HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Engineering Physics and Mathematics

Ilkka Kudjoi

Applying the Gittins Index to Scheduling of a Queueing System

Master's thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology

Espoo, 28th November 2007

Supervisor: Professor Esko Valkeila
Instructor: Ph.D. Samuli Aalto

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Engineering Physics and Mathematics

ABSTRACT OF MASTER'S THESIS

Author:	Ilkka Kudjoi
Department:	Department of Engineering Physics and Mathematics
Major subject:	Mathematics
Minor subject:	Computer and Information science
Title:	Applying the Gittins Index to Scheduling of a Queueing System
Title in finnish:	Gittinsin indeksin soveltaminen jonosysteemin skedulointiin
Chair:	Mat-1 Mathematics
Supervisor:	Professor Esko Valkeila
Instructor:	Ph.D. Samuli Aalto

Abstract: Multi-armed bandit is a known machine learning problem, in which there are n machines, which yield a random reward once they are allocated one at a time. The question assigned to the system is, how to find an optimal allocation order for the bandits, so that the system would produce best possible reward stream.

Gittins index is recognised as a powerful solution of the problem, but it is not that widely known that the index can be also used to schedule queueing systems. There exists various publications on the topic, but the common factor for most of them is that they are not too easy to follow, especially the original proof by the father of the Gittins index, J.C. Gittins, is challenging.

The motivation of this thesis is therefore to study the relevant publications and to assemble a summarising work on this topic that would be easier to follow. This is achieved by providing introduction to the relevant basic theory related to the topic and to the Gittins index. Further the focus is placed on the generalisation of the index for scheduling of a one server queueing system by simplifying and picking up the most relevant parts of the proof by Gittins to this thesis.

In practise the thesis shows that the Gittins index minimises the expected weighted flow-time (EWFT) of an M/G/1-queueing system, but there is no question about whether or not the index couldn't be generalised further to make also other types of schedulers more effective. However, the further generalisations is to be covered by further research.

Pages: 81+vi

Keywords: Gittins index, queueing systems, Markov decision processes, multi-armed bandit, scheduling

Department fills

Approved:

Library code:

TEKNILLINEN KORKEAKOULU
Teknillisen fysiikan ja matematiikan osasto

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Ilkka Kudjoi
Osasto:	Teknillisen fysiikan ja matematiikan osasto
Pääaine:	Matematiikka
Sivuaine:	Informaatiotekniikka
Työn nimi:	Gittinsin indeksin soveltaminen jonosysteemin skedulointiin
Title in English:	Applying the Gittins Index to Scheduling of a Queueing System
Professuurin koodi ja nimi:	Mat-1 Matematiikka
Työn valvoja:	Professori Esko Valkeila
Työn ohjaaja:	FT Samuli Aalto

Tiivistelmä: Monikätinen rosvo on tunnettu koneoppimisen ongelma, joka koostuu n:stä koneesta, jotka tuottavat satunnaisen tuoton, kun niitä käytetään yksi kerrallaan. Systeemiin liitetty ongelma on, missä järjestyksessä ja miten koneita tulisi käyttää, jotta niistä saataisiin mahdollisimman hyvä tuottovirta.

Gittinsin indeksi on osoittautunut tehokkaaksi ratkaisuksi monikätisen rosvon ongelmaan, mutta ei ole niinkään yleisesti tunnettua, että indeksiä voidaan käyttää myös jonosysteemien skedulointiin. Aiheeseen liittyen on tehty useita eri julkaisuja, mutta useimmille niistä julkaisuista yhteinen tekijä on niiden kehno seurattavuus. Erityisesti J.C. Gittinsin alkuperäisen todistuksen seuraaminen on haastavaa.

Sen tähden tämän diplomityön tavoitteena onkin tutkia aiheeseen liittyviä julkaisuja ja nivoa niistä yhteen kirjallinen tutkielma, jota olisi lähteitä helpompi seurata. Tavoite saavutetaan työssä tarjoamalla ensin lukijalle johdanto aiheeseen liittyvään perusteoriaan ja Gittinsin indeksiin, kun taas myöhemmin työssä keskitytään Gittinsin indeksin yleistykseen yhden palvelijan jonosysteemeille selventäen ja poimien oleellisimpia asioita Gittinsin omasta todistuksesta.

Tämä diplomityö siis osoittaa, että Gittinsin indeksi todella minimoi saapuvien töiden painotetun viiveen odotusarvon yhden palvelijan M/G/1-jonosysteemissä, mutta on selvää, että indeksiä voitaisiin yleistää edelleen muihinkin jonosysteemeihin, tehden niiden skeduloinnista tehokkaampaa. Tämä osuus jää kuitenkin jatkotutkimuksen varaan.

Sivumäärä: 81+vi	Avainsanat: Gittinsin indeksi, jonojärjestelmät, markoviset päätösprosessit, monikätiset rosvot, skedulointi

Täytetään osastolla

Hyväksytty: Kirjasto:

Acknowledgements

I wrote this mathematics Master's thesis as a full-time research assistant in the Networking laboratory at the Helsinki University of Technology under kind, professional and advisory instruction of Ph.D. Samuli Aalto. My assignment started in March 2007 and the work was finalised by the end of the year.

Besides the instructor Samuli Aalto I owe much to my supervisor, professor Esko Valkeila of the Institute of Mathematics for his kind and flexible supervision and my boss Jorma Virtamo, who has given me lots of valuable and experienced advice.

In addition I would like to thank my colleagues for their friendliness and support, especially Ph.D. Aleksi Penttinen for sharing his workroom, his support and good sense of humour that made the working hours more enjoyable.

This work was funded by project Fancy – Flow-Aware Networking: Applications and Analysis by the Finnish Academy of which I am most thankful.

Finally, I naturally would like to thank my family and friends for their support and pleasant leisure time that helped me to relax and keep my mind off the writing process outside the working hours.

Espoo, 28th November 2007

Ilkka Kudjoi

Contents

Chapter 1

Introduction

The decision whether or not to play a slot machine, a simple machine that allows one to play a game that yields a random reward against a small payment, is simple. The gambler has only two alternatives: To play or not to play, as the gambler has no further means of influencing the playing process. In the real world the slot machines often return less than 100% of the money that is inserted in the machine, and thus they may well leave the gambler penniless. Therefore slot machines are often regarded as one-armed bandits.

The question whether or not to gamble with a slot machine isn't very interesting, but if there are n machines, whose characteristics are unknown, the problem becomes more intriguing. The usual setting includes also the assumption that only one bandit is gambled at a time, so the problem is to find the best gambling schedule of n one-armed bandits to maximise the pay-off of the bandits. This system of slot machines is regarded as the *multi-armed bandit* (MAB) [19, 14, 6].

The multi-armed bandit problem was recognised widely after the Second World War and was for a relatively long time a difficult task to solve [6, p. ix]. Gittins proposed an index solution as early as in late sixties [6, p. ix], but his solution achieved recognition and popularity first over a decade later. The Gittins indices are evaluated for every bandit process and the scheduling rule is inferred from the index values by allocating always the bandit having the highest index value.

The index solution is astonishingly simple when compared to the traditional approaches, since the index values are evaluated by means of one single bandit at a time and the index of a bandit is essentially the expected constant reward rate of the bandit, i.e. the rate of reward that a equivalently attractive dummy bandit producing constant reward would produce.

Surprisingly, the Gittins index can be applied to many other problems similar to the multi-armed bandit system. In this work special attention is paid on scheduling a one-server queueing system with the Gittins index. The queueing system of interest is a so called *M/G/1-queue*, a queueing system that has one server which handles the customers or jobs that are arriving in the system according to a Poisson process. The distribution of the service times is general.

The number of bandits in a MAB system is fixed, but the number of jobs in a M/G/1-queue varies through time. Nevertheless, they are naturally jobs that are identified as bandits, when the index is applied to the queueing system, but this identification cannot be done at once, because the queueing system is more complex than a multi-armed bandit. The index theorem has to be generalised to handle systems that have arrivals and precedence constraints. The main motivation of this Master's Thesis is to make this generalisation understandable for a reader who isn't too familiar with the topic.

Besides on concentrating the understandability of the index this thesis was further aimed to show that the Gittins index indeed is an effective way to generate policies for multi-armed bandit systems and M/G/1-queueing systems. This thesis also includes an experimental part that compares the Gittins index to the traditional, iterative methods of creating optimal policies for MAB systems.

This work is structured into seven chapters. The introduction outlines the thesis and acts as chapter one, and it is followed by a chapter that introduces the reader to Markov decision processes. The Gittins index on bandit systems is divided into two chapters, of which the third chapter handles traditional Markov bandit processes and the fourth chapter observes the semi-Markov bandit processes. The definitions of the index provided by these chapters are essentially different, and the previous one does not adapt the original arrangement by Gittins himself directly – the index for Markov bandit processes obeys the definition of Peter Whittle. Nevertheless the indices defined by both authors are equal whenever they apply.

The fifth chapter concentrates on the generalisations of the index theorem, aiming at making the index applicable on a one-server queueing system. In addition the chapter concludes the theoretical part of this work, which is followed by a short experimental chapter number six. The sixth chapter illustrates the differences of the Gittins index and traditional approaches by a simple MAB problem. The work is closed in chapter seven with conclusions and discussion about further work that would be interesting to study in this topic.

Chapter 2

Markov decision processes

This section introduces the Markov decision processes in general, what kind of processes they are, what is the value of a Markov decision process, and various methods to determine the value of the processes. In addition multi-armed bandit systems and their index policies are discussed shortly.

2.1 The structure of a Markov decision process

A Markov decision process (MDP) [4, 11] is a probabilistic model of a system that changes state as time progresses, according to a decision process. The considered decision process is called D and it is observed at discrete time points t_i $(i = 0, 1, 2, \ldots)$ that are usually called *stages* or *decision times*. When speaking of an ordinary Markov decision process, the time points are also equally spaced, commonly $t_i = i \in \mathbb{N} \cup \{0\}$. At each decision time t_i the *state* of the process is denoted by $x(t_i)$, and it is assumed that $x(t_i)$ is a random variable that can take values from the finite set $\Theta = \{1, 2, \ldots, N\}$ which is called the *state space*. Furthermore the notation $\{x(t_i) = x\}$ corresponds with the event "the process is in state x at time t_i".

The decision process is in most cases maintained by a so-called *controller*, who chooses an *action* $a \in A(x) = \{1, 2, \ldots, m(x)\}$ at time t if the process is in state x at that time. The action may be regarded as a realisation of a random variable A_t denoting the controller's choice at time t. In addition it is assumed that every controller's choice results in an immediate reward $r(x, a)$ followed by a probabilistic transition to a new state $x' \in \Theta$. If the decision process obeys condition

$$p(x'|x, a) := \mathbf{P}\{x(t_{i+1}) = x'|x(t_i) = x, A_{t_i} = a\} \quad \forall\, t_i \ (i = 1, 2, \ldots),$$

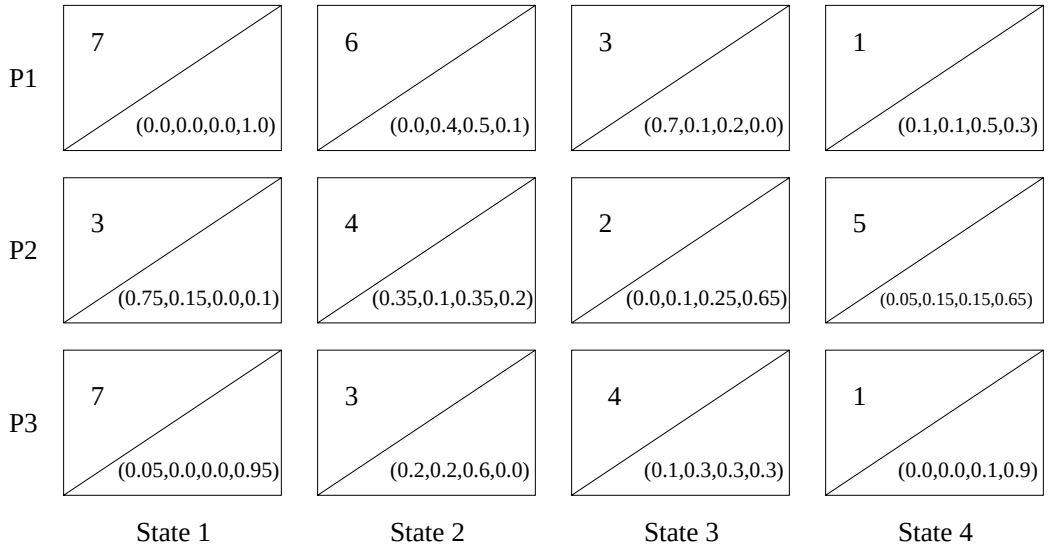
that is, the transition probabilities are stationary, or equivalently time-independent, and they do not depend on the history but only on the current state x then the process is called *Stationary Markov decision process*.

