it as deterministic but we certainly could get a better model by dealing with the stochastic nature of the function.
The simplest form of 'randomness' is to assume that at each time t we measure the 'true' (deterministic) value of $R(w)$ at w^0 plus some 'noise'. So if $R(w^0)$ is the true value of the objective function at this point, we say that we observe $\widehat{R}(w^0,t)$ which is a *random* quantity and we write it as

$$\widehat{R}(w^0,t) = R(w^0) + \epsilon_t$$

where ε_t are random variables which are *independent and identically distributed* for all t. Specifically we assume that ε_t has *zero mean* and constant variance. It is common to refer to error terms like these as a *white noise process*. Such processes do not vary systematically with t. In the scenario is as

described above we could of course obtain a sample of observations $\{x_1, x_2, \ldots, x_n\}$ on $\widehat{R}(w^0)$ and then use the mean of these observations

$$\frac{1}{n}\sum_{t=1}^{n} x_i$$

as an estimator of $R(w^0)$[2].
However, our goal is *not* to know with (statistical) certainty what $R(w^0)$ is. It would be unfortunate to obtain samples of $R(w^0)$ with the aim of reducing uncertainty when w^0 corresponds to elements which no one clicks on. This would result in loss of revenue. This points us to the main problem with the one-step approach. As we have seen it aims to find to some extent some underlying deterministic function and then, through a series of steps, find the optimum. This is all well if it results in us finding a point which is optimal at time t and then continues to be approximately optimal in the future. In such a case we could say that we have found the optimal 'mean' input of the system. However, the empirical evidence suggest that the system is anything but stable. If the environment in which we optimize constantly changes we would have to constantly find new 'optimal' solutions. We must turn our focus to the design inherent in the one-step method. It is designed to provide us with *the* optimal solution. These algorithms essentially do not pay any attention to the output during the optimization (as long as the samples lead the algorithm to what it believes is a maximum) and the cost of taking samples on the function is not taken into account. The focus in these algorithms is too much on the 'end result' and does not pay enough attention to the intermediate steps leading up to an optimal solution. It is in fact the revenue generated during the optimization process which needs to be maximized. If we spend most of our time optimizing versus an environment which is bound to change as soon as we reach a potential maximum, thus having to repeat the optimization again, we simply spend most of our time optimizing! This insight points us to optimization algorithms relying on a different concept. The concept of being optimal with respect to a *planning horizon*. Its aim is to provide a smooth flow of optimal input parameters to the system in contrast to the one-step approach with its coup de grace solution. A 'planning algorithm' is characterized by an 'adaptive rule' which is specifically designed to provide optimal solution schemes that change or adapt in response to a dynamic environment. The discussion in the next chapter further motivates such algorithms and they will be the focus of the chapters to come. But before that we turn to some modifications that can make the one-step approach appear more suitable, depending to a large extent on the amount of available data. Suppose now on the other hand that the response of an element actually changes over time in a

[2]Now one can argue that it does not make sense to have measurement error on the variables. What we really observe is the *actual* click rates on the web. What constitutes this measurement error then? The fluctuations in the click rate we observe reflects user behavior on the internet occurring at a given time. We cannot however account for all this underlying behavior in our model, we want our objective function to reflect the 'mean response' of each weight vector. That is what click rate that can be *expected* for the elements at any given time. The error term simply represents all the factors we have not accounted for. Hence, the focus is on the mean response of the system. The key assumption here is that revenue should vary unsystematically around the mean response $R(w^0)$.

systematic fashion[3]. This implies that any estimator of $R(w^0)$ will be *biased*[4]. Some of the systematic time-dependent characteristics *could* of course be accounted for. Some more obvious than others. For example one ad element may be worthless during the night and very good during the day. If we take a sample of $R(w)$ during the night and one during the day, these samples will differ a lot (the estimator has huge bias) and should essentially be considered samples of two different functions. A better model would be

$$R(w,t) = R(w) + S(t) = R(w) + (s_1(t) + \cdots + s_n(t))$$

where the function $s_i(t)$ compensates for time specific characteristics of element i. Then we would use $R(w,t)$ when we sample. So we have a different model depending on when we want to sample the R function. Methods like these have the purpose of making $R(w)$ stationary in time.
We will not dwell further into these issues of the one-step approach but will highlight them again when we introduce the predictor in the planning algorithm.

[3]By systematic we mean distortions that are *not* due to measurement error since this type of error was defined as unsystematic.

[4]An estimator for a parameter θ is unbiased if it has expected value equal to θ. This is an often stated requirement of an estimator. The property establishes the *class of unbiased estimators*.[5]

Chapter 5

The Exploration vs Exploitation Trade-off

5.1 One-Step Optimization vs. Data Acquisition

When an ad-element has weight w_i at time t this means that this element is shown a fraction w_i of all the times the pool is shown. So the amount of data on each element depends on the weight. This is the data problem we have previously mentioned. Since we optimize with respect to the weights, the *performance* of the system depends on the weights. But since our knowledge of the system also depends on the weights we say that we have an *Exploration vs Exploitation tradeoff*. This basically means that we have to balance the acts of exploring/gaining knowledge of our system and that of purely exploiting the present situation. The main point here is that the purpose of 'exploring' is to gain knowledge to be able to make better decisions in the future.

Suppose for example that the pool has been shown 1000 times. If the weight is 0 for element i we will get no data at all on this element. It is then impossible to say anything about how good this direction is in the weight space. A naive way would be just to set it to zero, but then a global search would never explore this direction of the weight space again. It is clear that elements with low weights are not shown that many times and thus the CPI estimates of these elements will be unreliable. When the n-dimensional function which resides in the weight space is uncertain in some parts of the space, it will drastically reduce the performance of a global optimization algorithm. Algorithms will have a tendency to terminate prematurely due to the uncertainty and get stuck at local extrema. That is , fail to find the global maximum. We previously spoke of difficulties when evaluating $R(w)$ at some points. The reason for this is that we cannot calculate $R(w)$ in a reliable fashion when any component of the w vector is too small. Because this can cause the calculated CPI of one or more of the elements that are part of the R function to be misleading. Maybe over time, a global optimization procedure might compensate for this automatically by the nature of how an optimization algorithm works. Suppose we have created some meta-model for $R(w)$ and a

specific direction w_k in the weight space looks promising (due to overestimation). If the weight in this direction previously was very low, this direction is uncertain, however this may lead the algorithm to search in direction k, thus evaluating $R(w)$ at a point $w^{'}$ where the k:th component is greater. This in turn leads to more impressions on element k because the weight is set higher, and in turn a more reliable estimate of this direction is created to base future decisions upon. On the other hand, in this process, the algorithm may have to explore the whole weight space by oscillating between several overestimated parts of the weight space. This is clearly not desirable. A natural question is, if we know what the problem is, why do we not try to model it? This turns out to be more complicated than it sounds. We considered how one could add some kind of penalty term or factor in the objective function, for example

$$R'(w) = R(w) + \delta(w)$$

where δ is loss function which penalizes weights that are 'too extreme'. The loss function represents a way of handling the exploration vs exploitation trade-off. The purpose of δ is to penalize choices today that prevent us from making good decisions tomorrow. This function is really the key in providing a good algorithm. The real problem in this thesis is managing the exploitation vs exploration tradeoff, taking on the role of the δ-mechanism above.
The following section presents a simple and straightforward way of heuristically handling the exploration vs exploitation trade-off. The existing solution at TradeDoubler used the principle behind this method.

5.2 Introducing ε-type Algorithms

A common and simple way of heuristically dealing with the exploration vs exploitation trade-off is to allow an *exploration* algorithm, call it A, to control a fraction ε of all the impressions in the pool. Then we let another algorithm, the *exploitation* algorithm B, control the remaining fraction $(1 - \varepsilon)$. The algorithms are constructed for different purposes. The objective of the exploration algorithm is to provide information on element performance while the purpose of the exploitation algorithm is to use the information on element performance to maximize revenue. These conflicting objectives are then weighted together into the final element weight vector according to the ε-parameter according to

$$\mathbf{w} = \varepsilon \cdot \mathbf{w_A} + (1 - \varepsilon) \cdot \mathbf{w_B} \tag{5.2.1}$$

where the vector $\mathbf{w_A}$ contains the element weights from the exploration algorithm and the vector $\mathbf{w_B}$ contains the element weights from the exploitation algorithm. The purpose of the number ε is thus to define the relative merit of exploration compared to exploitation. Algorithms such as these are usually referred to as ε-type algorithms in the literature.

5.2.1 Simple Exploration Algorithms

The simplest form for the exploration algorithm is just a fixed uniform distribution of weights such that $\mathbf{w_A} = \frac{1}{n}\mathbf{1}$ where $\mathbf{1}$ is a vector of ones. This

ensures that all elements are shown. Letting the weights controlled by the exploitation algorithm be the vector $\mathbf{w_B}$ the final weight vector is

$$\mathbf{w} = \varepsilon \cdot \frac{1}{n}\mathbf{1} + (1-\varepsilon) \cdot \mathbf{w_B} \tag{5.2.2}$$

Choosing a weight vector of ones for the exploration part naturally produces an equal number of samples on each element. Instead of a vector of ones, any (fixed) 'prior' distribution could be chosen based on practical knowledge of the performance of the elements. At TradeDoubler, the constant weight vector in the secondary algorithm was specified by the owner of the web site, based on prior belief and experience regarding the revenue of the elements. Admittedly, it should be possible to specify a better constant weight vector than the somewhat naive vector of ones, but it still is an intuitive choice.

5.2.2 Simple Exploitation Algorithms

For the *exploitation algorithm* we specify something called *aggressiveness*. Aggressiveness is just a term used to quantify how heavily the exploitation algorithm should exploit the element with highest CPI.
In practice we calculate the CPI c_j of every element j during some time period. The length of the period is determined by how many impressions that the calculation should be based upon. To quantify aggressiveness a polynomial model is employed where the weights are determined by the relation

$$w_j = \frac{c_j^k}{c_1^k + c_2^k + \cdots + c_n^k} \tag{5.2.3}$$

where $\mathbf{w_B} = (w_1, \ldots, w_n)$ and k is a constant representing aggressiveness. Increasing values of k makes the algorithm react more heavily to changes in CPI when the weight is determined. It is easily verified that in all cases $\sum_{i=1}^{n} w_i = 1$.

- **The linear case** This means that we observe the CPI of each element and then set the weight *proportionately* compared to the other elements. The linear type corresponds to the case $k = 1$ in (5.2.3).
- **The quadratic case** This works like the linear case but the proportions are first squared, thus corresponding to $k = 2$ in (5.2.3).
- **The maximally aggressive** The strategy here is simply to show *only* the element with the highest CPI. Mathematically characterized as

  $$w_i = 1, \quad w_j = 0 \quad \forall j \neq i.$$

  This constitutes maximal exploitation in the current time period. In the framework of polynomial aggressiveness described by formula (5.2.3), maximal exploitation is the limiting case when 'aggressiveness tends to infinity'.

These three heuristics will be denoted ε-linear, ε-quadratic and ε-greedy respectively, and will be referred to when we present our test results in chapter 9.

5.2.3 Balancing Exploration vs Exploitation

How should ε be chosen? Generally it should be as low as one can get away with since a high value of ε implies higher exploration and thus detracts from the exploitation of good elements. On the other hand, if it is set too low, the accuracy of the performance estimates will be so low that the exploitation part basically ends up picking elements at random.
The optimal ratio ε thus differs depending on the nature of the application. In the case of TradeDoubler it is likely to vary between sites and affiliates. The two most significant factors in this case are probably the traffic intensity and the number of elements in the pool. Remember that the value of ε determines the *fraction* of observations available to estimate the revenue of the ad-elements. If there is large amount of traffic a small value of ε might be sufficient for an accurate estimation of element performance. In the same manner, a pool with many elements needs a higher ε to reach the same number of observations per element.
We admit that the algorithms presented here are very simple heuristic solutions. It is difficult to find any conditions where it is possible to prove that these ε-heuristics are best, consequently we label them *suboptimal* heuristic solutions. Although further modeling can still be made within the ε-type framework. For example, if an element is *known* to be worthless during the night, then it would be foolish to collect data during the night for this element. Time-dependence in the form of seasonal functions can be included in the ε-heuristics as well. In the next chapter we will pave the way for solutions that *can* be proven optimal.

Chapter 6

The Planning Algorithm

6.1 Introduction

We will in this section make the following assumption

> There is an underlying probability of each impression generating a click that is independent of how often this element is shown.

Consider a *weight based policy* over a time period t_1 to t_2. This is a policy which is based upon optimization of the aggregate $R(w)$ to produce a flow of m input parameters $\{w_1, w_2, \ldots, w_m\}$ in the time period t_1 to t_2. Each w_k lies in the weight space as defined in section (4.1). An *equivalent policy*, under the stated assumption, can be produced by dividing up the interval $[t_1, t_2]$ into suitable subintervals and showing *one element at a time* in each subinterval . Suppose we have a pool with 2 elements and we want to show one element 70% of the time and another element the remaining 30%. In this case $w = (0.7, 0.3)$, further let the length of the interval be 10 time units. Equivalently we could produce the same result by having ten 'choices' instead of 1, and show the first element 7 out of these ten times, the second element the remaining 3. The problem has thus been reduced to a *discrete* selection problem. For a pool with n ad-elements, a discrete set of n choices is vastly easier to enumerate over in search of a maximum than a surface in a n-dimensional space! This reduction is also interesting because the most extensively studied forms of exploration vs. exploitation trade-off problems in the literature are computed over discrete search spaces.
Instead of using the aggregate $R(w)$ for the whole system, we will treat every ad-element *individually* and construct a model for how each element behaves separately. This builds on the assumption that the ad-elements evolve independently through time according to a model. We propose an algorithm which shows *one element at a time* 100%. In the language of the previous section our algorithm is maximally aggressive in each time period. But we will now be optimizing with respect to a planning horizon instead of a single time step. We exploit in each time period but we have an explorative outlook when our algorithm is looked upon with regard to the time horizon. When we choose an element we will show it for a length of time which we call the *duration*. The duration is simply chosen so that the banner element is shown a

fixed number of impressions each time the banner is selected. The length of the duration is of course a parameter subject to tuning[1]. For the length of the duration we collect observations of the element. Each observation is an impression with two different outcomes. The impression can either result in a click or not in a click. Every time we get an impression on an ad-element we thus observe a binary outcome random variable.
Planning problems occur frequently in the literature. One of the most well-known problems is that of planning a route with shortest length through a road network from point A to point B. Road networks can be modeled with a graph where each node represents a point where a decision is made as to which arc to take in taking the next step through the graph. Each arc has a length specified. A solution to this planning problem is a set of arcs that together form a path from point A to B with minimal length. A detailed discussion of this problem together with the necessary background on graph theory can be found in [22]. Historically methods for the solution of these problems are called *Dynamic Programming* and dates back to Bellman's famous work in 1957.
In the this example we specified a 'length' on each arc and the goal was to minimize the total length of the arcs in the path through the network. Any type of cost or reward can be specified on the arcs. If we have random quantities on the arcs we enter the realm of *Stochastic Programming*. In this case the cost of traversing or selecting an arc in the graph will be an outcome of a stochastic variable instead of a fixed cost specified on the arc. The solution methods of Stochastic Programming are very similar to Dynamic Programming. The approach we will use is that of *Discrete Stochastic Dynamic Programming*, where the word 'discrete' emphasizes that decisions are made at discrete points in time.

6.2 Theoretical Foundations

In this section we provide the theoretical background of why we have chosen the particular solution method presented in chapter 8. In choosing the modeling technique we have been inspired by a class of problems called *bandit problems*.