The example below this paragraph introduces a basic Markov Decision Process. The example is very similar to Example 2.1.1. in [4].

Example 1 (Markov decision process)**.** Let $\Theta = \{1, 2, 3, 4\}$, $A(1) = A(2) = A(3) = A(4) = \{1, 2, 3\}$. Since the amount of states and actions is limited, the data of

the problem can be represented conveniently in a format, in which each state of the process is illustrated with a rectangle (Fig. 2.1). In this representation a box

Figure 2.1: An illustration of a Markov decision process



with a diagonal divider portrays an action in a state and its reward with transition probabilities. For instance, above $r(1,1) = 7$ and transition probabilities $p(1|1,1) = 0.0$ and $p(1|2,2) = 0.35$. The question that arises from this example is, how one can achieve the best possible total reward? In this case, as often is, it is all but clear, because the states that yield greater rewards are not persistent, transitions away from the "good" states occur frequently. Although projects 1 and 3 have maximum reward of 7, it actually appears that project 2 has the best average reward.

It is possible to assign a *strategy* or equivalently a *policy* to the Markov decision processes. In the book of Filar and Vrieze [4] they are referred as strategies and represented as vectors, that are assigned to the states

$$\alpha(x) = \{\alpha(x,1), \alpha(x,2), \ldots, \alpha(x,m(x))\},$$

where $\alpha(x,a)$ represents the probability that the controller chooses action a when the process is in state x. Because the quantities are probability values, they must satisfy condition

$$\sum_{a=1}^{m(x)} \alpha(x,a) = 1.$$

In this work mainly policies that are *deterministic* are considered. A policy is deterministic if $\alpha(x,a) \in \{0,1\}$ for all $a \in A(x)$. That is, the controller selects some particular action a_x in state x with probability 1, whenever this state is visited. The definition includes also implicitly stationarity, because there is no dependency upon time t, and the decision are also memoryless or Markov, i.e. the decision depends only on the current state x. Such policies are determined stationary Markov policies. When such a policy is applied to a Markov decision process, the process becomes fully Markovian. Because the decisions are fully deterministic, there transitions of

the process are fully probabilistic. Actually the process is not any decision process any more, but a *Markov process*.

The state transition probabilities with a valid policy define a *transition probability matrix*, which is equivalently called a *stochastic matrix*, when speaking of Markov chains,

$$\mathbf{P}(\alpha) = (p(x'|x,\alpha))_{x,x'=1}^{N},$$

with entries given by

$$p(x'|x,\alpha) = \sum_{a=1}^{m(x)} p(x'|x,a)\alpha(x,a).$$

An example policy in Example 1 would be a policy that always chooses action, or project 2. The corresponding transition probability matrix is

$$\begin{bmatrix} 0.75 & 0.15 & 0.0 & 0.1 \\ 0.35 & 0.1 & 0.35 & 0.2 \\ 0.0 & 0.1 & 0.25 & 0.65 \\ 0.05 & 0.15 & 0.15 & 0.65 \end{bmatrix}.$$

A special case of interest among Markov decision processes, when considering this work, is a Multi-armed bandit, which will be introduced and discussed in the next section. Note that in following sections the dependency upon a policy is denoted simply with a single letter, which is in most cases α. The set of all policies is denoted with $\mathcal{A}$.

Occasionally the $\mathbf{P}$-matrix defines not one but at least two systems of linear equations. This event is a feature of the Markov Decision Process, as the process might not always end up on the same final states called *recurrent class*. This type of Markov Decision Process is called *multichain* [11, §8.3.1]. If there exists only one recurrent class, the type of the MDP is *unichain*.

2.2 Multi-armed bandit

Multi-armed bandit (MAB) [19, 14, 6] is a machine learning problem, a special type of a Markov decision process and a generalisation of the slot machine [14, §2.1], also known as one-armed bandit. Suppose that a player has n gambling machines (the "bandits") and wishes to choose the machine which he plays at each stage so as to maximise the total expected pay-off, while the other machines remain silent and preserve their states. The pay-off may be discounted with a factor $\beta \in (0,1)$ if the time horizon is infinite or alternatively the average reward can be observed.

The pay-off probability of the ith machine is unknown. However, gambler may build up an estimate of the pay-off probability, the probability that a gambler benefits from gambling with a machine, which will improve as the gambler gains more experience of the machine. The conflict, then, is between playing a machine that is known to have a good pay-off probability and experimenting with a machine about which little is known, but which just might prove even better.

A bandit process has a control set $A(x)$, that consists, for every state x, of two elements 0 and 1. Control 0 *freezes* the process in the sense that when it is applied

no reward accrues and the state of the project or process does not change. In contrast, control 1 is termed the *continuation control*. Each bandit process behaves independently from other bandits like a Markov decision process.

In addition, a *standard* bandit process is a dummy process that has only one state, and for which every point of time is a decision time. Thus a policy for a standard bandit process is simply a random Lebesgue-measurable function $I(t)$ from the positive real line to the set $\{0, 1\}$, specifying the control to be applied at each point in time. The total reward yielded by such a policy is $\lambda \int \beta^t I(t) \mathrm{d}t$, where the parameter λ depends on the particular standard bandit process. A set of standard bandit processes with discount factor β is denoted by Λ_β, or if there is no question about confusion, simply Λ.

The *state space* of a MAB system is the product space of the state spaces of the bandits, including all the possible state combinations. The objective is to find the best policy α, that is, the optimal decision rule, how the gambling should be carried on at each stage and state of the state space.

2.3 Value of a MDP and Howard equation

The discounted value of a state of a Markov decision process is the expected total discounted reward given by a fixed policy α in infinite time [11, §5.1]

$$v(\alpha, x) = \lim_{n \to \infty} \mathbf{E} \left[\sum_{i=0}^{n} \beta^{t_i} r(\alpha, x(t_i)) | x_0 = x \right], \tag{2.1}$$

where $x(t_i)$ is the state of the MDP at time t_i. Further the value of the process after n steps is defined as

$$v_n(\alpha, x) = \mathbf{E} \left[\sum_{i=0}^{n} \beta^{t_i} r(\alpha, x(t_i)) | x_0 = x \right] \tag{2.2}$$

Note that the terminal reward $r(\alpha, x(t_i))$ has dependence upon the policy, i.e. the reward depends on the decision that is made at time t in state x_t. Moreover note that this expectation is always finite, if the terminal reward $r(\alpha, x_t)$ for all t is bounded by a constant.

The infinite sum is an inconvenient way to determine the value of a certain state with a fixed policy. Fortunately the value of a state depends upon the values of other states of the Markov decision process according to the Howard equation [18, 11, 15, 16]

$$v(\alpha, x) = r(\alpha, x) + \beta E[v(\alpha, x') | x], \tag{2.3}$$

where x' is the (stochastic) state of the process after applying a policy α in state x. Derivation of this formula is intuitive, the value of a project should be equal to the accrued reward after one step of continuation added with the expected value of the project after the step, conditioned on the initial condition of the project.

When the transition probabilities of the process are known, it is possible to write the expectation in (2.3) in terms of the transition probability matrix and values of

single states. Thus (2.3) can be rewritten for all states in form

$$v(\alpha) = r(\alpha) + \beta \mathbf{P}(\alpha) v(\alpha) \tag{2.4}$$

where $\mathbf{P}(\alpha) \in \mathbb{R}^n \times \mathbb{R}^n$ (stochastic), $r(\alpha) \in \mathbb{R}^n$ and $v(\alpha) \in \mathbb{R}^n$ represent the policy dependent transition probability matrix, the policy dependent reward vector and the value vector, respectively. The same notation may also be applied to (2.1) and (2.2).

Equation (2.4) is solvable in most cases, because $(I - \beta \mathbf{P})$ is invertible:

$$v(\alpha) = (I - \beta \mathbf{P}(\alpha))^{-1} r(\alpha). \tag{2.5}$$

If the MDP system does not feature reward discounting and the average reward is observed instead, the previous equation (2.5) may not be used, since the multiplier matrix becomes singular. The definition of the discounted value of a state for discounted model (2.1) neither applies further, instead, average value (reward) is defined by

$$\bar{v}(\alpha, x) = \lim_{n \to \infty} \frac{1}{n} \mathbf{E} \left[\sum_{i=0}^{n} r(\alpha, x(t_i)) | x(t_0) = x \right],$$

or by vector representation

$$\bar{v}(\alpha) = \lim_{n \to \infty} \frac{1}{n} \underbrace{\sum_{i=0}^{n} P^i r(\alpha)}_{:= v_n(\alpha)}.$$

Further there exists a modification of the Howard equation for the average reward model

$$\bar{v}(\alpha) + v(\alpha) = r(\alpha) + \mathbf{P}(\alpha) v(\alpha). \tag{2.6}$$

In the equation $v(\alpha) \in \mathbb{R}^n$ represents the *relative values* of different states. If the Markov process related to policy α is aperiodic and irreducible, every initial state will produce exactly the same average reward as all other states, but nevertheless some states still are better than others, because they might produce better rewards in the early phase of the process. Relative values represent these differences. If states j and k are in the same closed class (transition from j to k is possible in finite time with positive probability and vice versa) the following holds

$$v(\alpha, j) - v(\alpha, k) = \lim_{N \to \infty} [v_n(\alpha, j) - v_n(\alpha, k)].$$

Hence v equals the asymptotic relative difference in total reward that results from starting the process in state j instead of in state k.

If the MDP is unichain, one may alternatively consider the average reward as constant, independent from the initial state, but when observing the multichain case, the average reward may depend on the initial state. Often the vector is nevertheless constant.

Equation (2.6)can be easily motivated as follows. The total reward after n steps is noted with v_N, and v_n is related to v_{n+1} with

$$r(\alpha) + \mathbf{P}(\alpha) v_n(\alpha) = v_{n+1}(\alpha). \tag{2.7}$$

Alternatively v_n may asymptotically be expressed in terms of the average reward

$$v_n(\alpha) \to n\bar{v}(\alpha) + v(\alpha). \tag{2.8}$$

Then (2.8) is substituted in (2.7). If the approximation of the average reward $\bar{v}$ is good enough, i.e. the necessary equality $\mathbf{P}\bar{v} = \bar{v}$ holds, it follows

$$\begin{aligned} & r(\alpha) + \mathbf{P}(\alpha)(n\bar{v}(\alpha) + v(\alpha)) = (n+1)\bar{v}(\alpha) + v(\alpha) \\ \Leftrightarrow \quad & r(\alpha) + n\mathbf{P}(\alpha)\bar{v}(\alpha) + \mathbf{P}(\alpha)v(\alpha) = n\bar{v}(\alpha) + \bar{v}(\alpha) + v(\alpha) \\ \Leftrightarrow \quad & r(\alpha) + \mathbf{P}(\alpha)v(\alpha) = \bar{v}(\alpha) + v(\alpha), \end{aligned}$$

which is the average model Howard equation (2.6). Note that if some $v(\alpha)$ satisfies the equation above, also any vector that is got by summing a constant to $v(\alpha)$ also satisfies the equation. Thus if one decides to solve the relative rewards from (2.6), an additional constraint is needed.