6.2.1 Bandit Problems

The *multi-armed bandit problem* has its origins in the old fashioned slot machines, still seen in casinos all around the world. When you pull a lever on the machine you earn a reward. The reward on each lever can be taken as a stochastic variable. Assume $k \geq 2$ number of levers. If these levers are pulled sequentially, a sequence of rewards $(Z_1, Z_2, Z_3, \ldots)$ is observed. Then one typically defines the *expected payoff* as

$$E\left[\sum_{i=1}^{\infty} \alpha_i Z_i\right]$$

where $(\alpha_1, \alpha_2, \ldots)$ is a *discount sequence* with $\alpha_i > 0$. The discount sequence reflects the time-preference of rewards. The geometric discount sequence

[1] Specification of the duration depends on the traffic intensity (impressions/minute) at the web site. See appendix (B).

involving powers of α, α^k where $0 < \alpha < 1$ is common in the literature where the importance of earlier rewards can be specified to decline geometrically. The objective of an agent pulling levers is to maximize the expected payoff by making informed choices of which levers to pull (in our setting, selecting an arm to pull represents selecting an ad element to show to a visitor, and the reward represents clicks resulting from that action). The standard solution of this type of problem rests on the following assumptions

1. Levers are independent of each other.

2. The rewards for the levers are governed by a (known) fixed family of distributions.

When the family of reward distributions is known it is possible to estimate the parameters of this family, either by classical parameter estimation or by bayesian estimation. For example, suppose each lever k follows a bernoulli distribution with parameter θ_k, then the family of distributions is the bernoulli family and the parameter to be estimated is θ_k. The standard approach is to use the bayesian approach to estimation when estimating parameters[2]. The reason for this is that the estimation technique is particularly useful for 'adaptive learning'. When data arrives sequentially as levers are pulled, distributions are conveniently updated by means of Bayes Rule. Because of this, such solution methods are in the literature referred to as the *Bayesian* approach to bandit problems. This approach is a meld of bayesian estimation techniques and dynamic programming. By generating a search tree over all possible lever selections and outcome scenarios in each time step, and iteratively updating the mean estimate of each distribution by Bayes' rule one can select levers in a way that constitutes a strategy, that will generate the highest total expected payoff over the time horizon. Such a strategy is then *optimal* with regard to the statistical model. The analogy between levers and ad-elements should now be apparent. The main and important difference between the bandit problem and our formulation is that in the bandit formulation, only the parameters of levers which are pulled may change over time. The states of all other elements remain fixed. When a lever is pulled, the estimated distribution for this lever is updated, but the estimates of all other levers remain fixed. Consequently, the states of all elements are always known in the bandit-problem. In our problem on the other hand, only the reward distribution of the ad-element currently being shown on the web site is known. All the other elements evolve according to an unknown process, governed by user behavior on the internet. The fact that we only know the CPI of one element at a time forces us to provide a stochastic (process) model for the other elements as opposed to static distributions in the original bandit-problem. This is what constitutes our 'dynamic environment'. Additionally, planning with regard to an infinite number of rewards is unrealistic for our problem. For example, it is far from evident how the discounting should take place, and we have not studied the practical interpretation of an infinite horizon. We will consider a *finite* horizon with a

[2]See 8.7.2 for a brief introduction to bayesian estimation and bayesian updating in the Bernoulli family.

uniform discount sequence, taken as

$$\alpha = (1, 1, \ldots, 1, 0 \ldots)$$

and solve the planning problem which maximizes the expected value of

$$\sum_{i=1}^{\infty} \alpha_i Z_i = \sum_{i=1}^{N} Z_i$$

for some finite N. In the case of a finite horizon the planning problem can be solved with dynamical programming using methods of backward-induction which will be the topic of chapter 8. The solution through dynamic programming provides an optimal strategy, although the method is regarded as computationally intractable because the computations grow exponentially with the number of time steps considered [2].

6.2.2 Research Overview

The purpose of the statistical planning algorithm described above is solving the exploration vs. exploitation problem, a problem which is inherently multidisciplinary. It is treated at length in many diverse areas of research. In reinforcement learning [17], [29], an agent has to learn to act in an unknown environment. In control theory[3], [32] the objective is to keep an unknown process within certain boundaries and in economics where pricing strategies are devised when demand is unknown [3]. Finally, but not least, in statistics where Robbins [34] first formalized the bandit problem from the perspective of sequential analysis and hypothesis testing.
Here however we will just note that there are many variants of the problem, but few of them have computationally tractable optimal solutions. In response, many heuristical solutions have been proposed (see [31]). The ε-greedy method described earlier is one of the most widely known. Some of the heuristics are at least asymptotically optimal on static reward distributions. That is, as time tends towards infinity the solution is guaranteed to converge on the 'best' element. This result does not however apply in dynamic environments. Further, asymptotic results such as these give few guarantees on the performance in the short term.
To the authors' knowledge there has been no attempt at solving the full finite horizon planning problem when the reward distributions are dynamic. We develop an approximate solution to this problem.

[3] A subfield that directly deals with the planning approach outlined earlier is that of adaptive optimal *dual* control.

Chapter 7

Statistical Models of Element Uncertainty

In the previous chapter we said that each ad-element follows a distribution which can either be stationary or dynamical. In the case of stationary distributions we mentioned that if the family of distributions is known we can estimate the mean reward (as a measure of how good an element is) with parametric statistical methods or bayesian methods. We concluded that in our case we have dynamical distributions which change over time. This section will construct a model for this dynamical case. The parameters of the model we propose will be estimated trough standard parameter estimation, although we admit that a Bayesian technique may be applied. The purpose of this chapter is to construct a predictor as a stand-alone component. All that is required of a predictor is a rule which assigns probabilities to different states as will become clear in chapter 8. For now we emphasize that the optimization algorithm requires a predictor and that the performance of the optimization algorithm depends on the accuracy of the predictor.

The question of how to model a process over time arises in many different disciplines and is the main focus in for example the field of *Time Series Analysis*. What separates these models is essentially the number of assumptions regarding the underlying mechanics which dictates the evolution of the process. What we have found out based on our analysis of the data at TradeDoubler is that there are many factors that influence data. Many of these are hard to describe or characterize. In these situations a model with minimal assumptions regarding structure of the process is preferred. We will assume that elements we do not observe follow a *Random Walk Model*. This is a very simple model with few structural assumptions. Its simple form makes it useful for modeling a wide variety of processes. It is perhaps most well known in the field of financial engineering where it is a building block in developing more elaborate models of how the price of financial derivatives fluctuate with time. See for example [21]. Although a more complex model can be utilized, remember from the overview on function modeling in chapter 3 that for increasing complexity one needs more data. Also remember that we concluded in chapter 4 that there was not enough available data for the proposed model. In fact, we showed that the availability of data has to be taken into account in

the optimization, which was discussed in chapters 5 and 6. These reasons together justify choosing the random walk, which is the simplest possible non-static temporal model.

7.1 Estimating the CPI

The first thing we need to do is make precise what we mean when we say that an impression is an outcome of a binary stochastic variable. When an ad-element is shown on a web site once, it can either be clicked on or not clicked on. Assume that we show ad-element k a length of time such that it results in m number of impressions. During this time the element accumulates a number of clicks equal to

$$C_k = \sum_{i=1}^{m} y_i$$

where

$$y_i \sim \text{Bernoulli}(\theta_k)$$

Here θ_k is the *probability* of clicking on element k. Each time the element is shown can be seen as a 'trial' , which can either result in a click or non-click. This means that $y_i = 1$ (click) with probability θ_k and $y_i = 0$ (non-click) with probability $1 - \theta_k$. Of course each trial faces a different web site visitor and preferences are likely to differ between different visitors so each of these trials have in reality different θ_k. However we assume all these Bernoulli variables to be identically distributed so that θ_k can be interpreted as the 'average' rate of clicking on the ad-element among all visitors. We are summing Bernoulli variables so we know that this sum follows the binomial distribution

$$C_k \sim \text{Binomial}(m, \theta_k)$$

Here θ_k is a parameter to be estimated. This is the essence of the CPI estimation. The obvious estimator in the binomial family is $\overline{c} = C_k/m$, the mean of the observations, where $E[C_k/m] = E[C_k]/m = \frac{m \cdot \theta_k}{m} = \theta_k$ by well known properties of the binomial distribution. The Weak Law of Large Numbers from probability theory shows that this estimator is consistent [1] e.g.

$$\frac{1}{m} \sum_{i=1}^{m} y_i \longrightarrow \theta_k$$

in probability as m tends to infinity.
We define the CPI of element k as

$$\text{CPI}_k = \frac{C_k}{m} = \frac{1}{m} \sum_{i=1}^{m} y_i$$

so when m is 'sufficiently large' we can make the approximation

$$\frac{c_k}{m} \approx \theta_k \tag{7.1.1}$$

[1] For more information on consistency see for example [5]

where c_k is an observation of the random variable C_k. The *sample error* is thus the difference between CPI_k and the true value of the CPI, namely θ_k, defined as

$$\textbf{Sample Error} = \frac{1}{m}\sum_{i=1}^{m} y_i - \theta_k \tag{7.1.2}$$

Now since $\sum_{i=1}^{m} y_i$ is binomially distributed, we can use the approximation

$$\sum_{i=1}^{m} y_i \sim \text{Bin}(m, \theta_k) \approx \mathcal{N}(m\theta_k, m\theta_k(1-\theta_k))$$

thus $\frac{1}{m}\sum_{i=1}^{m} y_i \sim \mathcal{N}\left(\theta_k, \frac{\theta_k(1-\theta_k)}{m}\right)$ and so we have established that

$$\textbf{Sample Error} \sim \mathcal{N}\left(0, \frac{\theta_k(1-\theta_k)}{m}\right) = \mathcal{N}(0, \sigma_s^2)$$

The quantity $\sigma_s^2 = \frac{\theta_k(1-\theta_k)}{m}$ will be referred to as the *sample error variance*. Note that it is decreasing in m and approaches zero as $m \to \infty$. For analytical reasons it is much more convenient to have a normally distributed sampling error, as will be seen in section 7.4.

7.2 The Basic Random Walk

We have now defined the sample error which occurs when an ad-element is 'sampled'. The next step is to provide a model how θ_k evolves through time when it is not observed. The idea is to let an element we do not observe evolve as a random walk from a starting value equal to the last observed CPI for this element. In each step we do not observe it, we add a normally distributed error term. To introduce the random walk in a simple manner assume that the sample error in (7.1.2) is zero, meaning that $\frac{1}{m}\sum_{i=1}^{m} y_i = \theta_k$, in section 7.4 we will relax this assumption.
Let the CPI in period t be described by the random variable Y_t. Assume we observe an element, say k, in period t. This yields m impressions on element k, and the average of these $\frac{1}{m}\sum_{i=1}^{m} y_i$, is in absence of sample error, equal to the true CPI, θ_k. Thus

$$Y_t = \frac{1}{m}\sum_{i=1}^{m} y_i = \theta_k \tag{7.2.1}$$

If the CPI of this element is unobserved in next period (that is it is not selected) we model its CPI in the next period by

$$Y_{t+1} = Y_t + \varepsilon_t \tag{7.2.2}$$

where

$$\varepsilon_t \sim \mathcal{N}(0, \sigma_v^2)$$

so that

$$Y_{t+1} \sim \mathcal{N}(\theta_k, \sigma_v^2)$$

where ε_t is a white noise process with variance σ_v^2, we thus make the following important definition

$$\textbf{Volatility} = \sigma_v^2$$

> The volatility measures how much, on average the CPI of an element changes in one time step.

The idea with this model is to increase the uncertainty of elements we do not observe. For every period we do not observe an element we add a normally distributed random variable to this element. If we let this model evolve several steps into the future we get a prediction for Y_{t+h} where $h \geq 1$ (this just builds upon equation (7.2.2))

$$Y_{t+h} = Y_t + (\varepsilon_t + \varepsilon_{t+1} + \cdots + \varepsilon_{t+h-1}) \tag{7.2.3}$$

Where h is the number of steps that have passed since we last sampled the element. By taking expectation and variance of (7.2.3) we obtain

$$Y_{t+h} \sim \mathcal{N}(\theta_k, h\sigma_v^2) \tag{7.2.4}$$

7.2.1 Proof that Y_t is the BLUP in Absence of Sample Error

Recall equation (7.2.1). We will in this section show that

$$Y_t = \theta_k = \mathbf{E}[Y_{t+h}|Y_t, Y_{t-1}, \ldots, Y_1] \tag{7.2.5}$$

That is, in the absence of sample error, Y_t is the *best prediction of the expected value* of Y_{t+h} based on the t earlier terms

$$Y_1, Y_2, \ldots, Y_{t-1}, Y_t \tag{7.2.6}$$

In fact, the expectation on the right-hand side in (7.2.5) is referred to as the *best linear unbiased predictor* (BLUP) for the underlying model. Keep in mind that even though the proposition that only the most recent sample is needed to know the CPI of an element in the absence sample error may seem trivial, when we introduce sample error in section 7.4, the BLUP will not have the simple form as in this section. Since when introducing sample error, the need to look at samples further back in time in the series (7.2.6) arises, in order to reduce the uncertainty due to sampling.
The BLUP (for h steps into the future) is expressed as

$$\mathbf{E}[Y_{t+h}|Y_t, Y_{t-1}, \ldots, Y_1] = P_t Y_{t+h} \tag{7.2.7}$$

where P_t is called the *prediction operator* based on the t earlier terms, using the time series representation in (7.2.6). The prediction (7.2.5) of the future value Y_{t+h} is, because of the underlying random walk model representation

(7.2.3), a linear combination of normally distributed error terms[2]. The BLUP P_tY_{t+h} is formally defined as the linear combination of previous values of the time series which minimizes the squared error of prediction

$$E[(Y_{t+h} - P_tY_{t+h})^2] \tag{7.2.8}$$

and forms an *unbiased* predictor by satisfying

$$E[Y_{t+h} - P_tY_{t+h}] = 0 \tag{7.2.9}$$

In fact (7.2.8) can be shown to be satisfied if the predictor is unbiased according to (7.2.9), and that the linear combination of terms, constituting the predictor, are uncorrelated with the error $E[Y_{t+h} - PY_{t+h}]$. We will verify these two properties for the simple random walk model without sample error, which turns out to be fairly trivial. In other words, we want to show that $P_tY_{t+h} = Y_t$ fulfills these two conditions, and prove the proposition (7.2.5) initially stated - that is, only the very last sample in (7.2.6) will be a component of the BLUP.
Insertion of the expression for Y_{t+h}, (7.2.2) , and $P_tY_{t+h} = Y_t$ into (7.2.9) yields

$$\begin{aligned}\text{bias} &= E[Y_{t+h} - P_tY_{t+h}] = \\ &= E[(Y_t + \varepsilon_t + \ldots + \varepsilon_{t+h-1}) - Y_t] = \\ &= E[(\varepsilon_t + \ldots + \varepsilon_{t+h-1}] = 0\end{aligned}$$

which shows that the predictor is unbiased. Further,

$$\begin{aligned}\text{covariance} &= E[(Y_{t+h} - P_tY_{t+h})Y_{t-j}] \\ &= E[(\varepsilon_t + \ldots + \varepsilon_{t+h-1})Y_{t-j}] = 0, \quad j = 0 \ldots t-1\end{aligned}$$

which shows that the errors are uncorrelated with the previous terms Y_{t-j} by using the uncorrelatedness of the ε_t terms. Consequently,

$$P_tY_{t+h} = \mathbf{E}[Y_{t+h}|Y_t, Y_{t-1}, \ldots, Y_1] = Y_t = \theta_k$$

which was to be proven.

7.3 Estimating the Volatility σ_v^2

Uncertainty in our model arises in two different ways. The first one is the sample error which was defined in section (7.1). We showed that the sample error decreases with n - the number of trials (impressions) used in the estimation. But we then made the simplifying assumption that the sample error was zero, and derived the random walk predictor based on this assumption. In this section we will discuss the second type of error, the volatility. We said that this was a measure of how much on average an ad-element changes during one time period. This error is purely model-based and is the parameter σ_v^2 in the distribution governing the random walk model. Since σ_v^2 is unknown it must be estimated. We will make the simplifying

[2]See section 2.5 of [4] for details on the predictor operator.