2.4 Dynamic programming equation

In following sections the objective is to find an optimal policy for a Markov decision process. The optimal policy is the solution of the following dynamic programming equation also known as Bellman's Equation [2, 18]

$$v^* = \max_{\alpha \in A}\{r(\alpha) + \beta\mathbf{P}(\alpha)v^*\}, \tag{2.9}$$

or if the model features average reward optimisation

$$\bar{v}^* + v^* = \max_{\alpha \in A}\{r(\alpha) + \mathbf{P}(\alpha)v^*\}. \tag{2.10}$$

In the equation (2.9) v^* is the optimal discounted reward, which is the value of the optimal policy. In symbols

$$v^* = \sup_{\alpha \in A} v(\alpha).$$

In the latter equation (2.10) the average reward is optimal

$$\bar{v}^* = \sup_{\alpha \in A} \bar{v}(\alpha).$$

The dynamic programming equation appears also in the book of Ross [12], although instead of maximising the rewards of a MDP, he aims at minimising costs of the process, that is, the process is considered to cause costs instead of yielding rewards. If the costs are minimised, the maximum in (2.9) and (2.10) is replaced with minimum and the rewards $r(\alpha)$ with (commonly non-positive) costs $c(\alpha)$. Nevertheless, in this work the only cost that is accrued is the loss of time.

There are plenty of different algorithms to infer the optimal policy or an approximation to the dynamic programming equations. In the following sections some of them are introduced. In addition, in the experiments section the methods are compared with an example Multi-armed bandit system.

2.5 Policy iteration

Policy iteration [11, §6.4] is a fast converging algorithm, which iteratively infers the optimal value and policy of a Markov decision process. This section is divided into two subsections, of which the first one handles the discounted reward model and the second one concentrates on optimising the average reward.

2.5.1 Discounted policy iteration

Basically, the idea of policy iteration is simple. The iteration is begun with an arbitrary policy $\alpha_0 \in \mathcal{A}$. The policy yields a certain reward that can be evaluated with the help of Howard equation (2.4). After the evaluation the policy is improved by selecting a policy that maximises the expected reward with respect to the newly updated value vector. The algorithm is sketched as follows.

Definition 1 (Discounted policy iteration algorithm)**.**

1. *Set $n = 0$ and select an arbitrary policy $\alpha_0 \in A$.*
2. *(Policy evaluation) Evaluate the value vector $v(\alpha_n) \in \mathbb{R}^n$ of the policy with the Howard equation* (2.4)
$$v(\alpha_n) = (I - \beta \mathbf{P}(\alpha_n))^{-1} r(\alpha_n). \tag{2.11}$$
3. *(Policy improvement) Choose $\alpha_{n+1} \in A$ which satisfies*
$$\alpha_{n+1} \in \arg\max_{\alpha \in \mathcal{A}} \{r(\alpha) + \beta \mathbf{P}(\alpha) v(\alpha_n)\},$$
setting $\alpha_{n+1} = \alpha_n$ if possible. Note, that the policy can be inferred trouble-free through observing the policy component-wise, the matrix notation is used only to simplify the notation.
4. *If $\alpha_{n+1} = \alpha_n$, stop and set $\alpha = \alpha_n$. Otherwise, increment n by 1 and return to step 2.*

Note that the policy does not have to be unambiguous. The procedure is repeated by increasing n by one until it is possible to set $\alpha_{n+1} = \alpha_n$, since then the value vector cannot improve any further, even if the policy was selected otherwise. Generally, it is possible that the recursion is infinite, but in the experiment of this work the set of policies is finite and therefore a policy providing optimal discounted reward is surely found, usually after a few steps of iteration. The advantage of this algorithm is its rapidity, but meanwhile the algorithm is quite laborious because of the inversion of the multiplier matrix in (2.11), which becomes more difficult as the number of states and actions grow.

2.5.2 Policy iteration with average reward

The average reward model policy iteration [15, 16, 11] is essentially based on the average reward Howard equation (2.6).

$$\begin{aligned} & \bar{v}(\alpha) + v(\alpha) = r(\alpha) + \mathbf{P}(\alpha) v(\alpha) \\ \Leftrightarrow \quad & (I - \mathbf{P}(\alpha)) v(\alpha) = r(\alpha) - \bar{v}(\alpha). \end{aligned} \tag{2.12}$$

As in the discounted reward model, in each iteration round the value vector is updated and then the corresponding optimal policy is sought. The problem of the algorithm is that the Howard equation is not trivially solvable. In principle the equation above becomes solvable when one of the relative rewards in vector $v(\alpha)$ is fixed, because then the equation has exactly as many unknown variables as there are equations — in most cases.

The book of Puterman provides an excellent algorithm for solving the average reward of a multichain Markov Decision Process [11, §9.2.1]. Instead of fixing a value of $\bar{v}$ per each recurrent class alternative subject conditions are provided to augment the problem. The condition used in this work is

$$\mathbf{P}^*(\alpha_n)v(\alpha_n) = 0. \tag{2.13}$$

This condition requires that the relative rewards must be equal to zero when they are time-averaged, since the matrix $\mathbf{P}^*(\alpha_n)$ is the transition probability matrix of policy α_n with infinite number of transitions

$$\mathbf{P}^*(\alpha_n) = \lim_{n\to\infty} \mathbf{P}(\alpha_n)^n. \tag{2.14}$$

Alternatively, if the MDP is unichain, then the transition probability matrix with infinite number of transitions may be calculated straightforward with following procedure [11, §A.4]. If vector π is the *stationary distribution* of the MDP, it satisfies equations

$$\pi\mathbf{P} = \pi \quad \wedge \quad \|\pi\|_1 = 1,$$

and the transition probability matrix $\mathbf{P}$ is unichain and aperiodic, then the limiting matrix (2.14) may be calculated directly with

$$\mathbf{P}^* = \pi e^T,$$

where e is a vector whose all components are equal to one. Unfortunately, unlike in the unichain case, in the multichain case the row vectors of the limiting matrix are not identically constant. The complete algorithm is sketched as follows.

Definition 2 (Average reward policy iteration algorithm)**.**

1. *Set $n = 0$ and select an arbitrary policy $\alpha_0 \in \mathcal{A}$.*
2. *(Policy evaluation) Obtain a $\bar{v}(\alpha_n)$ and a $v(\alpha_n)$ which satisfy*

$$(\mathbf{P}(\alpha_n) - I)\bar{v}(\alpha_n) = 0,$$
$$r(\alpha_n) - \bar{v}(\alpha_n) + (\mathbf{P}(\alpha_n) - I)v(\alpha_n) = 0$$

   subject to restriction (2.13)*.*
3. *(Policy improvement)*
   (a) *Choose $\alpha_{n+1} \in A$ which satisfies*

$$\alpha_{n+1} \in \arg\max_{\alpha\in\mathcal{A}}\{\mathbf{P}(\alpha)\bar{v}(\alpha_n)\},$$

   setting $\alpha_{n+1} = \alpha_n$ if possible. If $\alpha_{n+1} = \alpha_n$ go to (b), otherwise increment n by 1 and return to step 2.

(b) Choose $\alpha_{n+1} \in A$ *to satisfy*

$$\alpha_{n+1} \in \arg\max_{\alpha \in \mathcal{A}} \{r(\alpha) + \mathbf{P}(\alpha) v(\alpha_n)\},$$

setting $\alpha_{n+1} = \alpha_n$ *if possible.*

4. *If* $\alpha_{n+1} = \alpha_n$*, stop and set* $\alpha = \alpha_n$*. Otherwise, increment* n *by* 1 *and return to step 2.*

2.6 Value iteration

Unlike the policy iteration, the value iteration algorithm [11, §6.3] has no potential calculation problems, but the drawback of the algorithm is that it may converge slowly and it does not necessarily provide optimal results, but only approximations of an optimal policy.

The idea of the algorithm is to find the ε-optimal value vector through an iteration procedure, which resembles fixed point iteration, and then to determine the corresponding most valuable policy.

2.6.1 Discounted value iteration

The following rule defines the ε-optimal discounted value iteration algorithm.

Definition 3 (Discounted value iteration algorithm)**.**

1. *First the value vector* v^0 *is fixed to an arbitrary value, e.g.* $v^0 \equiv 0$*,* $\varepsilon > 0$ *is chosen and* n *is set to* 0*.*

2. *The value iteration is performed with equation*

$$v_{n+1} = \max_{\alpha \in \mathcal{A}} \{r(\alpha) + \beta \mathbf{P}(\alpha) v_n\}. \tag{2.15}$$

3. *If*

$$\|v_{n+1} - v_n\| < \varepsilon \frac{1-\beta}{2\beta}, \tag{2.16}$$

terminate the iteration and set

$$\alpha_\epsilon \in \arg\max_{\alpha \in \mathcal{A}} \{r(\alpha) + \beta \mathbf{P}(\alpha) v_{n+1}\}$$

component-wise. Otherwise increment n *by* 1 *and repeat step 2.*

When condition (2.16) *is fulfilled, the value vector is at least* ε*-optimal [11, Theorem 6.3.1]*

$$\|v_{n+1} - v^*\| < \frac{\varepsilon}{2}.$$

Despite the algorithm is guaranteed to give only ε-optimal policies, it is nevertheless possible that the policy is optimal. The optimality can be verified with Equation 2.9 by replacing v^* with the result of the algorithm, and experimenting, whether the equation holds or not.

2.6.2 Value iteration with average reward

Value iteration in the average reward model is based on the idea that the difference of two successive value vectors evaluated by (2.15) setting β to 1 tend to the average reward assuming that the transition matrix $\mathbf{P}$ is acyclic

$$\bar{v} = \lim_{n\to\infty} |v_{n+1} - v_n|. \tag{2.17}$$

The relative rewards are then

$$v = \lim_{n\to\infty} (v_n - n\bar{v}). \tag{2.18}$$

These results are proven and discussed by Puterman [11, §9.4.1], but there are also alternative approaches to determine the average reward. One option is to set β near enough to one and to find a policy that is ε-optimal for the chosen discount factor. In some cases the acquired policy may be the optimal policy. The optimality can be verified with the dynamic programming equation (2.10), and the optimality topic is also discussed in [12, §6.17].

Another possible approach is similar to the latter solution. Again, the discounted reward is evaluated with discount factor β close to one, therefore each value vector v^n is discounted with factor β^n, that likewise is almost one. The greater β is, the better the approximation of the average reward is achieved, when the total discounted reward is scaled with the weights of the sum, i.e.

$$\sum_{k=0}^{\infty} \beta^k = \frac{1}{1-\beta}. \tag{2.19}$$

Hence the discounted reward normalised with (2.19) is an approximation of the average reward. The approximation becomes better as β approaches one

$$\lim_{\beta\to 1} v_\beta(1-\beta) = \bar{v}.$$

2.7 Greedy policy

If the value iteration algorithm is initialised with an identically zero value vector, the policy acquired after one step of iteration is actually the naive and simplest estimate of the optimal policy, the *greedy* policy. Greedy policy maximises always the immediate reward, i.e. chooses always the control that yields the best possible *immediate* reward, no matter what the future rewards would be. In terms of mathematical symbols the policy is defined as

$$\alpha_{greedy} := \arg\max_{\alpha\in\mathcal{A}} r(\alpha), \tag{2.20}$$

which can be obtained from (2.15) by setting $v \equiv 0$, as noted in the beginning of this paragraph. Unfortunately this policy is **not** usually the optimal policy [13].

2.8 Index policy on bandit systems

The following paragraphs discuss the Gittins index policy, but before discussing index policies it should be explained what they are in general. Suppose that there are n projects, each denoted by B_i $(i = 1, 2, \ldots, n)$ that produce certain rewards and should be ordered in some certain way, preferably optimal, to earn the optimal reward. One possibility to express the policy in question is simply to give an order explicitly, but an *index policy* gives the order implicitly.

Definition 4 (Index policy)**.** *Suppose that there is a mapping from every project B_i and state $x_i(t)$ to a real value $\nu(B_i, x_i(t))$. An index policy with respect to $\nu(B_i, x_i(t))$ chooses always the project k that has the greatest index at time t*

$$k = \arg\max_{1 \leq i \leq n} \nu(B_i, x_i(t)). \tag{2.21}$$

Although an index policy was defined here exclusively for bandit systems, it is worth noting that similar index policies may also be constructed in many other environments.

The next paragraph provides an additional policy that is an index policy, the *Gittins index policy*. However, the policy in question is not applicable on Markov decision processes in general, but on Multi-Armed Bandit systems. Therefore it is naturally to defer the introduction of the Gittins index out from this MDP chapter to the next chapter, which is dedicated for the index.