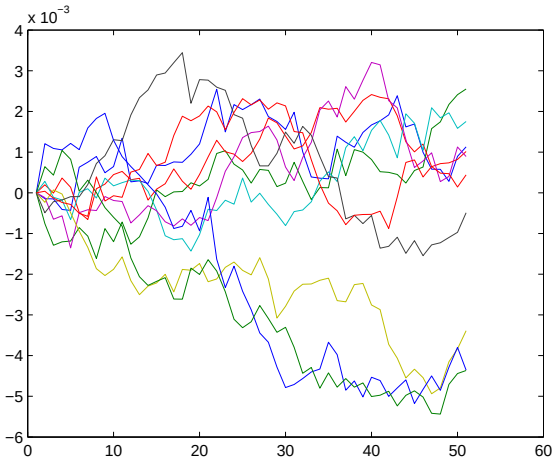

Figure 7.1: Simulation of 10 random walks with starting value 0 for a duration of 50 periods

assumption that all elements have the same level of volatility. This is practical because it increases the amount of data available to estimate σ_v^2, however a model with element-specific volatility could in theory increase accuracy, if it is believed that there are substantial differences between the variances (volatility parameters) of the sample paths for different ad-elments. Figure 7.1 shows a set of sample paths for ad-elements evolving through time according to random walk models with the same volatility.
Because of our assumptions regarding the error term in (7.2.3) we can form the difference $\Delta Y_t = Y_t - Y_{t-1}$, which fortunately is a serially uncorrelated process. Thus we will in practice use the *observed* differences between two samples of an element, separated in time, to form an estimator of σ_v^2. Recall that the planning algorithm always shows *one element at a time*. So in order to form a difference of this type, we must first observe an ad-element once, then wait until the algorithm decides to show this element again. This way we can measure how much the element has changed between two points in time. So assume we observe the revenue of some element at time t, and that the next time we observe this element is h steps later. Then we can form the difference

$$\Delta Y_t = Y_{t+h} - Y_t$$

the value of Y_{t+h} is the value Y_t plus h random shocks (recall the definition of the random walk in (7.2.3)). Consequently, the difference above is the sum of these h random shocks. We want to estimate the random shock for *one* step. According to the underlying model, and the distribution of the h step predictor (7.2.4), we can obtain one sample on the volatility σ_v^2 by just dividing by h as follows

$$\frac{Y_{t+h} - Y_t}{h} = \delta \tag{7.3.1}$$

Since all elements share the same volatility, we can use differences defined by (7.3.1) from each element to estimate σ_v^2. The only requirement is that the

components Y_{t+h} and Y_t used in forming each difference, must be samples of the same element. Over a horizon T consisting of many time steps, several different ad-elements are shown, so many differences can be formed. Note again that only elements which have been shown more than once during this horizon will give rise to a difference sample. Let δ_i^k be the i:th difference formed using two samples of element k according to (7.3.1), and let the total number of differences formed on element k during the horizon T be n_k. Then we can sum all the differences of this type during the horizon T and obtain an estimator ψ^2 of σ_v^2 defined as

$$\psi^2 = \frac{\sum_{k=1}^{L} \left[n_k \sum_{i=1}^{n_k} (\delta_i^k)^2 \right]}{n_1 + n_2 + \cdots + n_L} \tag{7.3.2}$$

where we assume there are L different ad-elements. Note that $m = n_1 + n_2 + \cdots + n_L$ is the total number of samples (differences) generated through this process. This is simply a statistically correct 'mean' of the deviations which arise as described above, where a relative weight n_k is attached to the sum of differences generated by element k. In general the number of samples (differences) m will be small if there are many ad-elements (L is large) and few selections (samples) of ad-elements. Although it is theoretically possible to get a sufficient number of repetitions on each element to estimate σ_v^2 with the difference estimator ψ^2, in practice however our simulations show this can yield a unreliable results initially. Therefore performance is enhanced by specifying a *prior volatility* common to all elements. Then we can *gradually* weight together the estimated volatility obtained through the difference procedure described above with the prior volatility. We have used a method from statistics for this, resembling the standard 'variance pooling' formula

$$\hat{\sigma_v}^2 = \frac{n_p s_p^2 + m\psi^2}{n_p + m} \tag{7.3.3}$$

here n_p is the hypothetical 'prior' sample size fixed beforehand. m is the number of samples (differences) from the difference estimation. As m grows (as more and more differences are formed and added in the sum (7.3.2) increasing the accuracy of the estimator ψ^2) more and more significance will be given to the difference estimator.

7.4 Random Walk with Sampling Error

In this section we will expand on the dynamic model, incorporating a sampling error in the model. This implies a need to take repeated samples (i.e taking several measurements of CPI to estimate θ_k). We show that the 'dynamical mechanism', the volatility concept of the random walk, will still allow for accurate estimation, because the value of samples further back in time is small compared to more recent samples in a dynamic environment. In the extension of the random walk linear process (with a sampling error component) we show that the larger the volatility is compared to the sampling error, the less samples further back in time contribute to the best linear predictor. This is essentially governed by the ratio σ_s/σ_v. Since the elements are volatile (or dynamic), the value of taking repeated measurements depends on this ratio, as samples further back in time quickly become obsolete.

7.4.1 Random Walk Process with Sampling Error

Recall equation (7.2.2) which can equivalently be expressed as

$$Y_t = Y_{t-1} + \varepsilon_t, \quad \{\varepsilon_t\} \sim \mathcal{N}(0, \sigma_v^2) \tag{7.4.1}$$

where Y_t is the current value of the element reward process without sample error, i.e an observation is of the form $Y_t = \theta_k$ where θ_k is the true CPI of element k. As previously, σ_v represents the rate at which the process changes in one time period. Suppose now that we do not directly observe Y_t but instead a new process X_t defined as

$$X_t = Y_t + \delta_t, \quad \{\delta_t\} \sim \mathcal{N}(0, \sigma_s^2) \tag{7.4.2}$$

where δ_t are i.i.d. errors induced from sampling as discussed in section 7.1 with standard deviation σ_s. So X_t represents a reading at time t of the CPI of element k with the addition of measurement noise. The measurement error affects the best linear predictor of Y_{t+1}, the true CPI in the next step. If the sampling error is large compared to the volatility we expect the BLUP will resemble the arithmetic mean. The arithmetic mean assigns uniform weights to samples regardless of their age. This is the optimal predictor in a static environment, where the problem reduces to estimating the mean of the noise distribution. At the other extreme, we showed in section 7.2 that the optimal predictor of a random walk is just the last sample, assigning no weight at all to any older samples. It is expected that having a model with both measurement error and volatility produces an optimal predictor lying somewhere in between these two extremes. A preference will be given to more recent samples, but older samples will enter the predictor as well.
Deriving the BLUP of X_t is somewhat formidable, because a random walk is non-stationary since the variance of the terms X_t increase with time[3] . This makes expressing the covariance functions a bit tricky. However, X_t can be treated as an ARIMA process. In section 3.3.4 we introduced the ARMA processes. An ARIMA(p,d,q) process is an extension of the ARMA(p,q) process to handle non-stationary series by *differencing* the time series d times

[3]Stationarity was defined in section 3.3.4

before applying the ARMA model. The first difference of the pure random walk Y_t is

$$D_t = Y_t - Y_{t-1} = (Y_{t-1} + \varepsilon_t) - Y_{t-1} = \varepsilon_t$$

which is just white noise, thus differencing once eliminates the non-stationary behavior of the process. In the same fashion, the first difference of the random walk with sample error X_t is

$$\begin{aligned} D_t = X_t - X_{t-1} &= (Y_t + \delta_t) - (Y_{t-1} + \delta_{t-1}) \\ &= ((Y_{t-1} + \varepsilon_t) + \delta_t) - (Y_{t-1} + \delta_{t-1}) \\ &= \varepsilon_t + \delta_t + \delta_{t-1}. \end{aligned} \tag{7.4.3}$$

This is just a sum of uncorrelated error terms and the differenced series is even in this case stationary. Notice that the only lagged term present is δ_{t-1} and error terms are by definition i.i.d. and only correlated with themselves which implies that the autocorrelation function of D_t is zero for lags greater than one. This is why this series is called 1-correlated. By proposition 2.1.1 in [4], any stationary, q-correlated series can be represented as an MA(q) process, so D_t can be modeled as an MA(1) process. Consequently, X_t is an ARIMA(0,1,1) process. To summarize, X_t is first differenced once, this produces an MA(1) or an ARMA(0,1) process (these are just two different notations for the same process), establishing that X_t is an ARIMA(0,1,1) process.
Having characterized X_t as an ARIMA process, we can use ARIMA forecasting of section 6.4 of [4] for our purposes. In general when forecasting ARIMA processes, the best linear predictor (one step into the future) can be formulated as

$$\begin{aligned} P_t X_{n+1} &= P_t(X_0 + D_1 + \cdots + D_t + D_{t+1}) \\ &= P_t(X_t + D_{t+1}) = X_t + P_t D_{t+1}. \end{aligned} \tag{7.4.4}$$

since $X_t = X_0 + D_1 + \cdots + D_t$ and $P_t X_t = X_t$ because the best prediction of X_t when X_t is known is X_t (recall the definition of P_t (7.2.7) as the best linear predictor based on the t earlier terms, further the linearity of P_t enables us to express the prediction of a sum as a sum of predictions). So the optimal predictor of X_t amounts to optimal predictor of D_t and adding the last sample X_t.

7.4.2 Large Sample Best Linear Predictor

Expression (7.4.4) points us to the BLUP of D_{t+1}, that is the BLUP of an MA(1) process. Finding a best linear predictor $P_t D_{t+1}$ amounts to solving a least squares problem of the n (known) earlier values of the time series. However our task is to show in what fashion the importance of earlier samples change with time. This leads us to considering instead the *large sample* predictor P (to be distinguished from the finite sample predictor P_t of the previous section).

$$P D_{t+1} = \mathbf{E}[D_{t+1} \mid D_t, D_{t-1}, \ldots] \tag{7.4.5}$$

The large sample predictor is defined as the predictor of the future value $P D_{t+1}$ in a least squares sense based on an *infinite* set of samples. In some

sense this can be seen as finding the solution to an infinite set of linear equations. Fortunately, solving such problems is easy for invertible processes. The concept of invertibility of a time series was introduced in section 3.3.4. Generally, an (AR)MA process has an inverse as long as the (complex) roots of the filter polynomial $\theta(B)$ lies within the unit circle $|z| < 1$. Since an MA(1) process has the representation[4]

$$X_t = Z_t + \theta Z_{t-1} = (1 + \theta B)Z_t = \theta(B)Z_t$$

Where $\theta(B)$ is the first degree polynomial $(1 + \theta B)$, the invertibility criterion reduces to the condition that $|\theta| < 1$. We will restrict attention to invertible MA(1) processes, because as will be shown the differenced series D_t has such a representation. Assuming that the MA(1) process is invertible (with an analogous argument of that which led us to equation (3.3.10)) the process can be expressed as

$$\begin{aligned} D_t &= \theta(B)Z_t, \quad \{Z_t\} \sim \mathcal{N}(0, \sigma^2) \\ Z_t &= \frac{1}{\theta(B)} D_t. \end{aligned}$$

Where Z_t are the usual i.i.d noise terms. For an MA(1) process we thus have

$$Z_t = \frac{1}{1 + \theta B} D_t \tag{7.4.6}$$

or

$$Z_t = \sum_{j=0}^{\infty} (-\theta)^j D_{t-j} \tag{7.4.7}$$

by using the geometric series representation of $\frac{1}{1+\theta B}$ which is valid only for $|\theta| < 1$, a guaranteed condition when the series is invertible. Manipulating equation (7.4.7) yields

$$D_t = Z_t - \sum_{j=1}^{\infty} (-\theta)^j D_{t-j}$$

and by applying the prediction operator P

$$\begin{aligned} PD_{t+1} = P\left[Z_{t+1} - \sum_{j=1}^{\infty} (-\theta)^j D_{t+1-j} \right] &= -\sum_{j=1}^{\infty} (-\theta)^j P D_{t+1-j} \\ = -\sum_{j=1}^{\infty} (-\theta)^j D_{t+1-j} \end{aligned} \tag{7.4.8}$$

[4]Unfortunately it is standard to use the same notation θ for both the parameter θ as well as the function $\theta(\cdot)$.

since $PZ_{t+1} = 0$ and $PD_{t+1-j} = \mathbf{E}[D_{t+1-j} \mid D_t, D_{t-1}, \ldots] = D_{t+1-j}$ since D_{t+1-j} is treated as a constant in the expectation conditioned on earlier values $D_t, D_{t-1}, \ldots$. This establishes the expression for PD_{t+1}, the best linear predictor of D_{t+1} based on an infinite number of earlier values[5].
We were originally interested in the best linear predictor in terms of X_t, not the differenced series D_t. Insertion of (7.4.8) in (7.4.4) yields

$$PX_{t+1} = X_t + PD_{t+1} = X_t - \sum_{j=1}^{\infty}(-\theta)^j D_{t+1-j}$$

Since $D_t = X_t - X_{t-1}$ we can express the series in terms of X_t. The goal is to manipulate the sums until we finally get an expression for the large sample best linear predictor PX_{t+1} in terms of lagged values of X_t.

$$\begin{aligned}
PX_{t+1} &= X_t - \left[\sum_{j=1}^{\infty}(-\theta)^j X_{t+1-j} - \sum_{j=1}^{\infty}(-\theta)^j X_{t-j}\right] \\
&= -\sum_{j=1}^{\infty}(-\theta)^j X_{t+1-j} + \sum_{j=0}^{\infty}(-\theta)^j X_{t-j} \\
&= -\sum_{j=0}^{\infty}(-\theta)^{j+1} X_{t-j} + \sum_{j=0}^{\infty}(-\theta)^j X_{t-j} \\
&= \theta\sum_{j=0}^{\infty}(-\theta)^j X_{t-j} + \sum_{j=0}^{\infty}(-\theta)^j X_{t-j} \\
&= (1+\theta)\sum_{j=0}^{\infty}(-\theta)^j X_{t-j} \qquad (7.4.9)
\end{aligned}$$

From (7.4.9) it is clear that the coefficients of older values of X_t decay exponentially with θ, meaning that older terms will have a diminishing impact on the best linear predictor. How fast the value of older samples decay is determined by θ. Values of θ close to 1 means that older samples will decay slower than for smaller values of θ. This insight should be coupled with the fact that the variance of older terms in the underlying random walk Y_t is smaller than more recent samples (recall that Y_{t+h} has h times the variance of Y_t).
The parameter θ is the theoretical construct relating volatility and sampling error in the ARIMA(0,1,1) representation of X_t. We will now show how σ_s and σ_v relate exactly as well as prove the earlier assumption that the MA(1) process D_t has an invertible ($|\theta| < 1$) representation.