In contrast, the policies defined earlier in this chapter can be also applied on Multi-armed bandit systems. The difference to MDPs in general is that the bandits have only two actions, freeze and continuation controls, but there may be more than one bandit. One may conveniently suppose that the decision of a bandit is the action that is associated to the Markov Decision process.

Later on, the bandit processes are called projects and the value functions of single bandit processes are noted with ϕ_i and states with x_i. It is also good to note that the transition probability matrix $\mathbf{P}$ in equations from equation (2.4) on in case of an MAB system is very sparse, because transition from every state combination to another, arbitrary state combination in general is not possible at all in one single step, if the states of the projects differ in more than one project.

In the following section the observed bandit system is supposed to have n machines or projects, where each of them can stay in m different states. Therefore there are m^n possible combinations of the states and n different decisions for every state combination, hence the option of evaluating all the policies is not practicable at all, since there are $n^{(m^n)}$ different policies. Thus the Gittins index proves to be very advantageous policy generator.

Chapter 3

The Gittins index for Markov bandit processes

In this section the Gittins index is handled in a normal Markov bandit process setting following the approach by Whittle [19, §14], while the next chapter considers the setup in a *semi*-Markov environment studied by Gittins [6] himself. Additionally both chapters discuss the optimality of the index.

3.1 Whittle's definition of the Gittins index

Whittle approaches the multi-armed bandit problem considering it as an *optimal stopping problem*. Each project is associated with a 'retirement' reward M and the gambler has always the option of retiring, that is, stopping a project and getting a retirement reward of M. The value function of the a project is denoted by $\phi(x, M)$ and it obeys the dynamic programming equation [18, 17], which was already discussed without retirement option in (2.9),

$$\phi(x, M) = \max\{M, r(x) + \beta E[\phi(x', M)|x]\}, \tag{3.1}$$

which may be abbreviated to

$$\phi = \max\{M, L\phi\}. \tag{3.2}$$

Here x and x' are the states of the project before and after one stage of operation and $\phi(x, M)$ is the value of the project augmented with the stopping option and retirement reward M.

With respect to M the function $\phi(x, M)$ is non-decreasing and convex [19], and for values of M greater than a critical value $M(x)$ it is equal to M. Above $M(x)$ it is optimal to accept the retirement reward instead of continuing the project and at $M(x)$ the choices between continuing and retiring are equally attractive. This quantity is the best value to be taken as the Gittins index, or more preferably scaled with the geometric sum of discount factor. Consequently the scaled index represents the *discounted average reward* of the project or, as later will be described, the *equivalent constant reward rate*

$$\nu_i(x) = (1 - \beta)M(x). \tag{3.3}$$

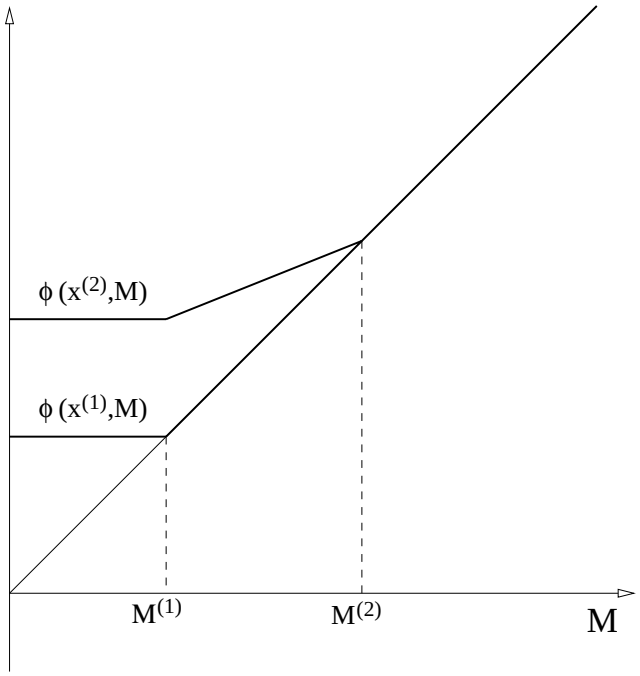


Figure 3.1: The graph of $\phi(x, M)$ as a function of M. Here $x \in \{x^{(1)}, x^{(2)}\}$ and M are respectively the state of the project and the reward if the retirement option is accepted, and $\phi(x, M)$ is the value function of the corresponding optimal stopping problem

There is an example graph of two value functions of a project in two different states $x \in \{x^{(1)}, x^{(2)}\}$ in Figure 3.1. As the figure illustrates, the value functions intersect with line $f(M) = M$ at points $M^{(1)}$ and $M^{(2)}$.

Note that M is not the fair buy-out price for the project when in state x rather the size of an *annuity* that a capital sum M would buy. One has to choose between two alternatives, either to "invest" in the project with uncertain return for some desired time and then reconsider the decision, or to move to the 'certain' project which offers a constant income of ν.

3.2 Properties of the index and $\phi(x, M)$

Bertsekas proves an interesting result about the differential of the value function [2, Lemma 1.5.1]. The slope of the value function depends on the retirement time as follows

Lemma 1. *For fixed state x, let τ_M denote the retirement time under an optimal policy when the retirement reward is M. Then for all M for which $\partial\phi(x, M)/\partial M$ exists the derivative obeys equation*

$$\frac{\partial \phi(x, M)}{\partial M} = \boldsymbol{E}\{\beta^{\tau_M} | x_0 = x\}.$$

If the optimal retirement time is zero, one should accept the retirement reward immediately. This means that the slope of the value function is one, i.e. $\phi(x, M) =$

M for this fixed state x. If the retirement reward is small, the bandit should be continued forever, having the retirement time $\tau_M = \infty$. From the lemma it follows that the value function is constant at that point. Otherwise the retirement time is positive and finite, corresponding to a slope factor greater than zero but less than one. The proof of the lemma is quite straightforward, but is not essential to this study and is therefore bypassed. The proof by difference quotient is provided in [2, Lemma 1.5.1].

From the definition of the index (3.3) it follows that the index is equal to the *equivalent constant reward rate* of the project. This quantity is the constant reward of a standard bandit process that would produce exactly as good rewards as the now observed bandit B in state x. The quantity will be used more often in the next Chapter, but here it is needed to understand what quantities the indices really are.

When undergoing the experiments with the Gittins index some interesting observations were made. The greatest index of the bandit B in some state x_{max} was always equal to the immediate reward of the state, say R. This is evident, because a bigger reward cannot be achieved anyhow, but it is always beneficial to continue the bandit at least once, if M is less than R, therefore $\nu(B, x_{max}) = R$. In contrary, suppose that the least index of the bandit was assigned to state x_{min}. The index seemed to be close to the average reward of the bandit process as the discount factor β approached one, which is natural, because the index equals the discounted average reward of the project. Alternative explanation is to observe that it is always worth to run a project if M is less than the average reward of the bandit process, assuming that the Markov decision process of the bandit B is unichain.

The benefits of the Gittins index include the property that the index is evaluated in terms of one project alone. The optimal policy is retrieved from the order of the indices, the project with greatest index should be chosen at each stage of the process. The n-dimensional problem has been reduced to a stopping problem of individual projects. Therefore the observations in the following section are concentrated on one individual project.

3.3 Solving ϕ-functions

By setting first M to zero it is trivial to determine the values of the ϕ-functions, because the maximisation in (3.2) can be ignored if the non-negativity of ϕ-functions is assumed. The equation (3.2) is reduced to

$$\begin{aligned}\phi(x,0) &= L\phi(x,0) \\ &= r(x) + \beta E[\phi(x',0)|x].\end{aligned}$$

The expectation of the value function may be expressed with help of the transition probabilities and other value functions as follows

$$\phi(x,0) = r(x) + \beta \sum_k p_{jk}\phi(k,0),$$

where $j = x$, accompanied with $r(x)$ and p_{jk}, which represent the rewards and transition probabilities of a single project respectively. Further this system of linear equations can be expressed with matrices

$$\begin{aligned} & \phi(\mathbf{x},0) = \mathbf{r}(\mathbf{x}) + \beta \mathbf{P}\phi(\mathbf{x},0) \\ \Leftrightarrow \quad & \phi(\mathbf{x},0) = (I - \beta\mathbf{P})^{-1}\mathbf{r}(\mathbf{x}), \end{aligned}$$

that is, the Howard equation. This system of linear equations is solvable if $\beta < 1$, and the values for $\phi(x,0)$ for every i and x are obtained. These values are, in fact, the discounted cumulative rewards of B without the retirement option, and by multiplying them by $(1-\beta)$ the corresponding discounted average rewards are obtained. Further because ϕ is non-decreasing but at least M and (3.2) holds, it is clear, that $\phi(x,M) = \phi(x,0)$ for every $M \leq M^{(1)}$, where

$$M^{(1)} := \min_x \ \phi(x,0).$$

This quantity multiplied by $(1-\beta)$ is also the Gittins index of the project in state $x^{(1)}$, where

$$x^{(1)} := \arg\min_x \phi(x,0).$$

Further it is derived that

$$\phi(x^{(1)},M) = \begin{cases} \phi(x^{(1)},0) & M \leq \phi(x^{(1)},0) \\ M & \text{otherwise} \end{cases},$$

a one-edge piecewise defined function. With this information the algorithm may be taken one step further, now it is possible to determine the second least index $M^{(2)}$ and corresponding state $x^{(2)}$. The value function of the project having the second least index is two-edged, third graph three-edged and so forth.

3.4 The optimality of the Gittins index

In this section the Whittle approach to prove the optimality of the Gittins index is shown. Whittle provides the proof for the problem augmented with the retirement option. After solving the problem, the solution for the original problem is also acquired at once by setting $M = 0$.

In following the composite state $(x_1, x_2, \ldots, x_n)$ is denoted with x and the value function for the problem of choosing optimally between any one of these projects and the additional option of total retirement with a terminal reward M is denoted with $\Phi(x,M)$. The function obeys equation

$$\Phi = \max\{M, \max_i L_i\Phi\}, \tag{3.4}$$

where L_i is defined in the same way as L in (3.2)

$$L_i\Phi = r_i(x_i) + \beta\mathbf{E}[\Phi(x',M)|x] := r_i(x_i) + \beta\mathbf{E}[\Phi(x + (x_i' - x_i)e_i)|x_i],$$

where L_i acts only on the x_i-argument of Φ, i.e. x' and x differ only on the i:th component.

Furthermore the uniform upper and lower bounds on the reward rates $r_i(x_i)$ are denoted with $A(1-\beta)$ and $B(1-\beta)$, respectively, so that A and B would be the lower and upper bounds on the total expected discounted reward obtainable if there was no retirement option. Therefore there exists following limits for the discounted reward and $M_i(x_i)$

$$\begin{aligned} & \sum_{k=0}^{\infty} \beta^k \min_i r_i(x_i) \leq \phi_i(x_i, 0) \leq \sum_{k=0}^{\infty} \beta^k \max_i r_i(x_i) \\ \Leftrightarrow \quad & \frac{1}{1-\beta} A(1-\beta) \leq \phi_i(x_i, 0) \leq \frac{1}{1-\beta} B(1-\beta). \end{aligned}$$

The statement follows immediately

$$A \leq \phi_i(x_i, 0) \leq B.$$

Thus $\phi_i(x_i, M)$ intersects with M likewise inside the same interval, and it is concluded that for $M_i(x_i)$ holds

$$A \leq M_i(x_i) \leq B \quad \forall i.$$

3.4.1 Simple evaluation of the value function

In following it is shown that the value function Φ has the evaluation in terms of the one-project value functions ϕ_i. For the time being this evaluation is noted by Ψ

$$\Psi(x, M) = B - \int_M^B \prod_i \frac{\partial \phi_i(x_i, m)}{\partial m} \mathrm{d}m. \tag{3.5}$$

Lemma 2. *The expression* (3.5) *may alternatively be written*

$$\Psi(x, M) = \phi_i(x_i, M)\mathbf{P}_i(x, M) + \int_M^{\infty} \phi_i(x_i, m) \mathrm{d}_m \mathbf{P}_i(x, m), \tag{3.6}$$

where i is arbitrary (among the project indices)

$$\mathbf{P}_i(x, M) := \prod_{j \neq i} \frac{\partial \phi_j(x_j, M)}{\partial M},$$

and $\mathrm{d}_m \mathbf{P}_i(x, m)$ is the differential corresponding to $\mathrm{d}m$.

Proof. Note that $\mathbf{P}_i(x, M)$ is non-negative, non-decreasing in M and equal to unity for

$$M > M_{(i)} := \max_{j \neq i} M_j.$$

The properties of $\mathbf{P}_i$ follow immediately from properties of $\phi_i(x_i, M)$, since ϕ_i, as a function of M, is non-decreasing, convex and equal to M for $M \geq M_i$. Observe that

M_i and $M_{(i)}$ have dependence upon x that is suppressed. With these observations the proof of lemma is straightforward

$$\begin{aligned}\Psi(x,M) &= B - \int_M^B \prod_i \frac{\partial \phi_i(x_i,M)}{\partial M}\mathrm{d}m \\ &= B - \Big|_M^B \phi_i(x_i,m)\mathbf{P}_i(x,m) + \int_M^B \phi_i(x_i,m)\mathrm{d}_m\mathbf{P}_i(x,m) \\ &= B - \underbrace{\phi_i(x_i,B)}_{=B}\underbrace{\mathbf{P}_i(x,B)}_{=1} + \phi_i(x_i,M)\mathbf{P}_i(x,M) \\ &\quad + \int_M^B \phi_i(x_i,m)\mathrm{d}_m\mathbf{P}_i(x,m) + \int_B^\infty \phi_i(x_i,m)\underbrace{\mathrm{d}_m\mathbf{P}_i(x,m)}_{=0} \\ &= \phi_i(x_i,M)\mathbf{P}_i(x,M) + \int_M^\infty \phi_i(x_i,m)\mathrm{d}_m\mathbf{P}_i(x,m). \end{aligned}$$ □

Consider the quantity

$$\delta_i(x_i,M) = \phi_i(x_i,M) - L_i\phi_i(x_i,M)$$

and note that $\delta_i \geq 0$, with equality for $M \leq M_i$ which is a direct consequence of (3.2). Note that the x-argument of M was suppressed to simplify the notation.

Lemma 3. *Expression* (3.5) *and hence* (3.6) *satisfies the relations*

$$\Psi \geq M \tag{3.7}$$

with equality if $M \geq \max_j M_j$, *and*

$$\Psi(M) - L_i\Psi(M) = \delta_i(M)\mathbf{P}_i(M) + \int_M^\infty \delta_i(m)\mathrm{d}_m\mathbf{P}_i(m) \geq 0 \tag{3.8}$$

with equality if $M_i = \max_j M_j \geq M$.

Proof. Inequality (3.7) and the characterisations of the equality case follow from (3.5), since

$$\prod_i \frac{\partial \phi_i(x_i,M)}{\partial M} = 1 \quad \text{for} \quad M \geq \max_j M_j$$

and otherwise at most 1, because the partial derivatives are non-decreasing. It is concluded that

$$\Psi = B - \int_M^B \prod_i \frac{\partial \phi_i(x_i,M)}{\partial M}\mathrm{d}m \geq B - \int_M^B \mathrm{d}m = B - (B-M) = M.$$

The second part (3.8) of the lemma follows immediately from (3.6)

$$\begin{aligned}&\Psi(x,M) - L_i\Psi(x,M) \\ &= \phi_i(x_i,M)\mathbf{P}_i(x,M) + \int_M^\infty \phi_i(x_i,m)\mathrm{d}_m\mathbf{P}_i(x,m) \\ &\qquad - L_i\phi_i(x_i,M)\mathbf{P}_i(x,M) - L_i\int_M^\infty \phi_i(x_i,m)\mathrm{d}_m\mathbf{P}_i(x,m) \\ &= \left(\phi_i(x_i,m) - L_i\phi_i(x_i,M)\right)\mathbf{P}_i(x,M) \\ &\qquad + \int_M^\infty \left(\phi_i(x_i,m) - L_i\phi_i(x_i,M)\right)\mathrm{d}_m\mathbf{P}_i(x,m) \\ &= \delta_i(M)\mathbf{P}_i(M) + \int_M^\infty \delta_i(m)\mathrm{d}_m\mathbf{P}_i(m). \end{aligned}$$

The non-negativity of the expression is trivial, since all the factors are non-negative. The equality case is an immediate consequence of observations $\delta_i(M) = 0$ for $M \leq M_i$ and $\mathrm{d}_m\mathbf{P}_i(m) = 0$ for $m \geq M_{(i)}$. Then if M_i is the greatest of all M_j:s, i.e. $M_i = \max_j M_j$, it holds $M_i \geq M_{(i)}$ and the expression (3.8) is equal to zero as was claimed. □

Theorem 4. *The function $\Psi(x, M)$ (3.5) is indeed a solution of the augmented problem and the Gittins index policy is optimal.*

Proof. The assertions of Lemma 3 guarantee that $\Psi(x, M)$ actually satisfies the dynamic programming equation (3.4) and the Gittins index policy, the policy which chooses the project with the greatest index (augmented by the recommendation of termination if M exceeds $\max_i M_i$) provides the maximising option in (3.4). But since the solution of (3.4) is unique and the maximising option indicates the optimal action both assertions are proved. □

Further an example, in which state follows a diffusion process, is given. The example is cited from [19, p. 276].

Example 2 (Diffusion process)**.** Suppose that the state x of the project takes values on the real line, that the project yields reward at rate $r(x) = x$ while it is being operated, that reward is discounted at rate α, and that x itself follows a diffusion process with drift and diffusion coefficients μ and N. This conveys the general idea of a project whose return improves with its 'condition', but whose condition varies randomly.

The value function obeys in this case the equation

$$x - \alpha\phi + \mu\phi_x + \frac{1}{2}N\phi_{xx} = 0, \quad (x > \xi) \tag{3.9}$$

where ξ is the optimal breakpoint for retirement reward M. The easily verifiable solution of the differential equation (3.9) is

$$\phi(x, M) = \frac{x}{\alpha} + \frac{\mu}{\alpha^2} + ce^{px}, \tag{3.10}$$

where p is the negative solution of

$$\frac{1}{2}Np^2 + \mu p - \alpha = 0,$$

and c is an arbitrary constant. The general solution of (3.9) would also contain an exponential term corresponding to the positive root of this last equation, but this will be excluded since ϕ cannot grow faster that linearly with increasing x. The unknown variables c and ξ may be determined by the boundary conditions $\phi(\xi, M) = M$ and $\phi_x(\xi, M) = 0$. Substituting the solution (3.10) to these boundary conditions gives finally the following relation between M and ξ, and further gives a representation of the Gittins index

$$\nu(x) = \alpha M(x) = x + \frac{\mu + \sqrt{\mu^2 + 2\alpha N}}{2\alpha}.$$

The constant added to x represents the future discounted reward expected from future change in x. This is positive even if μ is negative – a consequence of the fact that one can take advantage of a random surge against trend if this occurs, but can retire if it does not.

Chapter 4

The Gittins index for semi-Markov bandit processes

Whittle's definition of the Gittins index in the previous chapter was for Markov bandit processes as was the original index-approach to solve the multi-armed bandit problem by Gittins [5] in 1979, but Gittins has generalised his index theorem to apply also in a *semi*-Markov setting [6]. In this and following chapters this *semi*-Markov approach is introduced.

The difference between Markov decision processes and semi-Markov decision processes is essentially that semi-Markov decision processes are generalisations of Markov decision processes that are processed in continuous time. More accurately, according to [11, §11] semi-Markov decision processes are MDPs in which

a) The decision maker is allowed, or required to choose actions whenever the system state changes (t_i arbitrary).

b) The system evolution is modelled in continuous time.

c) The time spent in a particular state is allowed to follow an arbitrary probability distribution.

However, for some of the Gittins applications the semi-Markov setup is too general, and further restrictions may be needed. One such a restriction is Condition A.

Condition A (Decision time restriction)**.** In a finite time there occurs at most a finite number of transitions.

The condition may be defined formally as follows. Let $\mathbf{P}(A|x,u)$ denote the probability that the state y of the process immediately after time t belongs to A, given that at time t the process is in state x and control u is applied. Moreover define $F(B|x,y,u)$ as the probability that the duration of the interval until the next decision time belongs to the set B, given that at time t the process is in state x, and control u is applied, leading to a transition to state y. Then the Condition A may be expressed in symbols as follows.

Condition A holds if there exists $\delta > 0$ and $\varepsilon > 0$ such that

$$\int_{\Theta} F((\delta, \infty)|x, y, u)\mathbf{P}(\mathrm{d}y|x, u) > \varepsilon \quad (x \in \Theta, u \in A(x)),$$

where (δ, ∞) denotes the unbounded open interval to the right of δ. In other words, this condition ensures that the probability, that the interval between decision times, and therefore between transitions, is greater than δ, is at least ε.

To introduce the definition of the Gittins index by Gittins himself it is necessary to introduce *families of alternative bandit processes* and their *freezing rules* first, as they are needed for the definition and play an important role in the Gittins book [6], which this text faithfully follows.

4.1 Families of bandit processes

A simple family of alternative bandit processes (SFABP) is a finite set of bandit processes, all of which can be continued at any decision time, that is, in a SFABP there is no restrictions on choosing different bandits at a decision time. In contrary, a *family of alternative bandit processes* (FABP) does not have such a character. The set of bandits that can be chosen to be continued at a certain decision time is termed the *control set*. The allocation rule that chooses a bandit to be continued and others to remain untouched at every decision time is called a *freezing rule*.

4.2 Freezing and stopping rules

Any policy for a family of alternative bandit processes realises a certain allocation rule termed freezing rule for a single bandit B in a family, the policy indicates whether a bandit process should be continued or remain unchanged at time t. *Freezing rule* is expressed with past-measurable random variables f_i $(i = 0, 1, 2, \ldots)$, where f_i indicates the total time for which control 0 is applied before the $(i+1)$th application of control 1 to a bandit process.

Because at all times one and only one bandit process is continued, the quantity f_i is also the total time during which control 1 has so far been applied to the other bandit processes in the family. Consequently there is no question of f_i taking a negative value.

Respectively, the continuation time is denoted by t_i. The quantity is the process time i.e. the total time when control 1 is applied to bandit B for the $(i+1)$th time. Thus the first application of control 1 lasts time t_1, the second $t_2 - t_1$ and ith application takes time of $t_i - t_{i-1}$.

The time points defined by t_i and f_i are pictured in Figure 4.1. Further the definitions allow one to define the expected total reward from a bandit process B under the freezing rule f in symbols

$$R_f(B) = \mathbf{E}\sum_{i=0}^{\infty} \beta^{t_i+f_i} r(t_i), \tag{4.1}$$

where time point $t_i + f_i$ is the total time that is elapsed when reward $r(t_i)$ accrued at process time t_i is gained.

If a freezing rule allocates a bandit only once starting from time zero, i.e. applies control 1 from time zero up to some time τ and control 0 always thereafter, equivalently in symbols $f_i = 0$ when t_i is smaller than some τ and $f_i = \infty$ if $t_i \geq \tau$ for all i, the freezing rule is termed a *stopping rule*, where τ is the corresponding *stopping time*. Such rules that do not apply control 1 to a bandit at all or instead apply the control always are also stopping rules having the corresponding stopping times $\tau = 0$ and $\tau = \infty$ respectively. Figure 4.1 illustrates also the difference between a freezing and a stopping rule.

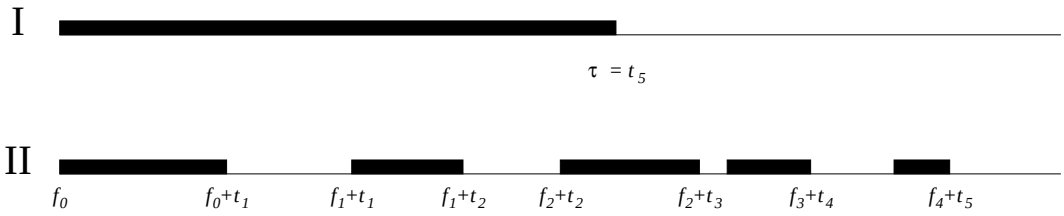


Figure 4.1: An illustration of continuation and freezing of a bandit under (I) a stopping rule and under (II) a freezing rule, with equal total continuation time τ.