7.4.3 Expressing θ in terms of σ_s and σ_v.

We now want to determine the parameter θ of the MA(1) process. Any general MA(1) process has the autocovariance function $\varphi_1(h)$ defined as

$$\varphi_1(h) = \begin{cases} (1+\theta^2)\sigma^2 & \text{if } h = 0, \\ \theta\sigma^2 & \text{if } |h| = 1. \\ 0 & \text{if } |h| > 1. \end{cases}$$

[5] See Section 2.5 of [4] for a brief introduction to large sample prediction.

where θ and σ^2 are the parameters of the model. We are interested in the MA(1) representation of $D_t = X_t - X_{t-1}$. It is straightforward to establish that the autocovariance function for D_t is $\varphi_2(h)$ given by

$$\varphi_2(h) = \begin{cases} \sigma_v^2 + 2\sigma_s^2 & \text{if } h = 0, \\ -\sigma_s^2 & \text{if } |h| = 1. \\ 0 & \text{if } |h| > 1. \end{cases}$$

where σ_v^2 is the volatility and σ_s^2 is the sampling error of the process. We now want to establish the MA(1) representation of D_t. This amounts to determining the MA(1) model parameters σ^2 and θ. However, we are only interested in the parameter θ because it enters the predictor in (7.4.9). We can find θ (and σ^2) by setting $\varphi_1(h) = \varphi_2(h)$, this gives two equations and two unknowns, so it is possible to solve for θ. The expression for θ is a function of both σ_s and σ_v, enabling us to study the predictor (7.4.9) in more detail by analyzing the relationship between the volatility and sampling error in the expression for θ. In either case, solving for θ amounts to solving the quadratic equation

$$\theta^2 - \left(\frac{\sigma_v^2}{\sigma_s^2} + 2\right)\theta + 1 = 0$$

with solutions

$$\theta = \frac{\sigma_v^2}{2\sigma_s^2} + 1 \pm \sqrt{\left(\frac{\sigma_v^2}{2\sigma_s^2}\right)^2 + \frac{\sigma_v^2}{\sigma_s^2}}.$$

Since $\frac{\sigma_v^2}{2\sigma_s^2} + 1 = \sqrt{\left(\frac{\sigma_v^2}{2\sigma_s^2} + 1\right)^2}$ we can rewrite the above expression as

$$\theta = \sqrt{\left(\frac{\sigma_v^2}{2\sigma_s^2}\right)^2 + \frac{\sigma_v^2}{\sigma_s^2} + 1} \pm \sqrt{\left(\frac{\sigma_v^2}{2\sigma_s^2}\right)^2 + \frac{\sigma_v^2}{\sigma_s^2}}.$$

Either solution yields a valid MA(1) representation of D_t with parameter θ. But it is only the solution with a (-) which yields an invertible process which is valid for the predictor in (7.4.9). The solution of interest is thus

$$\theta = \sqrt{\left(\frac{\sigma_v^2}{2\sigma_s^2}\right)^2 + \frac{\sigma_v^2}{\sigma_s^2} + 1} - \sqrt{\left(\frac{\sigma_v^2}{2\sigma_s^2}\right)^2 + \frac{\sigma_v^2}{\sigma_s^2}}$$

if we let $\alpha = \frac{\sigma_v^2}{2\sigma_s^2}$ we can rewrite it as

$$\theta = \sqrt{\alpha^2 + 2\alpha + 1} - \sqrt{\alpha^2 + 2\alpha} \tag{7.4.10}$$

Equation (7.4.10) is always positive since the first square root term is larger than the second. It is also easy to show that the difference is less than one for any $\alpha > 0$. We have thus found a solution $\theta \in (0, 1)$, proving that D_t can be expressed as an invertible MA(1) process by applying proposition 2.1.1 in [4]. The other quadratic solution for θ does not result in an invertible process and can be ignored.

Observe that the ratio between volatility and sampling error determines the size of θ in (7.4.10), and thus the rate at which the importance of older samples diminish compared to more recent samples in equation (7.4.9). When volatility (σ_v) is low compared to sampling error (σ_s), the additional 1 on the left side will dominate the ratio terms, θ will be close to one; the exponential decay will be slow so the best linear predictor will assign almost equal importance to all earlier samples. In the limit, this can be expressed as

$$\theta \to 1 \quad \text{as} \quad \sigma_v/\sigma_s \to 0$$

On the other hand, if σ_v dominates, the ratio terms will be close to 1 and the difference in (7.4.10) will be very small, yielding a θ close to 0. Such a θ will cause only the last sample to be significant in the predictor (7.4.9). Also

$$\theta \to 0 \quad \text{as} \quad \sigma_v/\sigma_s \to \infty$$

The above result is of special importance. For 'volatile' processes one can get a good approximation to the best linear predictor by just taking a small number of past steps into account, since the value of earlier samples decay fast when θ is small. Such a reduction is called the *approximate large sample best linear predictor* $\tilde{P}X_{n+1}$ of X_n which is an approximation to (7.4.9) using only a finite number of previous terms.

Chapter 8

The Markov Decision Process

Models for sequential decision making when outcomes are uncertain are commonly referred to as *Markov Decision Process models (MDP)*. Although a full treatment of this subject is beyond the scope of this text we illustrate the most important principles for our special case. MDP:s have a wide variety of applications ranging from ecology, economics to communications engineering. Our treatment will closely follow Puterman (1994) in [28].

8.1 Definitions

8.1.1 Epochs

We will begin by defining *decision epochs*. Every point in time where we make a decision is called a decision epoch. The discrete set epochs divide time into (time) periods. A decision epoch corresponds to the beginning of a period. We will assume a finite set of epochs, so that

$$\mathcal{T} = \{1, 2, \ldots, N\}$$

An element of $\mathcal{T}$ will be referred to as t as we have done previously when we spoke of time dependence. Since N is finite we have what is called a *finite horizon problem*. For theoretical reasons decisions are not made at decision epoch N. We then have an N-1 period problem.

8.1.2 States

Let S be the set of possible system states. The state of the system at the current epoch is defined as the CPI of the element currently being displayed. S will be taken as *discrete*. This is a central modeling assumption and it means that we are discretizing the CPI (i.e constructing a discrete probability distribution from a continuous one, a procedure which is explained in detail at the end of this chapter).

8.1.3 Decisions and Actions

A decision (or decision rule) is a rule which determines an *action* to be taken. An action in our context is that of selecting a particular element. The set of available actions is in our case the set A consisting of the elements of the pool.

$$A = \{1, \ldots, n\}$$

Every action leads to an outcome. Taking action $a_t = k$ means that we decide at time t to show element k with probability one at time $t + 1$.
Our decision rules will be *history dependent*. Given the actions made and the resulting states *up to time epoch* t we can define the history vector

$$h_t = (h_0, a_0, s_1, \ldots, s_{t-1}, a_{t-1}, s_t)$$

where h_0 represents the prior information on the elements. We also have the representation

$$h_t = (h_{t-1}, a_{t-1}, s_t)$$

where s_t is the system state at epoch t.
Our decisions will be functions defined on the set H_t which is defined as the set of all histories h_t. We write

$$d_t : H_t \to A$$

for a given $h_t \in H_t$, $d_t(h_t)$ is the action we take. The action is the selection of an element in A. Thus $d_t(h_t) \in A$.

8.1.4 Transition Probabilities

We assume that the CPI of each element evolves independently according to a probability model. Conditioned on the history of actions and states we can treat this as a single probability model for the system as a whole. It simply gives the probabilities of different outcomes (here measured in CPI) conditioned upon the history of the system and the action taken by the decision maker. We define this probability as

$$p_t(j \mid h_t, d_t(h_t))$$

which gives the conditional probability of observing the system in state j in the next period if we are currently in time t. Note that h_t represents *everything* we know of the system up to time t. $d_t(h_t)$ is the decision we make. An example might clarify things.

Example Let

$$\mathcal{T} = \{t_1, t_2\}, \quad S = \{\text{High}, \text{Low}\}, \quad A = \{1, 2\}$$

Assume $t = t_1$ and $h_{t_1} = (h_0, 1, Low)$ then

$$p_{t_1}(High \mid h_{t_1}, d_{t_1}(h_{t_1}))$$

is the probability of arriving in state 'High' in period t_2 given that element 1 was selected at t_1 and this resulted in the state 'Low'. The probability depends on the decision d_{t_1}. With $d_{t_1} = 1$ we get the probability of arriving in state 'High' for element 1. Which is one minus the probability of remaining in state 'Low'. For $d_{t_1} = 2$ we on the other hand get the probability of arriving in state 'High' when element 2 is selected.

8.1.5 Policy

A *policy* specifies the decision to be used at each decision epoch. Let Π be the set of all policies, then $\pi \in \Pi$ is defined by

$$\pi = (d_1, d_2, \ldots, d_{N-1})$$

Since each d_t is a function on the set of all possible histories up to time t. π completely specifies the decisions to be made for every possible sequence of states and actions.[1]

8.2 Optimality

8.2.1 Rewards

Before we proceed we need to mention something about *rewards*. Traditionally in MDP formulations a reward function is specified

$$\xi_t\big(h_t, d_t(h_t)\big)$$

where ξ_t is the random reward received at epoch t as a function of the random state of the system h_t and the decision made $d_t(h_t)$. Over a horizon $t = 1, \ldots, N$ we obtain a reward sequence

$$\big\{\xi_1\big(h_1, d_1(h_1)\big), \xi_2(h_2, d_2(h_2)\big), \ldots, \xi_{N-1}\big(h_{N-1}, d_{N-1}(h_{N-1})\big), \xi_N(h_N)\big\}$$

Since we make no decision at epoch N we obtain $\xi_N(h_N)$ at this stage. $\xi_N(h_N)$ is the *terminal reward*. With such a formulation the reward depends on the history h_t as well as the action taken.
Now in our case the current state and the current decision is not sufficient to completely determine the reward. Instead the reward is determined *randomly* by a probability distribution. This distribution is specified by the history h_t and the decision $d_t(h_t)$. We need to know what outcome a decision *leads* to in order to determine the reward. The reasoning is as follows.
Assume the system is in state s at time t. Then we take action a which results in the system being in state j at time $t + 1$. We assign a number to this sequence of events and denote it $\xi(s, a, j)$. Recall that conditioned on an action and a state the probability of the next state is determined by $p(\cdot \mid s, a)$. Thus for each decision (choice) and state (history) we have a probability distribution which determines the likelihood of each outcome j. If we take the expected value with respect to the end-states we obtain

$$\xi_t(s, a) = \sum_{j \in S} \xi(s, a, j) \cdot p_t(j \mid s, a)$$

[1] That is for every $h_N \in H_N$.

This can be seen as 'averaging' over all possible outcomes. Now the reward of action a is simply the state this action leads to. If the action leads to the outcome state j, *this is the reward*, because levels of revenue are taken as states in S. Thus $\xi(s, a, j) = j$. Using this, together with setting $a = d_t(h_t)$ and $s = h_t$ gives us the final expression for the *expected reward* of making decision $d_t(h_t)$

$$\xi_t(h_t, d_t(h_t)) = \sum_{j \in S} j \cdot p_t(j \mid h_t, d_t(h_t)) \tag{8.2.1}$$

Next, we show how to compute the conditional distribution $p_t(j \mid h_t, d_t(h_t))$.

8.3 Prediction

We have now explained how to calculate the reward. In order to be able to calculate the expectation in (8.2.1) we need a probability model for the elements. We begin with an example

Example Recall that the information we have about the system is given by the history vector h_t. Take $S = \{\text{High}, \text{Low}\}$ and assume there are three ad-elements. An outcome of 'High' means here that the banner was shown on the website during a time period and was profitable and generated high revenue. The state 'Low' refers to the opposite of this outcome. Let's say we start in $t = 1$ and stop at $t = 5$. Then depending on what actions are made and what the outcomes are we may observe a sequence of history vectors

$$\begin{aligned}
h_1 &= (h_0, 1, \text{Low}) \\
h_2 &= (h_0, 1, \text{Low}, 2, \text{High}) \\
h_3 &= (h_0, 1, \text{Low}, 2, \text{High}, 1, \text{Low}) \\
h_4 &= (h_0, 1, \text{Low}, 2, \text{High}, 1, \text{Low}, 3, \text{High}) \\
h_5 &= (h_0, 1, \text{Low}, 2, \text{High}, 1, \text{Low}, 3, \text{High}, 3, \text{High})
\end{aligned}$$

such as above. Remember that we make no decision for the last period. From these vectors we can in each period determine the set of distributions $p_t(j \mid h_t, d_t(h_t))$ for each $d_t(h_t) \in \{1, 2, 3\}$ and all $t \in [1, 5]$. Remember that these are calculated sequentially. For $t = 1$ we calculate $p_1(j \mid h_1, d_1(h_1))$ for $d_1(h_1) = 1$, $d_1(h_1) = 2$ and $d_1(h_1) = 3$ for all states j. Then at $t = 2$ $p_2(j \mid h_2, d_2(h_2))$ is calculated for $d_2(h_2) = 1$ and $d_2(h_2) = 2$ and so on. Note that we have more information at $t = 2$ than at $t = 1$.

Discretization of distributions

Our random walk model gives us a predictor which is a continuous normally distributed random variable y_t. However, we have defined our state space S as a discrete set with a finite number of outcomes. We want to translate outcomes of the continuous variable y_t into outcomes falling in the discrete set S. This is done by *discretizing* the normal distribution of y_t.

Let the pdf of y_t be $f(x)$. The *support* of f is defined as

$$\text{supp } f(x) = \{x \mid f(x) > 0\}$$

this is simply the set of x-values for which $f(x)$ takes on positive values. For example, the normal distribution has support on the continuous set $\mathbb{R}$. The idea is to divide the support of f into m number of equally sized subintervals. However, in practice it may be necessary to cut off part of the tails of the distribution. This is to avoid having intervals in the ends of the distribution with very little probability mass attached to them. For example we could instead use the support

$$A' = \{x \mid f(x) > k\}$$

where k is a constant. The horizontal line $y = k$ in the xy-plane specifies two cutoff points where the line crosses the normal pdf. For the case where $f(x)$ is the pdf of the standard normal distribution with mean zero and variance one we simply solve

$$\begin{aligned} f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} = k &\iff -x^2/2 = \ln(k\sqrt{2\pi}) \\ &\iff x^2 = \ln\left(\frac{1}{k^2 2\pi}\right) \end{aligned}$$

thus

$$x = \pm\sqrt{\ln\left(\frac{1}{k^2 2\pi}\right)}$$

The cutoff points are symmetrical around the y-axis. The result is analogous for different parameters of the normal distribution. Now call the solutions $x_1(k)$ and $x_2(k)$ where $x_1(k) = -x_2(k)$. The integral

$$\int_{x_1(k)}^{x_2(k)} f(x)\ dx = 1 - \alpha$$

specifies that we get an interval $[x_1(k), x_2(k)]$ which contains $(1-\alpha)\%$ of the density of the normal distribution. By the symmetry of the distribution these points satisfy

$$\int_{x_2}^{\infty} f(x)\ dx = \frac{\alpha}{2}$$

and

$$\int_{-\infty}^{x_1} f(x)\ dx = \frac{\alpha}{2}$$

These points are tabulated in tables of the standard normal distribution and it is possible to get the points x_1 and x_2 directly from these tables for any value of α. For $1 - \alpha = 0.95$ we get 95% of the distribution and the table gives us the points $x_1 = -1.96$ and $x_2 = 1.96$. The support A' is hence given by the

interval $[-1.96, 1.96] \approx [-2, 2]$. This interval has length 4. Since we are dealing with the standard normal distribution which has a standard deviation of one, this tells us that our interval covers 4 standard deviations of the distribution. We can then divide this interval into for example $m = 10$ number of subintervals. Then each subinterval has length $4/10 = 0.4$ and corresponds to a significant part of the mass of the probability distribution of y. The m intervals we construct will be taken as the state space S. When we observe an outcome of the continuous variable y we then assign it to one of the m intervals, and thus also to one of the *states* of S.
Now that we have a discrete set of states S it remains to specify the discrete probability distribution over these states. For each state $j \in S$ we want to determine $p(j)$. We also require that $\sum_{j \in S} p(j) = 1$ so that $p(\cdot)$ is a valid probability distribution. We know that each state j corresponds to a subinterval of the modified support A' of $f(x)$. Then we determine the area below the curve of the portion of $f(x)$ which corresponds to this interval. The area can be determined by a numerical integration method. We use the *trapezoidal rule*.[2]
If the interval corresponding to state j is $[l, u]$ where l and u is the lower and upper endpoint of the interval respectively we set

$$p(j) = \int_l^u f(x)\ dx \approx \frac{f(l) + f(u)}{2}(u - l)$$

Now we quickly note that

$$\sum_{j \in S} p(j) = \int_{A'} f(x)\ dx \neq 1$$

So in order to have $\sum_{j \in S} p(j) = 1$ we have to *normalize*[3]. Define a new distribution $p'(j)$ by

$$p'(j) = \frac{1}{\sum p(j)} p(j)$$

it is now easily seen that $\sum p'(j) = 1$.