4.3 Truncated processes and promotion rules

If a bandit process B follows an arbitrary freezing rule up to a certain time point τ but is never continued thereafter, although it should be continued according to the rule, the bandit process is called a *truncated* bandit process at time τ and is denoted with B^*. If such a truncated bandit is in a SFABP comprising B^* and another arbitrary bandit process A only, it follows that the freezing rule for A obeys condition $f_i \leq \tau$ for all i, because the truncated bandit process B^* is continued at most up to time τ.

By setting $p_i = f_i - \tau$ the expected total reward from A obeys an alternative expression

$$R_f(A) = \mathbf{E} \sum_{i=0}^{\infty} \beta^{t_i + p_i + \tau} r(t_i) = \mathbf{E} \left[\beta^{\tau} \mathbf{E} \sum_{i=0}^{\infty} \beta^{t_i + p_i} r(t_i) | \tau \right].$$

The inner expectation of the expression is conditional on the entire realisation of the bandit process B up to process time τ and is termed $R_p(A)$. The non-positive random variables p_i $(i = 0, 1, 2, \ldots)$ that are defined by the realisation of the bandit process B are past-measurable in terms of the bandit process A and they satisfy inequalities $-\tau \leq p_0 \leq p_1 \leq p_2 \leq \ldots \leq 0$, since $f_i \leq \tau$ for all i. It is clear that $(p_i + \tau)$s define a freezing rule for the bandit process A as well, but since the total freezing time of the latter rule does not exceed τ, the rule defined by p_is is distinguished from freezing rules in general by terming the rule as the *promotion rule p for A with respect to time* τ.

A promotion rule for which $p_i = 0$ $(i = 0, 1, 2, \ldots)$ is termed a *null* promotion rule (or a null freezing rule).

Figure 4.2 illustrates the promotion rules that would be induced to a bandit process that was in a family together with the bandit, on which the freezing rules of Figure 4.1 were applied.

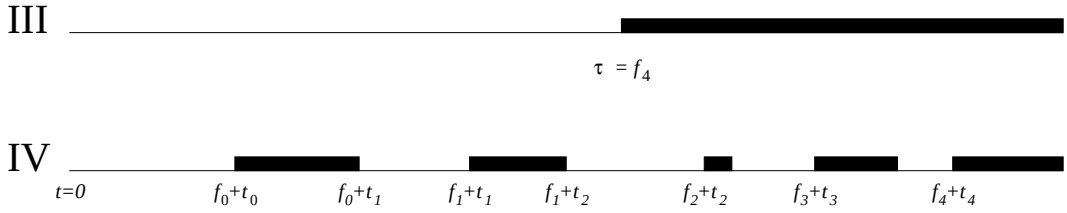


Figure 4.2: An illustration of continuation and freezing of a bandit under (III) a null promotion rule and under (IV) a promotion rule, both with respect to same time τ.

4.4 Gittins' definition of the index

In Section §3 it was mentioned that the Gittins index is actually the equivalent constant reward rate of a process, and the previous definition (3.3) was analogous with the statement. Gittins defines the index for a process with a freezing rule f as the fraction of the accrued reward and their discounts, which indeed is equal to the parameter that a equally attractive standard bandit process would have.

The expected total reward from a bandit process B under a freezing rule f was defined already by equation (4.1), but $W_f(B)$ defines the total accrued discounting that is necessary to evaluate the equivalent constant reward rate

$$W_f(B) = \mathbf{E} \sum_{i=0}^{\infty} \beta^{f_i} \int_{t_i}^{t_{i+1}} \beta^t \mathrm{d}t.$$

Thus the index for a freezing rule f is defined as

$$\nu_f(B) = \frac{R_f(B)}{W_f(B)}.$$

The Gittins index for a process B is defined as the supremum over the indices of all possible freezing rules that allocate the process first time at time zero

$$\nu'(B) = \sup_{\{f:f_0=0\}} \nu_f(B).$$

The same quantities are defined in a similar way for stopping rules,

$$R_\tau(B) = \mathbf{E} \sum_{t_i<\tau} \beta^{t_i} r(t_i), \quad W_\tau(B) = \mathbf{E} \int_0^\tau \beta^t \mathrm{d}t = \frac{1}{\log \beta} \mathbf{E}(\beta^\tau - 1)$$

$$\text{and} \qquad \nu_\tau(B) = \frac{R_\tau(B)}{W_\tau(B)}.$$

As soon will be discussed, the stopping rules are optimal among the freezing rules. Therefore the index is defined as the supremum over indices for stopping rules.

Definition 5 (The Gittins index for a semi-Markov decision process)**.**

$$\nu(B) = \sup_{\tau>0} \nu_\tau(B). \tag{4.2}$$

All the quantities mentioned in this section naturally have suppressed dependency upon the initial value of the process at time zero, $x(0)$. The definitions of $\nu_f(B)$ and $\nu_\tau(B)$ include the assumption that the process is continued at time zero. This restriction is due to the wish to rule out zero denominators from the fractions, and actually in the freezing rule case this does not imply loss of generality, because factor β^{f_0} would be a common factor of both the nominator and denominator.

Because stopping rules are special cases of freezing rules, it is evident that $\nu'(B) \geq \nu(B)$ holds. Actually the stopping rules are optimal among the freezing rules, i.e. one is allowed to replace the inequality with equality.

Lemma 5. *For any bandit process B, $\nu(B) = \nu'(B)$.*

Proof. The lemma is proven by observing a SFABP, which has only two bandit processes, namely bandit process B and a standard bandit process Λ. Let f be a freezing rule for B which freezes the bandit process at time zero but allocates the bandit at least once with a positive probability, in symbols

$$\mathbf{P}\{0 < f_0 < \infty\} > 0.$$

The proof then succeeds by showing

1. that a freezing rule f^* which is otherwise equal to f but the the first initial freezing period is removed and the application times of further controls are shifted to earlier time points respectively yields a better total reward. Or alternatively $f_0^* = \infty$, i.e. B is not continued at all.

2. that a freezing rule that once allocates the standard bandit process Λ never switches back to the bandit process B.

The latter statement is evident, because if the expected average reward rate from the standard bandit process Λ is at some time greater or at least equal to the rate from B, it always stays the same when only the standard bandit process is allocated, and B is never thereafter continued. The first statement is shown by comparing the reward produced by the SFABP $\{B, \Lambda\}$ and the standard bandit process Λ only, and concluding, that f^* is better than f when f_0^* is zero or infinite. For details see [6, Lemma 3.2]. □

Gittins shows in his book [6] that the index defined by (4.2) gives an optimal allocation policy for a simple family of alternative bandit processes.

Theorem 6 (The optimality of the index)**.** *A policy for a simple family of alternative bandit processes is optimal if it is an index policy with respect to $\nu(B_1, \cdot), \nu(B_2, \cdot), \ldots, \nu(B_n, \cdot)$.*

The proof of this optimality theorem and its generalisations beyond the bounds of SFABPs are discussed in the following sections up to an extent that is realistic and sensible in this work. The following two sections show some intermediate results in detail, where the next section compares null promotion rules to promotion rules in general and the following section shows that one cannot profit by changing the order of two bandit processes against the index $\nu(B_i)$. The full proof is provided in [6].

4.5 Comparison of promotion rules

The following lemma shows that it is always better to apply a null promotion rule to a SFABP when there are only two bandits, of which the other is a standard bandit process with parameter that equals the index of the non-standard bandit process.

Lemma 7. *For any bandit process B, the increase in the expected total reward which is caused by applying a promotion rule p instead of a null promotion rule (both defined with respect to the same time s) is less than or equal to the corresponding increase for a standard bandit process with the parameter $\nu(B)$.*

Note that an arbitrary promotion rule of a bandit cannot in general be applied to another bandit process, because different bandit processes may have different decision times. Fortunately for a standard bandit process every time point is a decision time, and therefore there is no problem with applying any promotion rules on the standard bandit process.

In following the idea of the proof is given. Reader, who wishes to see the explicit proof, is asked to refer to [6, Lemma 3.3].

Proof. If there was a family of two standard bandit processes, say Λ_1 and Λ_2, both with parameter $\nu(B)$, and they were gambled under a promotion rule, there would be no difference in the reward stream no matter what kind of promotion rule was used, because both bandits produce exactly the same rewards. It is supposed that a certain promotion rule is applied to one of the projects, say Λ_2, and the other project Λ_1 is allocated whenever the project Λ_2 isn't. With promotion rule p and null promotion rule 0 the same statement is expressed in symbols as

$$R_p(\{\Lambda_1, \Lambda_2\}) = R_0(\{\Lambda_1, \Lambda_2\}).$$

The lemma states that if Λ_2 is replaced with an arbitrary bandit process (on which the promotion rule can be applied), the equality sign is replaced with a "less than or equal"-sign. Thus the opposite statement

$$R_p(\{\Lambda_1, B\}) > R_0(\{\Lambda_1, B\})$$

contradicts the lemma.

If the inequality above is assumed, it means that there must exist a finite stopping time τ such that the equivalent constant reward rate from B up to process time τ is greater if the promotion rule p is followed instead of the null promotion rule. Let f be a freezing rule that implements this truncation at time τ by coinciding with p up to process time τ and then freezes the project for an infinite time. Thus $\nu_f(B) > \nu(B)$, but this contradicts Lemma 5 and therefore the statement of the lemma holds. □

4.6 Interchange argument for bandits

For two bandit processes B_1 and B_2 holds

Lemma 8 (Interchange argument). *If bandit processes B_1 and B_2 have indices $\nu(B_1)$ and $\nu(B_2)$ with $\nu(B_1) > \nu(B_2)$, and these are achieved with stopping times σ_1 and σ_2, the expected reward from selecting B_1 for time σ_1, and then B_2 for σ_2, is greater than from reversing order of selection.*

Proof. For the discount function W_{σ_i} holds

$$W_{\sigma_i}(B_i) = \mathbf{E}\int_0^{\sigma_i} \beta^t \mathrm{d}t = -\frac{1}{\log\beta}(1 - \mathbf{E}\beta^{\sigma_i}).$$

Therefore

$$\begin{aligned} \frac{R_{\sigma_1}(B_1)}{W_{\sigma_1}(B_1)} &= \nu(B_1) > \nu(B_2) = \frac{R_{\sigma_2}(B_2)}{W_{\sigma_1}(B_2)} \\ \Rightarrow \quad \frac{(-\log\beta)(R_{\sigma_1}(B_1))}{1 - \mathbf{E}\beta^{\sigma_1}} &> \frac{(-\log\beta)(R_{\sigma_2}(B_2))}{1 - \mathbf{E}\beta^{\sigma_2}}, \end{aligned}$$

and dividing by $-\log\beta > 0$ follows

$$\frac{R_{\sigma_1}(B_1)}{1 - \mathbf{E}\beta^{\sigma_1}} > \frac{R_{\sigma_2}(B_2)}{1 - \mathbf{E}\beta^{\sigma_2}},$$

and it is concluded

$$R_{\sigma_1}(B_1) + \mathbf{E}\beta^{\sigma_1} R_{\sigma_2}(B_2) > R_{\sigma_2}(B_2) + \mathbf{E}\beta^{\sigma_2} R_{\sigma_1}(B_1).$$

which is exactly the inequality that was to be proven. □

4.7 The index theorem for families of alternative bandit processes

This section discusses the proof of the Index theorem for SFABP (Theorem 6). The theorem is proven indirectly by proving the index theorem first for *forwards induction policies* and then proving them equal to the index defined in (4.2).

4.7.1 Forwards induction policy

A *myopic policy* is a policy that maximises the equivalent reward up to the next decision time, but instead a *forwards induction policy* is a policy which at each decision time maximises the equivalent constant reward rate up to an arbitrary stopping time. Neither is in general optimal, because the task is to optimise the reward over infinite time. However, the forwards induction policy generates an optimal policy for simple families of alternative bandit processes. In this section some necessary notation is introduced followed by an auxiliary theorem, which is also needed for other purposes.