Putting it together $p_t(j \mid h_t, d_t(h_t))$

We have thus established the probability predictor model for an element. When we want to be specific about the exact parameters that determine the model we write $p(j)$ as required by the MDP formulation

$$p_t(j \mid h_t, d_t(h_t))$$

when $d_t(h_t) = k$ this is the probability distribution for element k. Each ad-element has its own distribution with its own parameters. The vector h_t contains the observed outcomes up to time t, data which is necessary for us to specify the parameters of the discretisized normal distributions in the random

[2] For details see [8]

[3] This procedure can be illustrated intuitively as follows. Since for the normal distribution we have supp $f(x) = (-\infty, \infty)$ we have $\int_{-\infty}^{\infty} f(x) = 1$ we take the area we cut off when we constructed A', split it equally, and add one part to each $p(j)$.

walk predictors. The parameters to be established are the expected value and the volatility. We also know that h_t tells us how many steps, say k, that have passed since we last sampled the element. Recall that we multiply the volatility with this number in order to get a random walk predictor k-steps into the future.

8.4 Policy Evaluation

We follow the methodology in [28]. At the beginning of our optimization we assume that we are in a 'prior state'. This prior state specifies starting values for the parameters of the distributions governing each $y_{t,k}$ corresponding to $t = 1$. This is summarized by saying that h_1 is known and specified prior to any calculations. The quantity of interest is the *expected total reward* over the whole horizon

$$v_N^\pi = \mathbf{E}^\pi \left[\sum_{t=1}^{N-1} \xi_t(h_t, d_t(h_t)) + \xi_N(h_N) \right] \tag{8.4.1}$$

with respect to a policy π. Remember that a policy specifies the decisions to be taken for every sequence of history vectors $h_1, h_2, \ldots, h_{N-1}$. We want to find an optimal policy π^* such that

$$v_N^{\pi^*} \geq v_N^\pi$$

for all π. In other words, we want to maximize the expected total reward, which is the expectation of the sum of all reward functions.
To compute the *optimal* policy we must have an algorithm for evaluating the expected total reward for any given policy (optimal or not). We proceed by using a technique similar to that used in Dynamic Programming which uses backward-induction. Define

$$u_t^\pi = \mathbf{E}^\pi \left[\sum_{n=t}^{N-1} \xi_n(h_n, d_n(h_n)) + \xi_N(h_N) \right] \tag{8.4.2}$$

this is the expected total reward from epoch t to N, $t < N$. It is clear that $u_1^\pi = v_N^\pi$ and for $t = N$ we set $u_N^\pi(h_N) = \xi_N(h_N)$ where $h_N = (h_{N-1}, a_{N-1}, s)$ where it is emphasized that s is the 'end-state' and we make no decision at epoch N. The reason we employ a special treatment to the terminal epoch N is to enable us to use the backward-induction method of solution in a straightforward manner. The goal is to use the functions u_t^π to compute v_N^π.

A note on the terminal condition We have throughout the text said that we make no decision at epoch N. This is a boundary condition which enables us to use a 'backwards-calculating' algorithm to evaluate $u_t^\pi(h_t)$.
In the N:th epoch we have $t = N$ and $u_N^\pi(h_N)$ is equal to the terminal reward $\xi_N(h_N)$ where $h_N = (h_{N-1}, a_{N-1}, s_N) \in H_N$. The idea with not making a decision at the last epoch is that we can set

$$\xi_N(h_N) = \xi_N(s_N) = s_N$$

so that we set $u_N^\pi(h_N) = s_N$ for each $h_N = (h_{N-1}, a_{N-1}, s_N) \in H_N$. This means that we can calculate the reward 'backwards' for each and every $s_N \in S$. Or alternatively, for every end-state $s_N \in S$ we calculate all the possible 'paths' leading up to this end-state. Then we calculate the expected value of every such path.

The Finite Horizon-Policy Evaluation Algorithm

1. Set $t = N$ and $u_N^\pi(h_N) = \xi_N(h_N) = s_N$ for all $h_N = (h_{N-1}, a_{N-1}, s_N) \in H_N$
2. If $t = 1$, stop, otherwise go to step 3.
3. Substitute $t-1$ for t and compute $u_t^\pi(h_t)$ for each $h_t = (h_{t-1}, a_{t-1}, s_t) \in H_t$ by

$$u_t^\pi(h_t) = \xi_t(h_t, d_t(h_t)) + \sum_{j \in S} p_t(j \mid h_t, d_t(h_t)) u_{t+1}^\pi(h_t, d_t(h_t), j), \quad (8.4.3)$$

   observe that $(h_t, d_t(h_t), j) = h_{t+1} \in H_{t+1}$
4. Return to 2.

An intuitive way of writing (8.4.3) is

$$u_t^\pi(h_t) = \xi_t(h_t, d_t(h_t)) + \mathbf{E}_{h_t}^\pi \left[u_{t+1}^\pi(h_t, d_t(h_t), X_{t+1}) \right] \quad (8.4.4)$$

$u_t^\pi(h_t)$ is the sum of two components, the reward in epoch t plus the reward of epoch $t+1$ and onwards. We get an immediate reward at epoch t when executing action $d_t(h_t)$. This is denoted by $\xi_t(h_t, d_t(h_t))$. Then we add the expected reward of the forthcoming steps as explained by the expectation term. We study the expectation term in some more detail

$$\mathbf{E}_{h_t}^\pi \left[u_{t+1}^\pi(h_t, d_t(h_t), X_{t+1}) \right] = \sum_{j \in S} p_t(j \mid h_t, d_t(h_t)) u_{t+1}^\pi(h_t, d_t(h_t), j)$$

Every term in the expectation sum is composed of two factors. The first factor is the probability of being in state j at decision epoch $t+1$, if action $d_t(h_t)$ is selected. This factor is multiplied with the expected total reward obtained from period $t+1$ and onward using policy π, *given* that we chose action $d_t(h_t)$ and it resulted in the outcome j. Or in other words, the expected future reward given that $h_{t+1} = (h_t, d_t(h_t), j)$.
The recursion (8.4.3) is the essence of dynamic programming. It enables us to solve multistage problems by analyzing a sequence of simpler recursive single-stage problems. A proof of the fact that the presented algorithm actually gives u_t^π can be found in [28] pp. 83-.

8.5 The Bellman Equations

In this section we show how to determine an optimal policy π^* over the whole horizon. We have seen previously that v_N^π, given by (8.4.1), is computed recursively by computing the functions u_t^π, given by (8.4.2) using the

Finite-Horizon Policy Evaluation Algorithm. Now we are interested in computing the optimal expected total reward over the whole horizon v_N^*. This is done by computing the optimal expected total reward functions from t and onward u_t^*, which are defined as

$$u_t^{\pi^*} = \max_{\pi} u_t^{\pi}(h_t)$$

where the max is taken over all policies $\pi = (d_t, d_{t+1}, \ldots, d_{N-1})$. The above problem thus specifies the policy π^* which is optimal from t and onward. Recall the policy evaluation rule (8.4.3)

$$u_t^{\pi}(h_t) = \xi_t(h_t, d_t(h_t)) + \sum_{j \in S} p_t(j \mid h_t, d_t(h_t)) u_{t+1}^{\pi}(h_t, d_t(h_t), j) \tag{8.5.1}$$

These equations enable us to evaluate any policy π. Denote an action with $a = d_t(h_t)$ to simplify notation, then if we are interested in the *optimal policy over the whole horizon* we have the *optimality equations*

$$u_t^{\pi}(h_t) = \max_{a \in A} \left[\xi_t(h_t, a) + \sum_{j \in S} p_t(j \mid h_t, a) u_{t+1}^{\pi}(h_t, a, j) \right] \tag{8.5.2}$$

for $t = 1, \ldots, N-1$ and the boundary condition

$$u_N(h_N) = \xi_N(h_N)$$

(8.5.2) are called the Bellman Equations in dynamic programming. It simply states that in each stage we should chose the best action (the action with highest expected future reward). If we apply this principle throughout the horizon it can be shown by induction (for a proof see [28] p. 84-) that we arrive at the optimal solution. That is

- $u_t(h_t) = u_t^*(h_t)$ for all $h_t \in H_t$, $t = 1, \ldots, N$ and
- $u_1(h_1) = v_N^*(h_1)$ where h_1 is the prior history

The Optimal Policy Evaluation Algorithm

1. Set $t = N$ and

$$u_N^*(h_N) = s_N \quad \text{for all} \quad s_N \in S$$

2. Substitute $t-1$ for t and compute $u_t^*(h_t)$ *for each* $h_t = (h_{t-1}, a_{t-1}, s_t) \in H_t$ by

$$u_t^*(h_t) = \max_{a \in A} \left[\xi_t(h_t, a) + \sum_{j \in S} p_t(j \mid h_t, a) u_{t+1}^*(h_t, a, j) \right] \tag{8.5.3}$$

and set

$$d_t^*(h_t) = \arg\max_{a \in A} \left[\xi_t(h_t, a) + \sum_{j \in S} p_t(j \mid h_t, a) u_{t+1}^*(h_t, a, j) \right] \tag{8.5.4}$$

3. If $t = 1$ stop. Otherwise return to step 2.

8.6 Examples and Illustrations

Two examples will be presented to illustrate the ideas of the algorithm. Figure 8.1 in section 8.6.2 is also helpful in understanding the basics of the stochastic planning approach.

8.6.1 Two Basic Examples

Two periods (one-step approach)

Let the horizon $N = 2$ and $T = \{1, 2\}$. There are two periods consisting of the initial state (starting state) and a terminal state. Assume we have K ad-elements. Each ad-element is modeled with its predictor (random) model yielding the vector $(y_1, y_2, \ldots, y_K)$ where each y_i follows for example a random walk as shown previously. The statistical properties of each y_i are in this example fixed based on prior history. There are two time periods, so only one decision is made, and there are two rewards. The immediate reward received in period 1 and the expected reward in period 2.

$$\xi_1(h_1, d_1(h_1)) + \mathbf{E}^{\pi}_{h_1}\{X\}$$

where h_1 is the prior information on the distributional parameters. X is the random state of the system in period 2. This is simply the pay-off resulting from the single decision. Let $d_1(h_1) = a$ to simplify notation, note also that since we only make one decision $\pi = d_1(h_1) = a$. This yields

$$\begin{aligned}\xi_1(h_1, a) + \mathbf{E}^{a}_{h_1}\{X\} &= \mathbf{E}_{h_1}(y_a) + \mathbf{E}^{a}_{h_1}\{X\} \\ &= \mathbf{E}_{h_1}(y_a) + \sum_{j \in S} j \cdot p_1(j \mid h_1, a) \\ &= 2\mathbf{E}_{h_1}(y_a)\end{aligned}$$

where $a \in A = \{1, \ldots, K\}$ is just a selection of an element. The subscript h_1 on the expectations is to allow for different distributions of y_a for different histories (or in this case priors). If $a = j$ the expectation of X is simply $\mathbf{E}_{h_1}(y_j)$. If we take the maximum of these two quantities we immediately get that the purely greedy approach is optimal. That is, just selecting the element with the highest expected value. In the next example we will extend our analysis to three periods. In this setting we make *two* decisions. One decision and one *recourse* action. This will clearly illustrate the departure from the greedy approach.

Three periods (two-step approach)

We are now going to illustrate the optimal policy evaluation algorithm with an example. Let $N = 3$. We set $t = 3$ and

$$u^*_3(h_3) = u^*_3(h_2, a_2, s_3) = s_3 \quad \text{for all} \quad s_3 \in S \tag{8.6.1}$$

We again take $S = \{High, Low\}$. Further, suppose 'High'=10 and 'Low'=5. So that $u^*_3(h_2, a, 5) = 5$ and $u^*_3(h_2, a, 10) = 10$. We also remind the reader that the immediate rewards ξ are given by (8.2.1). For example $\xi(h_2, a)$ below is

equal to $\mathbf{E}_{h_2}[y_a]$. Which is the expected value (of the random walk) of ad-element a.
Now set $t = 2$. We must compute (8.5.3)

$$u_2^*(h_2) = \max_{a \in A} \left[\xi_2(h_2, a) + \sum_{j \in S} p_2(j \mid h_2, a) u_3^*(h_2, a, j) \right] \tag{8.6.2}$$

for every $h_2 \in H_2$ where we have used the *fixed* terminal conditions in (8.6.1). This yields

$$\begin{aligned} u_2^*(h_2) &= \max_{a \in A} \big(\xi_2(h_2, a) + p_2(5 \mid h_2, a) u_3^*(h_2, a, 5) + p_2(10 \mid h_2, a) u_3^*(h_2, a, 10) \big) \\ &= \max_{a \in A} \big(\xi_2(h_2, a) + p_2(5 \mid h_2, a) \cdot 5 + p_2(10 \mid h_2, a) \cdot 10 \big) \end{aligned}$$

and

$$d_2^*(h_2) = \arg\max_{a \in A} \big(\xi_2(h_2, a) + p_2(5 \mid h_2, a) \cdot 5 + p_2(10 \mid h_2, a) \cdot 10 \big)$$

We emphasize that $u_2^*(h_2)$ and $d_2^*(h_2)$ must be computed for each and every h_2, that is, each and every possible scenario $h_2 \in H_2$.
Now let $t = t - 1 = 1$ and by (8.5.3) again we have

$$u_1^*(h_1) = \max_{a \in A} \big(p_1(5 \mid h_1, a) u_2^*(h_1, a, 5) + p_1(10 \mid h_1, a) u_2^*(h_1, a, 10) \big)$$

and

$$d_1^*(h_1) = \arg\max_{a \in A} \big(p_1(5 \mid h_1, a) u_2^*(h_1, a, 5) + p_1(10 \mid h_1, a) u_2^*(h_1, a, 10) \big)$$

where both $u_2^*(h_1, a, 10)$ and $u_2^*(h_1, a, 5)$ were computed in the previous step for all h_1 and a since $(h_1, a, j) = h_2$.
Now we have the optimal policy $\pi^* = (d_1^*(h_1), d_2^*(h_2))$. In general we are only interested in the first decision (the one nearest in time) $d_1^*(h_1)$. Usually after the first step, the whole computation is remade with a new predictor p using the new additional data.

8.6.2 The Scenario Tree and Computational Complexity

The problem in each step t depends on every possible history $h_i \in H_t$. This amounts to calculating the 'search tree' of the stochastic planning approach. The size of this tree grows exponentially with the number of time steps one wants to plan ahead. As can be seen in figure 8.1 the tree essentially 'branches out' because the optimal element choice in the next step depends on the outcome resulting from the previous element choice. Thus, since the problem is stochastic one has to specify 'parallel' optimal paths in the decision tree, one for each possible outcome of the stochastic process governing each element (see the two thick arrows leading up to stage 2 in figure 8.1). Remember that we make the outcome spaces of these stochastic processes discrete, in this sense

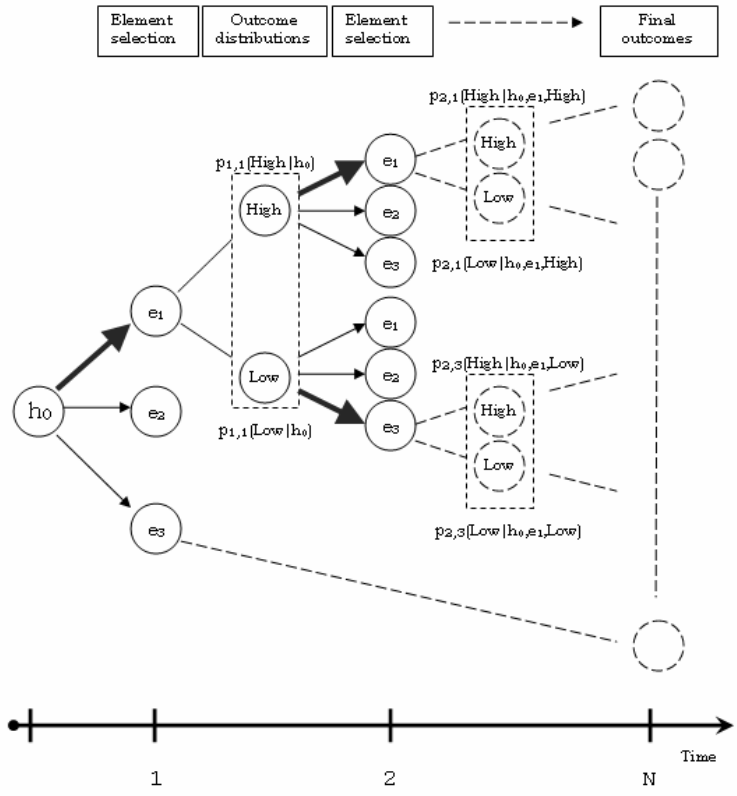

Figure 8.1: A fragment of a hypothetical scenario tree. The wide arrows represent optimal paths conditioned on the previous outcome. The probability distribution for element k at time t is $\big(p_{t,k}(\text{Low} \mid h), p_{t,k}(\text{High} \mid h)\big)$, conditioned on a history $h \in H_t$.

increasing the number of outcomes in the discrete outcome space S requires having more branches emanating from each decision node, increasing the computational complexity. The computational complexity of the stochastic planning approach is thus $\mathcal{O}((e \cdot o)^N)$ where e is the number of elements, o the number of discretized outcomes and N is the length (number of decisions) of the planning horizon. Such problems are generally considered to be computationally *intractable*. As an example, a problem with 5 elements, 10 outcomes and a mere 5 decisions generates a search tree with 312 500 000 nodes. Computing that by brute force is roughly what a modern computer can handle in a reasonable amount of time, which serves as a benchmark for on-line applications.