Suppose that there is given a decision process D with a policy g. This decision process is considered as a bandit process by assigning the freeze control 0 to it, and requiring that at each decision time either the control 0 or control given by g is applied. If g is deterministic, stationary and Markov, the application of the continuation control 1 to the decision process D equals to the application of control $g(x)$ to D, where x is the state of the decision process D at the time of application. With this setting the process D with policy g can be interpreted as a bandit process and it is denoted with D_g.

Moreover further notation related to the bandit process D_g is introduced. Let the quantities t_i, $x_g(t)$, $r_g(t)$ and $R_g(D)$ have the same definitions as the corresponding quantities of a bandit process, where D_g is in the place of B. Thus the quantities typical for a bandit process obey now following expressions: $R(D) = \sup_g R_g(D)$, $R_{g\tau}(D) = R_\tau(D_g)$, $W_{g\tau}(D) = W_\tau(D_g)$, $\nu_{g\tau}(D) = R_{g\tau}(D)/W_{g\tau}(D)$, $\nu_g(D) = \sup_{\tau>0} \nu_{g\tau}(D)$, and $\nu(D) = \sup_g \nu_g(D)$. Similarly $R_{gf}(D) = R_f(D_g)$, etc. The index for this decision process $\nu(D, x)$ may be simplified, when context makes the relevant decision process apparent, with $\nu(x)$

To give an explicit definition of the forwards induction policy it is necessary to introduce the following theorem.

Theorem 9. *For semi-Markov decision process D satisfying Condition A, and for which the control set is always finite:*

(i) for any $\xi \in \Theta, \nu(\xi) = \nu_{g\tau}(\xi))$ for some deterministic stationary Markov policy g and stopping time $\tau > 0$;

(ii) $\nu(\cdot)$ is an $\mathscr{X}$-measurable function;

(iii) the stopping set $\Theta_0 (\in \mathscr{X})$ defining the τ described in (i) *may be chosen to be either $\{x : \nu(x) < \nu(\xi)\}$ or $\{x : \nu(x) \leq \nu(\xi)\} - \{\xi\}$;*

also for any g and τ as described in (i),

(iv) $\mathbf{P}\{\nu_{g,\tau-t}(x(t)) \geq \nu(\xi)$ for every decision time $t < \tau | x(0) = \xi\} = 1$, and

(v) $\mathbf{P}\{\nu(x(\tau)) > \nu(\xi) | x(0) = \xi\} = 0$, where, for any t, $x(t)$ is the state of D at time t under policy g.

The proof of this Theorem is bypassed in this work. The reader who wants to study the proof is asked to refer to [6, Theorems 2.1, 2.2 and 3.4] as well to an introductory journal article by D. Blackwell [3].

Theorem 9 allows one to define the forwards induction policy formally. Let $t_0 = 0, t_1, t_2, \ldots$ be the decision times of a semi-Markov decision process D satisfying Condition A. The *forwards induction policy* (FI) is a policy that applies at each decision time t_i policy g_i that satisfies $\nu_{g_i}(x(t_i)) = \nu(x(t_i))$, i.e. chooses the control that is part of a policy that gives the greatest index for D. The control applied at time t_i may be referred to as u_i for later use.

A policy that continues a decision process always until the optimal stopping time, instead of reconsidering the decision at every decision time, but otherwise follows

the principle of the forwards induction policy, is termed *modified* forwards induction policy (FI*). Clearly both forward induction policies can be applied to bandit processes as well.

Remark 1. Modified forwards induction policy fulfils automatically the definition of *index consistence* of a policy defined by Weber [17, p. 1028]. A policy is said to be *index consistent* if a decision process that is once chosen to be continued is continued without interruption until its Gittins index drops below its initial value. Note though, a policy that is index consistent is not necessarily forwards induction policy at all, since the definition does not include any conditions on the index optimality.

4.7.2 Optimal policies

The next section shows that the Gittins index is an optimal policy for a simple family of alternative bandit processes. Before the proof it is necessary to discuss what the optimal policies are and to introduce some additional notation. First of all the Gittins index for a SFABP is defined.

Definition 6 (The Gittins index for a SFABP)**.** *The Gittins index for a SFABP $\mathscr{F}$ is defined similarly as for a single bandit* (4.2) *as*

$$\nu(\mathscr{F}) = \sup_{g,\tau} \frac{R_{g\tau}(\mathscr{F})}{W_{g\tau}(\mathscr{F})}.$$

The index is defined as the best possible expected constant reward rate of the family under an arbitrary policy g.

The following lemma shows that there isn't an optimal policy that applies control 1 to a bandit process after the index of the bandit drops below its initial value. The lemma has to be preceded with introduction of additional notation.

A decision or bandit process that is truncated would generally be still continued after the truncation time, say τ. The remaining part of the process is termed the *residual* decision or bandit process from that time. Because the reward of the truncated process following policy g is $R_{g\tau}(D)$ and the total reward from the process would be $R_g(D)$, it is clear that the reward of the residual process is $R_g(D) - R_{g\tau}(D)$.

Then consider some bandit B in a SFABP $\mathscr{F}$ when policy g is applied to it. Depending on the realisations of all the other bandit processes in $\mathscr{F}$ a certain freezing rule will be applied to B. If the dependence upon the other bandit processes is ignored, the freezing rule to the bandit can be interpreted as a random freezing rule, the randomising arising from the random realisations of all the other bandit processes. Such a random freezing rule is termed *the freezing rule for B defined by g.*

In addition, two other types of notation for random freezing rules are defined. Let f be a random freezing rule that is applied to bandit B. A freezing rule that coincides with f after a truncation time σ but applies control 0 before the truncation time is denoted with f_+ and similarly a freezing rule that coincides with the freezing rule f up to the truncation time σ and applies control 0 thereafter is denoted with f_-. This notion is used in the proof of the following lemma.

For a non-standard bandit process B, initially in state ξ, let successive decision times occur at process time $t_0(=0), t_1, t_2, \ldots$. Let σ be the earliest decision time t_i for which $\nu(B, x(t_i)) < \nu(B, \xi)$.

Lemma 10. *If f is a freezing rule for which $\mathbf{P}\{\exists i$ such that $\sigma \leq t_i < \infty$ and $f_i < \infty\} > 0$, then $\nu_f(B, \xi) < \nu(B, \xi)$.*

This means that the probability that there exists an optimal freezing rule that applies the continuation control after the time point where the index drops below its initial value is zero.

Proof. The index for the freezing rule may be expanded as follows

$$\nu_f(B, \xi) = \frac{R_f(B, \xi)}{W_f(B, \xi)} = \frac{R_{f_-}(B, \xi) + \mathbf{E}R_{f_+}(B, x(\sigma))}{W_{f_-}(B, \xi) + \mathbf{E}W_{f_+}(B, x(\sigma))}. \tag{4.3}$$

But because the freezing rule f_- is at most optimal

$$\frac{R_{f_-}(B, \xi)}{W_{f_-}(B, \xi)} = \nu_{f_-}(B, \xi) \leq \nu(B, \xi),$$

and the residual freezing rule worse than the initial index

$$\frac{R_{f_+}(B, x(\sigma))}{W_{f_+}(B, x(\sigma))} = \nu_{f_+}(B, x(\sigma)) \leq \nu(B, x(\sigma)) < \nu(B, \xi),$$

provided $W_{f_+}(B, x(\sigma)) > 0$. This condition follows directly from the assumptions of the lemma, and the lemma therefore follows from equation (4.3). □

4.7.3 Optimality of the index policy

Before the fundamental index theorem, the index theorem for a simple family of alternative bandit processes, it is necessary to introduce additional notation. Suppose that there is a family $\mathscr{F}$ of alternative bandit processes $B_1, B_2, \ldots, B_n$ with a policy g applied to it. Further suppose that the process is stopped at time τ and let τ_j $(j = 1, 2, \ldots, n)$ be the process times of bandit process B_j at that time. Denote the freezing rule for B_j defined by g with f_j and let t_i^j $(i = 0, 1, 2, \ldots)$ be the process time of B_j at time of the ith application of control 1 to the process, and with $f_i^j + t_i^j$ the true time when this occurs under f^j. Like in the previous section let f_-^j and f_+^j be the freezing rules partly coinciding with f^j that split the bandit processes to two parts with respect to process time τ_j.

With this notation the expected total reward from the truncation of $\mathscr{F}$ at time τ under policy g can be expressed with the following sum

$$R_{g\tau}(\mathscr{F}) = \sum_{j=1}^{n} R_{f_-^j}(B_j).$$

Equivalently the expected reward from the residual FABP from time τ onwards under the same policy g can be expressed by

$$R_g(\mathscr{F}) - R_{g\tau}(\mathscr{F}) = \sum_{j=1}^{n} [R_{f^j}(B_j) - R_{f_-^j}(B_j)] = \sum_{j=1}^{n} \mathbf{E}R_{f_+^j}(B_j, x_j(\tau_j)).$$

This discussion reflects the proof of the following index theorem.

Theorem 11 (The Index Theorem for a SFABP)**.** *The classes of index policies, forwards induction policies and optimal policies are identical for a simple family $\mathscr{F}$ of alternative bandit processes B_j $(j = 1, 2, \ldots, n)$.*

Essentially the idea of the proof of this theorem resembles the principle of the policy iteration algorithm (§2.5). The main idea of the proof is to show that it is always better to use a policy that is obtained by altering the first decision of an arbitrary policy to conform the Gittins index than the arbitrary policy itself. However, before the main proof of the Theorem the following lemma, which proves the equality of the index policies and forwards induction policies, is needed.

Lemma 12. *(i) The index for the family is the index of the best bandit, $\nu(\mathscr{F}) = \max_j \nu(B_j)$.*

(ii) If f_-^j is the freezing rule for $B_j (j = 1, 2, \ldots, n)$ defined by a policy γ for $\mathscr{F}$ and truncated at time σ, and $\nu_{\gamma\sigma}(\mathscr{F}) = \nu(\mathscr{F})$, then $\nu_{f_-^j}(B_j) = \nu(\mathscr{F})$ for all j such that $W_{f_-^j}(B_j) > 0$.

Proof. For the total reward of the SFABP $\mathscr{F}$ under policy γ and truncation at time σ holds

$$\begin{aligned} R_{\gamma\sigma}(\mathscr{F}) &= \sum_{j=1}^{n} R_{f_-^j}(B_j) = \sum_{j=1}^{n} W_{f_-^j}(B_j)\nu_{f_-^j}(B_j) \\ &\leq \sum_{j=1}^{n} W_{f_-^j}(B_j)\nu(B_j) \qquad (4.4) \\ &\leq \max_i \nu(B_i) \sum_{j=1}^{n} W_{f_-^j}(B_j) = \max_i \nu(B_i) W_{\gamma\sigma}(\mathscr{F}). \end{aligned}$$

Hence

$$\nu(\mathscr{F}) = \sup_{g,\tau} \nu_{g\tau}(\mathscr{F}) = \sup_{g,\tau} \frac{R_{g\tau}(\mathscr{F})}{W_{g\tau}(\mathscr{F})} = \frac{R_{\gamma\sigma}(\mathscr{F})}{W_{\gamma\sigma}(\mathscr{F})} \leq \max_i \nu(B_i).$$

However by part (i) of Theorem 9 it follows that there is a stopping time τ_k for B_k for which

$$\max_i \nu(B_i) = \nu(B_k) = \nu_{\tau_k}(B_k).$$

If g_k is the policy for $\mathscr{F}$ which applies the continuation control to bandit process B_k at all times, then

$$\nu(\mathscr{F}) \geq \nu_{g_k\tau_k}(\mathscr{F}) = \nu_{\tau_k}(B_k) = \max_i \nu(B_i).$$

Consequently $\nu(\mathscr{F}) = \max_i \nu(B_i)$ as required, which allows one to replace the inequalities in (4.4) by equalities. The second part of the lemma follows immediately, because if there was a freezing rule f_-^j for which $\nu_{f_-^j}(B_j) > \nu(\mathscr{F})$ it would contradict the first part of the lemma. On the other hand, if there is a freezing rule f_-^j for which $\nu_{f_-^j}(B_j) < \nu(\mathscr{F})$, it will not be allocated, as there are one or more bandits, that have index $\nu(\mathscr{F})$. □

Proof of Theorem 11. First define a SFABP $\mathscr{G}$ as the residual SFABP obtained by truncating $\mathscr{F}$ at time τ_k when policy g_k is used. If τ_k is infinite consider $\mathscr{G}$ to include the $n-1$ bandits that are left when B_k is removed from $\mathscr{F}$. This is not possible if there is only one bandit in the family, but the theorem holds for a family including only one bandit trivially, so it may be assumed that $n > 1$.