On a more detailed level, by exploiting symmetry in the search tree or imposing restrictions in the formulation of the stochastic planning problem, the problem can be made easier to compute. These are more methodological concerns and we will not focus on such modifications. Instead, as will be seen, we will approximate the underlying problem on a more general level.

8.7 Extensions

In this section we will briefly discuss some extensions to our theory which are slightly more technical.

8.7.1 Branch and Bound

It is possible to speed up the computations involved in optimizing the total expected reward by using a branch-and-bound rule. This means that some of the branches in the 'decision tree' for which there is no probability of improvement do not have to be computed.
Informally, we first create a 'best-case scenario' random walk predictor which is calculated using the data on all the elements. Let y^M_{t+h} denote the 'best case' random walk predictor h steps into the future starting at time t. Let μ_k be the last observed CPI observation of element k. Then the best possible expected value this element can achieve in h steps is[4]

$$\mu_k + \mathbf{E}\{y^M_{t+h}\}$$

Let

$$\gamma = \arg\max_i \mu_i$$

then if

$$\mu_k + \mathbf{E}\{y^M_{t+h}\} < \mu_\gamma$$

we draw the conclusion that element k is very unlikely to become better than the currently best known element γ within the next h time periods.

Statistical details on the upper bound*

The statistic of interest is called the max statistic.[5] It is defined as

$$y_{\max,t} = \max_k y_{t,k}$$

where $y_{t,k}$ is the random predictor variable of element k at time t. Since $y_{t,k}$ are not identically distributed for $k = 1, \ldots, n$ deriving the cdf is not straightforward. Although they are independent which we use below where we derive the cdf of $y_{\max,t}$.

$$\begin{aligned} F_{y_{\max,t}}(z) = \Pr\{y_{\max,t} \leq z\} &= \Pr\{y_{t,1} \leq z, y_{t,2} \leq z, \ldots, y_{t,n} \leq z\} \\ &= \Pr\{y_{t,1} \leq z\} \cdot \Pr\{y_{t,2} \leq z\} \cdots \Pr\{y_{t,n} \leq z\} \end{aligned}$$

[4]Note that the only difference between the 'best case' predictions for different elements is the starting value μ_k of the random walk

[5]The max statistic is in the literature defined as the maximum of n random variables $X_1, \ldots, X_n$ and is denoted

$$X_{(n)} = \max_k X_k$$

This is referred to as the n:th *order statistic*. See for example [5] for details

We are interested in calculating the discrete probability distribution of $y_{\max,t}$ over S. S defines a set of points along the real line corresponding to the different states. We now want to define the probability mass function over S. We perform the discretisization of $y_{\max,t}$ this time by using the cumulative distribution function (cdf) instead of the pdf[6].
For a state $j \in S$ we now know there is a corresponding interval with lower and upper endpoints l and u respectively. Thus

$$p_t(j) = \Pr\{l < y_{\max,t} < u\} = F_{y_{\max,t}}(u) - F_{y_{\max,t}}(l)$$

We can now calculate the (discrete approximation) of the expected value of $y_{\max,t}$

$$\Pi_t = \sum_{j \in S} j \cdot p_t(j)$$

Let p_k be the discrete probability distribution of y_k and let p_{max} be the distribution of $y_{\max}$. By the definition of the maximum distribution if $a \geq b$ we have

$$p_{\max}(a) \geq p_k(b) \quad \forall k$$

so

$$\Pi_t = \mathbf{E}[y_{\max,t}] = \sum_{j \in S} j \cdot p_{\max}(j) \geq \sum_{j \in S} j \cdot p_k(j) = \mathbf{E}[y_{t,k}] \quad \forall k$$

Thus $y_{\max}$ follows a distribution that is on average greater than or equal to the distributions of the other elements. We will use its expected value Π_t as an *upper bound* on $y_{t,k}$, $k = 1, \ldots, n$. Upper bounds are often used in algorithms of type 'branch and bound' which occur in the optimization literature. The upper bound is in some sense the best possible scenario for a specific strategy. If the upper bound for this strategy is lower than the outcome of some other known strategy it may be wise to reconsider the first strategy. Because, depending on the type of upper bound, the first strategy has little or no chance of outperforming the second strategy.

8.7.2 Bayesian Estimation

Because of the widespread use of Bayesian concepts in the statistical modeling of e-commerce we provide some ideas of how such concepts can be integrated into our analysis. For an in-depth treatment of statistical inference see [5], and for an application of adaptive bayesian updating see [14].
Let $X_1, \ldots, X_n$ be iid Bernoulli(p). Then $Y = \sum X_i$ is binomial(n, p). Recall from the CPI estimation that n is the number of impression used to estimate p which is the CPI estimate. Let the prior distribution of p be beta(α, β). This means that p is our prior belief on the CPI of the element. Upon observing the number of successes $Y = \sum X_i$ we can *update* our prior belief according to bayesian updating. This yields what is called a posterior distribution on p. If we calculate the mean of the posterior distribution we obtain one possible

[6]The cdf $F(x)$ of a random variable X is defined as $F(x) = \Pr\{X < x\}$.

posterior estimate $\mathbf{E}\{p \mid Y\}$. This can in fact easily be calculated since the beta family of distributions is a *conjugate family* of distributions thus the posterior distribution will also be a beta density.
It is also possible to use a bayesian estimation technique when specifying weights. Recall that weights $(w_1, \ldots, w_n)$ satisfy $\sum w_i = 1$ and $w_i \geq 0$. It is therefore natural to interpret the weights as *probabilities*. A bayesian approach is particularly useful if these can be regards as *multinomial probabilities*. This could be useful in e-commerce in the presence of dynamic content. For example suppose there are n distinct objects which generate a desirable 'event' such as a sale of an expensive product. Upon observing the sale, we can observe which object generated the sale and then use this information to update our belief on which objects has the highest probabilities of generating sales. The advantages of the bayesian approach is that we can specify a prior distribution over the multinomial distribution which reflects our *prior belief* in which objects are most likely to generate revenue. Then we can update our belief by deriving the posterior distribution. For multinomial probabilities there exists in fact a computationally tractable distribution. It turns out that having a prior in the dirichlet family of distribution produces a dirichlet posterior with easily interpreted parameters.

8.7.3 Seasonality and Trends in the Predictor

In economics one often tries to model and predict the demand for different goods and services. An integral part of such models is the ability to relate changes in demand to the calendar. For example, some services are requested more frequently during weekdays than during week-ends (take for example the day-to-day variation in the use of public transport). There are a myriad of effects one could argue influence the demand of products. Weather for example should probably have an effect on the sales of umbrellas, since demand for this product is naturally higher if it is raining compared to when the sky is clear. What we are interested in is demand on the internet. What factors influence how people click on banner advertisements? In this section we will only discuss calendar effects, which we call *seasonality*. The models discussed in this chapter are more of a practical nature than a theoretical one, there is no 'correct' mathematical approach so we are left with going for what works in practice. We model the seasonal effects of element k by estimating a seasonal function

$$s_k(t) = s_k^{(1)}(t) + s_k^{(2)}(t) + \cdots + s_k^{(l)}(t)$$

where it is assumed that the seasonal function is composed of l components. In order for the seasonality part of our algorithm to improve performance, a seasonal component of one element must show *greater* signs of seasonal effects than the seasonal component of another element estimated at the same time. Thus seasonal differences can be exploited to make better choices.

Chapter 9

Implementation and Results

We have implemented the algorithm outlined in the previous chapters in MATLAB. This chapter provides an overview of this implementation. We also present test results and a discussion of the methods we have used in the tests.

9.1 Implementation Details

This section will briefly explain the basic principles behind our implementation. MATLAB was chosen as the platform for the implementation, because development is straightforward in this language and MATLAB has many comfortable tools for data analysis. MATLAB also has an extensive cross-linking architecture and can interface directly with Java classes.
In the course of this thesis several programs were developed, some were just small tools for data exploration, others were more comprehensive systems. The implementation consists of three main components:

1. An automated system for information retrieval.

2. The statistical planning algorithm treated in chapter 8.

3. A flexible test bench for running simulations of algorithm performance.

We will describe these components in the order above.

9.1.1 The Information Retrieval System

The information retrieval system was a procedure invoked to run every night through the MATLAB COM server from a piece of JScript in the system scheduler. It then ran SQL queries to capture data that had accumulated on the pools of interest since the last run, and this was saved to disk. The relevant data at TradeDoubler is stored in several very large Oracle databases. To get all the necessary information is a quite complicated process that involves cross-database queries and some custom PL/SQL code. The specifics of this process will not be treated in any detail due to confidentiality.

9.1.2 The Planning Algorithm

The planning algorithm was implemented by recursive function calls, almost identically to the formalization in chapter 8. MATLAB is not ideal for recursion[1], but solving large problems of this type by brute force is computationally intractable and has not been our intention in either case. Methods of decreasing the computational expense will have to be theoretical in nature and not practical. The algorithm was made to be as flexible as possible, allowing us to try out different ways of reducing the computations involved by imposing different theoretical assumptions regarding the problem. The program allows the use of an upper bound (see section (8.7.1), speeding up the branching process. It is also possible to construct approximate solutions by reducing the number of decisions allowed over the horizon (i.e changing the length of the horizon). The simplest case is when the algorithm allows one 'exploration decision' and one decision for 'exploiting the remaining steps' of the horizon. In stochastic programming terminology such a model is referred to as the *one recourse decision* approach. Informally speaking, the freedom to change element at least once during a horizon is the minimum needed to get any exploration at all. This gives an algorithm with the ability to return to the previous element after testing a new one. This one step recourse formulation will be referred to as the *two step approximation* and is fast to compute. In the experiments below, this algorithm will be called `MDP_TWOSTEP`.
As has become apparent from the theoretical treatment, the stochastic planning approach can be employed together with any predictor, tailored to the characteristics of the underlying problem. Therefore the implementation of the predictor is separated into a distinct object from that of the planning algorithm, these objects are only combined during run-time. One of these objects, the predictor class, handles initialization of the probabilistic (statistical) model, incorporating parameter priors and adaptive updating of element rewards, as well as estimation of the volatility parameter during run-time. In essence the predictor class maintains the statistical distribution of each element predictor - the statistical model of uncertainty used in the stochastic planning.

9.1.3 The Data Sources

To achieve modularity the test data is separated from the test process. Currently two *data source objects* are implemented.

- Random Walk + Sampling Error (random data + sampling error)
- The 'Challenge' Data Set (deterministic data + sampling error)

The first data source object is a true random walk with a realistic sampling error added. The second data source object is an interface to a collection of Java test data classes from a recent online 'challenge' competition described below.

[1] Preliminary testing showed that C and Java were at least two factors of 10 faster at recursion.

Random Walk Data Set

The RW data set is simply a random walk assigned to each element which we sample using the binomial sampling (producing an uncertainty in the measured rewards, see section 7.1). It is important to realize that this is the Random Walk Data Source, and not the Random Walk Predictor in our statistical model.
When dealing with stochastic data, such as the RW data set, repeated runs are necessary to get a statistical *sample* of the performance of the algorithm[2]. When identical data sets are needed for such diagnostic purposes, a 'seed' can be passed to the random number generator so the randomization actually gives the same results each time. This is of course something that can only be made in a simulation environment. In reality no two 'dynamical environments' are identical.

The Challenge Data Set

Dynamic optimization problems are in no way new, the internet era has allowed merchants, media outlets and content vendors in general to run complex automated inventory optimizations based on visitor preferences. This has generated an increased interest in the area from the private sector. The purpose of this competition was specifically the dynamic exploration trade-off, which is also the main focus of this thesis. This interest led the PASCAL network [26] to arrange an Exploration vs. Exploitation challenge [27], with the help of private sponsors, as a means of strengthening the research interest on exploration vs. exploitation problems in dynamical environments. We use some of the data classes from this Exploration vs. Exploitation challenge.
This data is actually generated from deterministic functions so we do not really consider many of them as realistic models. However, some of them are built around periodic functions which is interesting from our perspective because they serve as demonstrations of how our algorithms adapts to seasonal variations, even when such variations are not explicitly modeled in the predictor[3]. Since we have the random walk assumption in our predictor we make no effort to exploit or predict the determinism of this data. For deterministic data, if the data would consist of a sine function with an unknown frequency it would be possibly to *a priori* assign a sine function to the predictor and try to estimate the unknown frequency of this function from the data set. In general when dealing with deterministic data of this kind it is possible to get perfect fits by assuming the correct functional form (see section 3.3.5). But an algorithm designed to perform well on sine functions, would generalize terribly to new data sets governed by a completely different function. That is why we have chosen a predictor model with as few assumptions as possible regarding the underlying data generating process, and this is the reason why our algorithm may generalize well to completely different data sets. This has become clear when trying out our algorithm on different functional forms.

[2]The performance of the algorithm over a full horizon is meant here, not to be confused with the need to take repeated samples to reduce the error when measuring rewards.

[3]See section 8.7.3 for a discussion on incorporating seasonal and trend components in the predictor

9.1.4 The Test Bench

The purpose of the test bench is to perform simulations and evaluate the performance of the algorithms in an easy and modular way. In the test bench it is possible to specify the number of simulations to be made and also the length of each time horizon. Results are produced in the form of a matrix with quantities of interest (explained later) as well as graphs. The test bench currently supports implementations of the ε-type algorithms presented in chapter 5 as well as the MDP stochastic planning algorithm of chapter 8.

Definitions

Each data source is responsible for collecting statistics on the algorithm reward, maximum reward, average reward and *efficiency*, which all will be defined shortly. The *algorithm reward* is simply the sum of all the rewards collected by the algorithm over the problem horizon. As such, it is the realization of the expected total reward concept underlying the stochastic planning approach. The *maximum reward* is the best possible reward sequence attainable over the horizon. Of course we can only calculate the maximum reward in computer simulations, since it requires having perfect knowledge of the states of all ad-elements. In a non-simulation environment, such a quantity has to be inferred based on data of previous states, yielding something like a 'hypothetical max'. This is the upper bound on the algorithm reward used in practice[4]. The *average reward* is simply a sum of arithmetic means of the rewards of every element in each period. The average reward represents the total expected reward that would be achieved if elements were chosen at random. Now we are prepared to define the efficiency of an algorithm as

$$\text{efficiency} = \frac{\text{algorithm reward} - \text{average reward}}{\text{maximum reward} - \text{average reward}}.$$

This is a normalized performance measure expressed as a percentage of the distance between the average and the best possible performance. It is a metric built on the observation that an optimization algorithm can only exploit differences in the underlying elements. If the elements are nearly identical the difference between algorithm and average performance will not be very large in absolute terms regardless of the algorithm making the selections. In other words, there must be 'good' and 'bad' elements to distinguish between. The efficiency ratio expresses the performance as a percentage where 0% is equivalent to the performance of an algorithm choosing elements at random and 100% is the performance of an algorithm always picking the best element.

9.2 Test Methodology

In simulations it is important to attempt to capture as realistic scenarios as possible. But simplifications are always necessary and must be carefully stated to produce scientifically interesting simulation results.

[4]The upper bound is a sample of a sum of max-distributed random variables (see 8.7.1). The max-distribution represents the reward from being able to chose among the ad-elements when the reward of every element is known. However the cost of having to explore elements prohibits such information, so the maximum reward is never attainable.

The temporal model of choice for the MDP algorithm is the basic random walk predictor of 7.2 (we stress here again that our implementation is flexible, making it possible to use any predictor with the algorithm). Consequently, for the RW data set, both the underlying data source and the predictor follow a random walk. This may seem dubious at first. However a random walk is just additive stochastic noise, the simplest inherently random non-stationary model. So it is in no way possible for us to closely 'match' our random walk with the random walk in the test data, even though they are both the same type of model.
All tests of the stochastic planning algorithm are run with element distributions discretized into 10 outcomes over ± 3 standard deviations, containing 99.7% of the area of the original continuous distribution. See section 8.3 for a description of this procedure. Empirical experience shows that these settings give good results while still being fast to compute (see section (8.6.2)) where we defined the computational complexity of the problem.

9.3 Test Results

First we demonstrate the element selection strategies of the algorithm in action on the stochastic data source (RW). In figure 9.1 we see that the algorithm will first try every element once and then attempt to stick with the currently best element (although the graph looks a bit cluttered, there is only one element selection for each decision stage on the horizontal axis). The algorithm will frequently try elements it has not sampled in a while, where the volatility has added up[5] to such an extent that it is likely that the element can offer an improvement.
Before we move on to discuss algorithm performance we need to run a test to see how many simulation runs we need to get stable results on the stochastic data source (i.e how large sample we need to establish algorithm performance). We show in figure 9.2 the mean efficiency of the `MDP_TWOSTEP` algorithm (the arithmetic mean of the efficiency from the number of simulation runs needed to get stable results on a stochastic data source) plotted against the number of runs. As can be seen we need about 200 simulations for tests on the stochastic data source to get stable results within 1% of the true value. So when referring to the efficiency of an algorithm on a data source we really mean the mean efficiency on this data source, because one run is not enough to determine the performance of the algorithm[6].

[5]Remember that the predicted element distributions become 'wider' in each step due to the volatility of the random walk, increasing the need for exploration.

[6]Note that the number of runs needed to get a stable efficiency estimate varies between data sources, the deterministic data source requires fewer simulation runs.

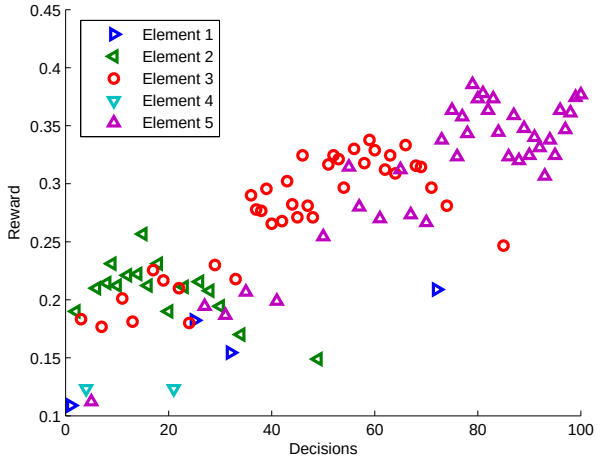

Figure 9.1: Demonstration of the choices of the MDP_TWOSTEP stochastic planning algorithm in a dynamic environment (RW data set). In each time step an element choice is made and the reward is recorded.

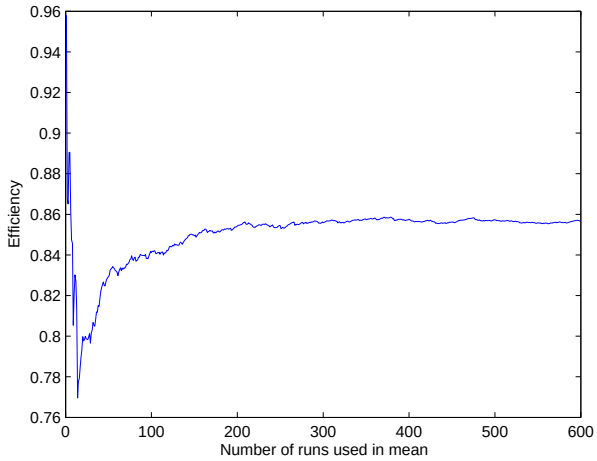

Figure 9.2: Mean efficiency as a function of the number of simulations. We see that the curve stabilizes after approximately 200 simulations.

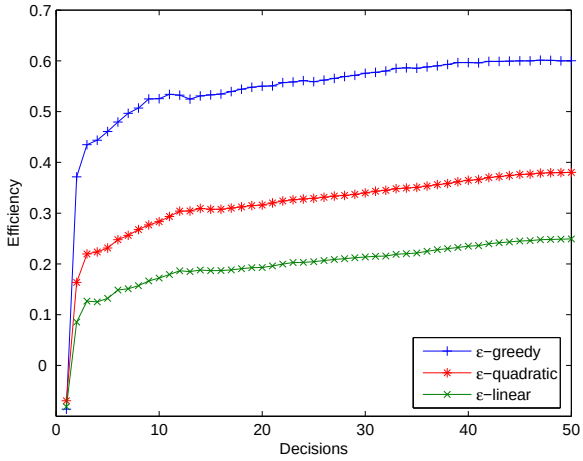

Figure 9.3: Performance of the three different ε-heuristics on the stochastic data source when $\sigma_v = \sigma_s$ (RW data set).

We first compare the members of the ε-heuristic family of algorithms described in section 5.2. This first test is set in a stochastic dynamic environment (RW data set) where the volatility of the (data source) random walk is set to 0.01. The sampling error is also set to this value, so that $\sigma_v = \sigma_s = 0.01$. As a general rule we set the volatility to about 5-10% of the starting value of each data source process.
As expected, the ε-greedy algorithm clearly outperforms the less aggressive alternatives. Based on the theoretical discussion of Chapter 5, and the empirical results here, we conclude that ε-greedy is superior to the other ε-polynomial algorithms and will be used exclusively in the tests against the planning algorithm[7] .
Next we compare the performance of `MDP_TWOSTEP` and ε-greedy on the stochastic data source. Some common parameter choices of ε are tried out, but since ε-algorithms are completely heuristical methods, there are no theoretical guide lines for picking the best choice of ε. An 'optimal' value of ε could be found by exhaustive search of the parameter space, but the amount of simulation required is prohibitory and even so it is disputable if such simulations would reflect realistic conditions[8]. In figure 9.4 we have plotted the performance of `MDP_TWOSTEP` versus five different ε-greedy choices. In the figure, for example 0.3-greedy corresponds to 30% exploration and 70% exploitation according to equation (5.2.2) of section 5.2 . Studying the graph in figure 9.4, we see that there is an 'initial exploration' move and then the curves clearly separate from each other. It also highlights how arbitrary ε-greedy performance can be depending on parameter choice. The superiority of the `MDP_TWOSTEP` algorithm for this data-set is also evident from the graph.

[7]Remember that this is under the assumption that there is no structure (saturation effect) in the weight space. See the end of chapter 4 for a discussion.

[8]Perhaps the choice of ε could be motivated on other grounds, we have discussed that 'optimal' ε values may vary between pools. The most significant factor is probably σ_s, reducing the sample error.

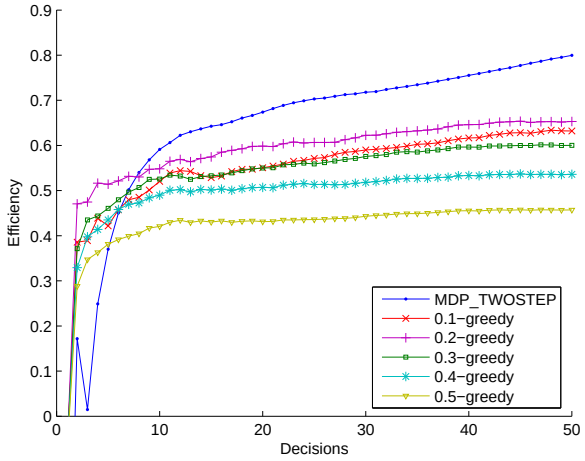

Figure 9.4: Performance comparison between MDP_TWOSTEP and ε-greedy (RW data set) with $\sigma_v = \sigma_s$

In figure 9.5 we have demonstrated the effect of different *prior* volatility estimates on the performance of `MDP_TWOSTEP`. As can be seen a prior that is twice as large as the true volatility of the underlying data-set, impacts performance negatively initially, but it adapts with time (according to equation (7.3.3) which 'smooths' the prior volatility together with observed volatility). A twice as large prior volatility than the true value must be regarded as significantly 'off-the-mark' so we conclude that `MDP_TWOSTEP` is quite robust with regard to prior volatility specification.
Finally we use the 'challenge' data source. As figure 9.6 demonstrates we get favorable results also on this data set. Little effort was made at constructing a good prior estimate on volatility. This demonstrates that the algorithm generalizes well beyond our data set since the performance advantage is about equal to the other tests. This generality is the power of merely assuming a random walk model.

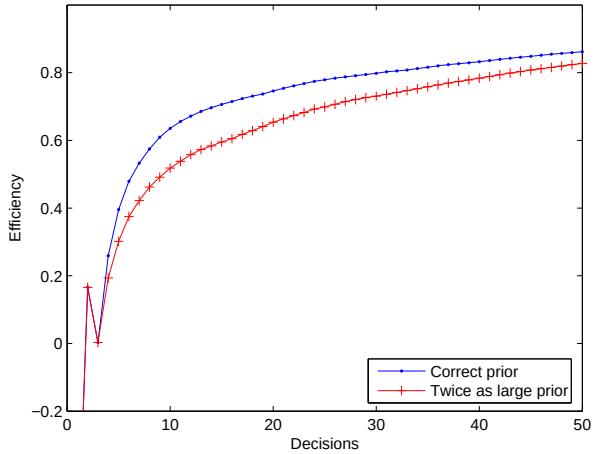

Figure 9.5: Performance of the MDP_TWOSTEP algorithm with different estimates for prior volatility, $\sigma_v = \sigma_s$ and $\sigma_v = 2\sigma_s$ respectively (RW data set).

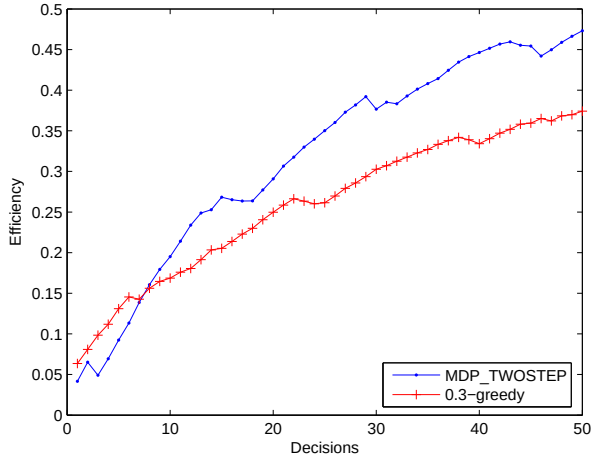

Figure 9.6: 100 simulated decisions on the *weekly variation challenge data set* consisting of periodic functions. MDP_TWOSTEP is compared with 0.3-greedy.

Chapter 10

Concluding Remarks

This concluding chapter contains two sections, giving a discussion of the results as well as a section providing an outline for future work.

10.1 Discussion

We saw that the two step MDP approximation performed very well in dynamic environments. This is comforting in view of the computational complexity argument of section 8.6.2 where we concluded that the full solution, modeling every decision over the whole time horizon, is computationally intractable. Performance degraded slightly when the initial guess of volatility was wrong, but as this estimate adapts over time, accuracy is re-established. In practice the prior volatility can often be estimated fairly accurately from the context, for instance in our application, other pools on the same web site or a web site with similar features can be used.
We made an assumption that the volatility is identical for all ad-elements. This is of course not true in practice, but as long as the elements are similar enough this simplification is motivated because the joint data on all the elements can be used together to construct a common volatility estimate, which speeds up the convergence of the estimator.
It seems reasonable to assume that changes in visitor preferences are largely due to sites attracting new groups of visitors rather than the attractiveness of an element changing suddenly in the eyes of the current audience. For example, a media outlet only needs to add a sensational news item to attract completely new groups with preferences aligned with the new article subject. In addition, different user groups allegedly have different internet usage habits, leading to preferences changing in time as content is altered and as a result, the composition of the audience is changed. This suggests that volatility is site- or pool specific, rather than element-specific, justifying a shared volatility estimate.
The overall results of the two step MDP algorithm are superior compared to the ε-greedy and even more so to the other kind of ε-heuristics described in chapter 5. In the absence of sampling error, performance is about as close to the theoretical max as could be expected. When sampling error is introduced, for example the case when using a sample error σ_s of equal magnitude to the

volatility ($\sigma_s = \sigma_v$), performance degrades but is still satisfactory.
Our hypothesis is that after an element has been explored, in the absence of sampling error, we only need one step of recourse for the remaining time horizon to get an algorithm close to optimal. Informally, this reduces to 'trying out' different elements. We select an element, decide if it was a good selection, then we have a chance to revise our decision. We suspect that the reason a mere one step recourse approximation may be sufficient is because the volatility that accumulates on each element which is not currently explored (i.e shown on the website), can be dispelled by exploring an element only once. If a subset of elements have just been 'tried out' in this manner, and the postulated (underlying) processes governing each element are identical, there is no reason to assume that one element on average would perform better than any other element within this subset. To summarize, The exploration vs. exploitation trade-off in the dynamic case can thus be approximated by one initial explorative phase and one exploitative phase for the remainder of a specified time horizon then a sequence of such planning problems are iteratively solved to span a desired real-time horizon.
In the more general problem, and more realistic setting with *both* volatility (a dynamic environment) and sampling error (measuring reward + noise), we think that a close approximation to the optimal solution is still possible with a small number of decisions. We will now present an informal argument of this. The presence of volatility gives preference to more recent samples since older samples become outdated. We proved this in section 7.4. We showed that it is possible to get an approximative prediction of element reward by only looking at the last N outcomes, where the accuracy of the approximation depended on the ratio of volatility to sampling error. As the sample error grows, more samples are needed to get a good estimate of the reward. But as the volatility increases, the value of additional samples further back in time decreases and consequently, with relatively high volatility compared to sampling error, the need to take several samples to estimate the reward diminishes, and in the extreme it is only motivated to give attention to the most recent sample.
To further illuminate these central facets of the dynamical environment consider the following example. Suppose we have a pool consisting of n elements and we consider a horizon of n steps. Assume also that prior to the experiment all elements have the same expected reward equal to zero, and also that they evolve according to identical stochastic processes starting at zero (the dynamic reward setting). During this horizon we explore each element once giving n explorative moves and zero exploitation moves. We make the following theoretical construct. The starting point is a uniform ordering that will be denoted $\mathcal{G}$. This is the ordering at the starting point, where each element has expected reward equal to zero, so no element is better than any other. Then we specify the ordering $\mathcal{G}_t$ of the elements with respect to the expected rewards observed at time t, which is an *update* of the ordering at time $t-1$, resulting from the explorative move made at time $t-1$. Subsequently as new explorations are made we update the orderings accordingly and obtain a sequence of orderings $(\mathcal{G}_t, \mathcal{G}_{t+1}, \ldots, \mathcal{G}_{t+n-1})$. Note that the ordering $\mathcal{G}_{t+k}$ is just a refinement of the ordering $\mathcal{G}_{t+k-1}$ with the addition of the information contained in the explorative move at time $t+k$. A question we now ask is, how long can these refinements be justified if we only want to model the decision at time t? Due to the aforementioned effect

volatility has on the predictor, as k increases the ordering $\mathcal{G}_{t+k}$ will be less and less dependent on the initial ordering $\mathcal{G}_t$.
The essence of the example is that comparisons between elements in a dynamic setting is bound to become outdated by the nature of the exploration vs exploitation trade-off. When orderings of this type are used sequentially in a planning algorithm, the value of increasing the decision horizon is limited, because the decisions made in nodes deeper into the decision tree will be made on the basis of orderings which have become obsolete. That is, decisions in the tree that are sufficiently separated in time, will be close to independent. This contradicts the whole concept of a planning algorithm, which relies on the concept of decisions 'branching out' in a highly inter-dependent fashion. With this insight we can justify the approximation with a small number of steps in the dynamical setting. This is in our opinion a rationale for the existence of a tractable close approximation to the optimal solution of the dynamic exploration vs. exploitation problem and gives the `MDP_TWOSTEP` solution theoretical substance. The best performance that can 'possibly' be achieved when optimizing with respect to elements that are volatile is, due to the increased uncertainty, lower than that which is achieved when optimizing with respect to static element reward distributions in the classic bandit problem. On the other hand, if one accepts our informal argument above, in the volatile case, optimality could allegedly (at least approximately) be achieved using a small number of steps in the planning algorithm.

10.2 Future Work

In our treatment we have aggregated all visitor behavior and as a consequence all visitors are treated the same. A differentiation of visitors and the separation of users into groups with a separate model for each group is highly desirable from a practical point of view. This is usually called 'targeting'. One form of targeting, is 'geographical targeting' or 'geotargeting' for short. This type of targeting is widely used by websites today where they differentiate between users according to their geographical location and show content with special relevance to their geographic area. An adaptive algorithm which incorporates such concepts would be highly interesting. In connection to this we mention that as soon as users are divided into groups, separated by significant characteristics of their *behavior*, models may have to be estimated separately for each group, decreasing the sample size in all model estimation procedures. To resolve such an issue one would have to exploit similarities (correlation) between different groups, thus entering the multivariate domain, yielding an even more complex exploration vs. exploitation relationship.
It should be noted that the predictor is not restricted to a random walk. The predictor can be improved by tailoring it to a more predictable environment to exploit any underlying structure which may be present. Finding a better model of the environment can be done by for example time series analysis. This is one of the main strengths of the stochastic planning approach. In the presence of more data (and of different type), the characterization of different 'shocks' would be interesting. Given such a shock or 'event', will this event affect the reward distributions of all ad-elements equally? or will this shock affect one element in a certain direction more than other elements? If this is the case, given an adequate model, it is possible to differentiate between the elements and exploit the differentiation by making event-related decisions. Estimation of seasonal components is one way of doing this, but any type of behavioral model could be applied.
A formal examination of the approximations of optimality we have made in this text would also be of interest, both because it is theoretically interesting and the potential it could have in practical applications. An obvious extension to our work is to extend both the `MDP_TWOSTEP` and the ε-greedy heuristic with the prediction theory in section 7.4 and investigate the performance with regard to errors in sampling. Attempts could also be made at better tuning the parameter of the ε-greedy heuristic. It would also be of practical interest to see a more exhaustive attempt to model the potential 'saturation' effects we dispensed with in chapter 5 due to lack of relevant data.

Bibliography

[1] Allen, W.I., Huang, D. , Notz, T.T. and Zeng, N. (2005), *Global Optimization of Stochastic Black-Box Systems via Sequential Kriging Meta-Models*, Journal of Global Optimimization 34(3):441-466

[2] Berry, D.A, B. Fristedt *Bandit Problems* Chapman and Hall, 1985

[3] Brooks, C.H., Gazzale, R.S., Das, R., Kephart, J.O., MacKie-Mason, J.K., Durfee, E.H *Model Selection in an Information Economy: Choosing What to Learn* Computational Intelligence 18(4), pp. 566-582, (2002)

[4] Brockwell, P. J. , Davis, R. A. *Introduction to Time Series and Forecasting* Springer Series in Statistics, Springer (2002)

[5] Casella, Berger *Statistical Inference* Duxbury Advanced Series, Duxbury (2002)

[6] Cristianini, N. , Shawe-Taylor, J. *An Introduction to Support Vector Machines and other Kernel-based Learning Methods* Cambridge University Press, (2000).

[7] Duda, R. O. , Hart, P. E. , Stork, D. G. *Pattern Classification* Wiley-Interscience; 2nd edition (2000)

[8] Eldén, L., Wittmeyer-Koch, L. *Numeriska Beräkningar, En introduktion* , Studentlitteratur (2001)

[9] Eldén, L. (2007): *Matrix Methods in Data Mining and Pattern Recognition*, SIAM, Philadelphia

[10] Flannery, B. P , Press, W. H. , Teukolsy, S. A. Vetterinling, W. T. *Numerical Recipes in C: The Art of Scientific Computing* Cambridge University Press; 2 edition (1992)

[11] Friedman, J., Hastie, T. , Tibshirani, R. *The Elements of Statistical Learning* Springer Series in Statistics, Springer (2001)

[12] Gut, Allan *An intermediate course in Probability Theory* , Springer (1995)

[13] Johnson, R. A., Wichern, D. W. *Applied Multivariate Statistical Analysis* Prentice-Hall; 5th edition (2002)

[14] Jensen, F. V. *Bayesian Networks and Decision Graphs*, Springer (2001)

[15] Jones, D.R., Schonlau, M., and Welch, W.J. (1998), *Efficient global optimization of expensive black-box functions*, Journal of Global Optimization 13, pp.455-492.

[16] Jones, D.R. *A Taxonomy of Global Optimization Methods Based on Response Surfaces* Journal of Global Optimization 21: 345-383, (2001)

[17] Kaelbling, L. P. , Littman, M. L. , Moore, A. W. *Reinforcement Learning: A Survey* Journal of Artificial Intelligence Research, Vol 4, 237-285, (1996)

[18] Liu, B. Hsu, W., Ma, Y. M. *"Integrating classification and association rule mining."* KDD-98, pp. 80-86, (1998).

[19] Lophaven, S.N. , Nielsen, H.B. , Søndergaard, J. *DACE - A Matlab Kriging Toolbox, Version 2.0*. Report IMM-REP-2002-12, Informatics and Mathematical Modelling, DTU. (2002).

[20] Lophaven, S.N. , Nielsen, H.B. , Søndergaard, J., *Aspects of the Matlab Toolbox DACE*. Report IMM-REP-2002-13, Informatics and Mathematical Modelling, DTU. (2002)

[21] Luenberger, D. G. *Investments*, Oxford University Press (1998)

[22] Lundgren, J. Rönnqvist, M. Värbrand, P. *Optimeringslära*, Studentlitteratur (2003)

[23] Muller, K.R. , Mika, S. Ratsch, G. , Tsuda, K. , Scholkopf, B. . *An introduction to kernelbased learning algorithms* IEEE Transactions on Neural Networks 12(2), pp. 181-201, (2001)

[24] Nocedal, J. , Wright, S. J. *Numerical Optimization*, Springer Series in Operations Research, Springer (1999)

[25] Notz, W., Santner, T. J. Williams, B. J. *The Design and Analysis of Computer Experiments* Springer (2003)

[26] Pattern Analysis, Statistical modelling and Computational Learning Network *The PASCAL Network*, Nov. 2006; http://www.pascal-network.org/

[27] Pattern Analysis, Statistical modelling and Computational Learning Network *Exploration vs. Exploitation Challenge*, Nov. 2006; http://www.pascal-network.org/Challenges/EEC/

[28] Puterman, M. L. *Markov Decision Processes: Discrete Dynamic Stochastic Programming* Wiley Series in Probability and Mathematical Statistics, John Wiley & Sons (1994)

[29] Sutton, R. S., Barto, A. G. *Reinforcement Learning* MIT Press; (1998)

[30] Thisted, R. A. *Elements of Statistical Computing: Numerical Computation* Chapman & Hall/CRC (1988)

[31] Thrun, S. , *The Role of Exploration in Learning Control*, Handbook for Intelligent Control: Neural, Fuzzy and Adaptive Approaches, Van Nostrand Reinhold, Florence, Kentucky, (1992).

[32] Wittenmark, B. *Adaptive dual control methods: An overview.* 5th IFAC Symposium on Adaptive Systems in Control and Signal Processing, pp. 67-73, (1995)

[33] Wolpert, D, Macready, W.G. *No free lunch theorems for optimization* IEEE Trans. Evolutionary Computation 1(1), pp. 67-82, (1997)

[34] Siegmund, D. *Herbert Robbins and Sequential Analysis: Invited paper* Annals of Statistics 31(2), pp. 349-365, (2003)

Appendix A

The Least Squares and Generalized Least Squares Problem

We will in this appendix derive the solutions to some least squares problems. In the *multiple regression problem* we have

$$y = F\beta + e \tag{A.0.1}$$

where y is an m-vector of outcomes. F is a deterministic design matrix, and β is a vector of parameters. e is the random (error) component, specifically we assume

$$E[e_i] = 0, \qquad E[e_i e_j] = \sigma^2 R_{ij}$$

or in vector notation

$$E[e] = 0, \qquad E[ee^T] = \sigma^2 R$$

A.1 The Ordinary Least Squares Problem

When $R = I$ then $E[ee^T] = \sigma^2 I$, R is a diagonal matrix with identical entries σ^2 i.e $E[e_i e_j] = 0$ when $i \neq j$ and $E[e_i e_j] = \sigma^2$ for $i = j$. We want to find the vector of parameters β which produces the 'smallest' error in (A.0.1). The solution depends on how me measure this error. When measuring distances in vector spaces a 'norm' is used, here we use the Euclidian norm denoted $\|\cdot\|_2$, so if we want to express the length of the error vector we write $\|y - F\beta\|_2 = \|e\|_2 = \sum e_i^2$ and when this quantity is minimized it produces the Least Squares Problem

$$\min_{\beta} \|y - F\beta\|_2$$

The solution to this problem can be seen geometrically, the distance is minimized when the vector e is orthogonal to the plane spanned by the columns of F, span(F). Thus

$$e^T f_{(j)} = 0, \quad j = 1, \ldots, n$$

or

$$e^T(f_{(1)} \quad f_{(2)} \quad \cdots \quad f_{(n)}) = e^T F = 0$$

where $f_{(j)}$ denotes the j:th column vector of F. Since $e = y - F\beta$ this is equivalent to

$$\begin{aligned}(y - F\beta)^T F &= (y^T - \beta^T F^T)F \\ &= y^T F - \beta^T F^T F = 0\end{aligned}$$

and by taking the transpose of both sides we obtain what is called the *normal equations*

$$F^T y = F^T F\beta \tag{A.1.1}$$

yielding the solution

$$\beta^* = (F^T F)^{-1} F^T Y$$

where we have substituted y with its uppercase counterpart, by which we mean the *random* variable Y. Hence, β^* will also be a random variable and we call it a Least Squares *estimator*.[9]

A.2 The Weighted Least Squares Problem

Now assume that the errors are still uncorrelated but have different variance, i.e $E[e_i e_i] = \sigma_i^2$ and $E[e_i e_j] = 0 \quad i \neq j$. Then since $E[e_i e_j] = \sigma^2 R_{ij}$, R must be

$$R = \text{diag}(\frac{\sigma_1^2}{\sigma^2}, \ldots, \frac{\sigma_n^2}{\sigma^2})$$

the plan is to apply a transformation and reduce the problem to the simpler case treated previously. The inverse of R is easily seen to be $R^{-1} = \text{diag}(\frac{\sigma^2}{\sigma_1^2}, \ldots, \frac{\sigma^2}{\sigma_n^2})$ so we introduce

$$W = \text{diag}(\frac{\sigma}{\sigma_1}, \ldots, \frac{\sigma}{\sigma_n})$$

so that $W^2 = R^{-1}$. Consider now the *weighted* observations

$$\tilde{Y} = WY = W(F\beta + e) = WF\beta + \tilde{e}$$

Then we have $\mathbf{E}[\tilde{e}] = W\mathbf{E}[e] = 0$ and

$$\begin{aligned}\mathbf{E}[\tilde{e}\tilde{e}^T] &= \mathbf{E}[(We)(We)^T] \\ &= W\mathbf{E}[ee^T]W^T = W\sigma^2 RW^T = WW^T R\sigma^2 = \sigma^2 I\end{aligned}$$

since R is diagonal, $WW^T = W^2 = R^{-1}$ and W is just a deterministic matrix that can be moved outside of the expectation operator. The transformed set of observations thus satisfies the assumptions behind the solution (A.1.1) and by replacing Y and F with WY and WF respectively we get the *weighted normal equations*

$$(F^T R^{-1} F)\beta = F^T R^{-1} Y \tag{A.2.1}$$

[20], [30] chapter 3.

A.3 The Generalized Least Squares Problem

Finally, we deal with the completely general case where R is any correlation matrix (not necessarily diagonal). Then the strategy is to diagonalize it by using the fact that R is symmetric and positively definite. Following that of [19] pp 2-4 and [30] chapter 3, R can be factorized according to the Cholesky factorization as

$$R = LL^T$$

where L is a lower-triangular matrix. and if we proceed as the previous section and find

$$R^{-1} = L^{-T}L^{-1} = L^{-1}L^{-T}$$

since the inverse of a symmetric matrix is symmetric. So we again let $W^2 = R^{-1}$ and thus $W = L^{-1}$ and use this matrix in (A.2.1) to solve this generalized problem.

Appendix B

Data aggregation and time aspects in optimization

A very important facet of the problem is that Exploration is *costly*, and the time spent on exploration should be put in relation to the time we have to exploit the rewards. When attempting to solve the trade-off one must then consider the total time we have for optimization, the *time horizon* of the problem.
It is clear that we take actions at points in time

$$t = t_0, t_1, t_2, \ldots, t_{N-1}$$

These points will in general be unequally spaced. The distance in time between two consecutive points k and $k+1$ is

$$|t_{k+1} - t_k| = L_k$$

where L_k is the length of interval k. Due to technical considerations we may require that

$$L_k \geq \underline{L} \tag{B.0.1}$$

where $\underline{L}$ is the minimal spacing between the sample points. This may be due to limitations in how often we can interact with the system which supplies us with revenue data.[1]
What is important for us is how many impressions occur in each interval. So we want to choose the points t_k such that a fixed number of impressions occur in each interval (t_{k+1}, t_k). If we show one element at a time we introduce a mapping function

$$\mathcal{S}\big[(t_{k+1}, t_k)\big]$$

$\mathcal{S}$ is the number of impressions during the interval (t_{k+1}, t_k). If we require that the number of impressions should be m for all time intervals we would have to set $t_0, t_1, ..., t_{n-1}$ such that the practical bound (B.0.1) is satisfied and so that

$$\mathcal{S}\big[(t_{k+1}, t_k)\big] = m \qquad k = 0, 1, \ldots, N-1$$

where m is chosen sufficiently large thus validating the approximation in (7.1.1).

[1] This may be due to bandwidth limitations in an Oracle Database environment.

B.0.1 Scaling

In the implementation we make a distinction between *stages* and *actual time steps*. It is possible to scale every stage involved in the algorithm to encompass several time steps. In the MDP formulation we have said that decisions are made at points in time $t = t_0, t_1, ..t_{N-1}$. Each t_k represents a *decision* stage. In our implementation it is possible to have a completely general framework regarding how many *time* 'tics' that should pass between decisions.