When $\mathscr{G}$ is defined as above, it is in a way complementary to a bandit process B_k^* that is obtained by truncating B_k at time τ_k, because whenever the continuation control 1 is applied to a bandit process in $\mathscr{F}$ it is applied either to B_k^* or to one of the bandit processes in $\mathscr{G}$. Thus one is allowed to divide the expected reward of $\mathscr{F}$ into two parts when an arbitrary policy g is applied to it. The division is assisted with an indicator function $I_g(t)$ that is equal to 1 if and only if control 1 is applied to B_k^* at time t and zero otherwise. Thus

$$R_g(\mathscr{F}) = \mathbf{E}\sum_{i=0}^{\infty}\beta^{t_i}I_g(t_i)r_g(t_i) + \mathbf{E}\sum_{i=0}^{\infty}\beta^{t_i}(1-I_g(t_i))r_g(t_i). \tag{4.5}$$

The first component is the expected reward $R_f(B_k)$ resulting from the bandit process B_k under the freezing rule f which is defined by applying policy g to $\mathscr{F}$ and truncating at process time τ_k. The second component is the reward resulting from the remaining process and it may alternatively be expressed as

$$\mathbf{E}\left[\mathbf{E}\sum_{i=0}^{\infty}\beta^{t_i}(1-I_g(t_i))r_g(t_i)\Big|B_k^*\right],$$

where the inner expectation is conditional on the entire realisation of B_k^*. This conditioning fixes the random SFABP $\mathscr{G}$ and with the policy g for $\mathscr{F}$ defines a policy h for $\mathscr{G}$ and a promotion rule p for the family $(\mathscr{G}, h)$ with respect to time τ_k. Thus the second component in equation (4.5) may alternatively be denoted with $\mathbf{E}R_{hp}(\mathscr{G})$, giving the following expression for (4.5)

$$R_g(\mathscr{F}) = R_f(B_k) + \mathbf{E}R_{hp}(\mathscr{G}). \tag{4.6}$$

Consider now a policy for $\mathscr{F}$ that is a slightly modified version of the original policy g. The policy applies first control 1 to B_k until process time τ_k, at which the state of $\mathscr{F}$ coincides with the initial state of $\mathscr{G}$, provided $\tau_k < \infty$. From that time onwards the policy is defined to coincide with h. This composite policy is denoted with $*g$ since it coincides with a FI* policy for the first stage, up to time τ_k. Denoting by 0 the null promotion rule for $(\mathscr{G}, h)$ with respect to τ_k, the reward of the policy $*g$ can again be divided in two parts

$$R_{*g}(\mathscr{F}) = R_{\tau_k}(B_k) + \mathbf{E}R_{h0}(\mathscr{G}). \tag{4.7}$$

The first step in the proof of the optimality of the index policies is to show that the reward above is at least as good as the reward yielded by policy g, i.e.

$$R_{*g}(\mathscr{F}) \geq R_g(\mathscr{F}). \tag{4.8}$$

From part (i) of Lemma 12 it follows that

$$\begin{aligned} R_f(B_k) &= W_f(B_k)\nu_f(B_k) \\ &\leq W_f(B_k)\nu_{\tau_k}(B_k) = W_f(B_k)\nu(B_k) = W_f(B_k)\nu(\mathscr{F}), \end{aligned} \tag{4.9}$$

and

$$R_{\tau_k}(B_k) = W_{\tau_k}(B_k)\nu_{\tau_k}(B_k) = W_{\tau_k}(B_k)\nu(\mathscr{F}). \tag{4.10}$$

Lemma 7 states

$$R_{hp}(\mathscr{G}) - R_{h0}(\mathscr{G}) \leq [W_{hp}(\mathscr{G}) - W_{h0}(\mathscr{G})]\nu(\mathscr{G}). \tag{4.11}$$

From the definition of $\mathscr{G}$ and part (v) of Theorem 9, it follows that $\nu(\mathscr{G}) \leq \nu(\mathscr{F})$, and since the multiplier $[W_{hp}(\mathscr{G}) - W_{h0}(\mathscr{G})]$ is clearly non-negative one may replace $\nu(\mathscr{G})$ with $\nu(\mathscr{F})$ in the inequality (4.11) giving

$$R_{hp}(\mathscr{G}) - R_{h0} \leq [W_{hp}(\mathscr{G}) - W_{h0}(\mathscr{G})]\nu(\mathscr{F}). \tag{4.12}$$

Now by subtracting (4.6) from (4.7) and by substituting (4.9), (4.10) and (4.12) to the result gives

$$R_g(\mathscr{F}) - R_{*g}(\mathscr{F}) \leq [W_f(B_k) - W_{\tau_k}(B_k)]\nu(\mathscr{F}) + \mathbf{E}[W_{hp}(\mathscr{G}) - W_{h0}(\mathscr{G})]\nu(\mathscr{F}).$$

However by noticing that

$$\begin{aligned} & W_f(B_k) + \mathbf{E}W_{hp}(\mathscr{G}) = W_{\tau_k}(B_k) + \mathbf{E}W_{h0}(\mathscr{G}) = \int_0^\infty \beta^t \mathrm{d}t \\ \Leftrightarrow \quad & W_f(B_k) - W_{\tau_k}(B_k) = -[\mathbf{E}W_{hp}(\mathscr{G}) - \mathbf{E}W_{h0}(\mathscr{G})], \end{aligned}$$

the statement of (4.8) follows immediately, the policy that first allocates the bandit with the best index and then continues some fixed policy instead of following the fixed policy from the beginning is always a good choice.

To complete the proof of the optimality of the index policy the following theorem on the existence of an optimal policy is needed.

Theorem 13. *For a semi-Markov decision process D satisfying Conditions A and B, and for which the control set $A(x)$ is finite for all $x \in \Theta$ holds*

(i) There is at least one optimal policy which is deterministic, stationary and Markov.

(ii) A policy is optimal if and only if, for every $x \in \Theta$, the control which it chooses in state x is such as to achieve the maximum on the right-hand side of the dynamic programming equation, which is a modification of (2.9) for semi-Markov decision processes.

The necessary Condition B restricts the behaviour of the discounted reward on $t \to \infty$ limit.

Condition B. A semi-Markov decision process D satisfies Condition B if $\mathbf{E}[\beta^{\lceil t \rceil} R_g(D, x_g(\lceil t \rceil))|x(0) = x]$ tends to zero uniformly over all policies g as t tends to infinity, for all $x \in \Theta$.

In the expression the notation $\lceil t \rceil$ does not represent the normal ceiling but the next decision time after time t. The proof of Theorem 13 is not included, but the Theorem is part of a Theorem in the book of Gittins [6, Theorem 2.2], which is further based on the previous Theorem in the book [6, Theorem 2.1] and proved by D. Blackwell [3].

Theorem 13 cannot be applied to the proof of Theorem 11 directly, because if $\mathscr{F}$ contains one or more standard bandits, the family does not obey Condition A. However, this restriction may be lifted by observing the following. In the first place there is clearly no point in continuing any standard bandit process except the one with the highest parameter value and if such a bandit is once chosen for continuation, it is continued permanently from that time onwards, as the other bandits will not get better as they are frozen (see [6, Lemma 3.2]). Hence attention may be restricted to policies that have the preceding property. This restriction has the effect of ensuring Condition A, and Theorem 13 may therefore be used.

To show that *only* index policies are optimal it is sufficient to show that (4.8) holds with strict inequality unless the bandit process selected by g at time zero is one with an index value at least as great as any other. If g does not have this property then $W_{hp}(\mathscr{G}) > W_{h0}(\mathscr{G})$ and, from part (ii) of Lemma 12, $\nu(\mathscr{F}) > \nu(\mathscr{G})$. Thus (4.12) holds with strict inequality, and therefore so does (4.8). This concludes the proof of Theorem 11. □

Now the index theorem is proven for simple families of alternative bandit processes, but that is not enough to conclude the optimality of index theorem for one-server queueing system. In the following chapter the application of the Gittins index to a queueing system is discussed further.

Chapter 5

Scheduling an M/G/1-queue with the Gittins index

The Gittins index not only provides an ultimate solution to the multi-armed bandit problem, but the index can also be used to solve other similar tasks. The purpose of this work is to convince the reader that the Gittins index may successfully be used to schedule a one-server queueing system.

The customers in a queueing system may be interpreted as *jobs*, which are certain bandit processes. The jobs and their indices will be introduced in the first two sections of this chapter. The latter sections concentrate on generalising the index theorem for SFABP, in order to make it applicable on queueing systems.

5.1 Jobs

Jobs are simple bandit processes that produce a single positive reward V at a random time and no further rewards. A simple family of such bandit processes defines a version of the well-known problem of scheduling a number of jobs on a single machine. The reward produced by such a family under a certain policy is

$$\sum_{i=1}^{n} V_i e^{-\gamma t_i},$$

where V_i is the completion reward of the project completed at time t_i that is policy dependent. The factor $e^{-\gamma t_i} = \beta^{t_i}$ is the discount factor and γ is the corresponding *discount parameter*, which reduces the reward relative to the completion time of the process.

If the discount parameter γ is considered small, the expression above may be well approximated with a first order Taylor-expansion

$$\sum_{i=1}^{n} V_i e^{-\gamma t_i} = \sum_{i=1}^{n} V_i - \gamma \sum_{i=1}^{n} t_i V_i + O(\gamma^2). \tag{5.1}$$

From the equation it can be seen that the objective to optimise the reward is equal to the problem of minimising the first-order term $\sum_{i=1}^{n} t_i V_i$, because the first term is

constant and does not depend upon the policy as every job has to be served sooner or later. The expectation of term $\sum_{i=1}^{n} t_i V_i$ is called the *expected weighted flow-time* (EWFT).

5.1.1 The Gittins index for a job

Gittins derives the general expression for the Gittins index by comparing jobs to standard bandit processes. Consider that the standard bandit process yield a continuous reward stream λ, with a time-dependent discount factor β^t.

Secondly the job process yields a reward V at completion time and which is likewise discounted with parameter γ and no further rewards. The time that is needed to complete the job is distributed with density function $f(t)$, having the distribution function $F(t)$ respectively. The question is, should one start with the job and after the completion change to the standard bandit process, or allocate the standard bandit process only, leaving the job untouched.

The job yields a reward $Ve^{-\gamma s}$ in time interval $[s, s + \mathrm{d}s)$ with probability $f(s)\mathrm{d}s$, so the expected accrued reward up to time t obeys equation

$$\int_0^t V f(s) e^{-\gamma s} \mathrm{d}s. \tag{5.2}$$

The reward that the standard bandit process would give in time interval $[s, s+\mathrm{d}s)$ is $\lambda e^{-\gamma s}\mathrm{d}s$ with probability $[1 - F(s)]$, because the probability that the job completes after time s is $[1-F(s)]$. Therefore the reward given by the standard bandit process is

$$\int_0^t \lambda [1 - F(s)] e^{-\gamma s} \mathrm{d}s. \tag{5.3}$$

If there is a time $t > 0$ that expression (5.2) is greater than expression (5.3), then it is best to start to work first with the job. The condition is identical with inequality

$$\lambda < \sup_{t>0} \frac{V \int_0^t f(s) e^{-\gamma s} \mathrm{d}s}{\int_0^t [1 - F(s)] e^{-\gamma s} \mathrm{d}s}. \tag{5.4}$$

If the job has been continued already for time x without successful completion, the density $f(s)$ and the distribution function $F(s)$ must be replaced with the conditional functions $f(s)/[1 - F(x)]$ and $[F(s) - F(x)]/[1 - F(x)]$ respectively. Substituting these in (5.4) gives the definition of the Gittins index for a job, which has been already continued for time x.

Definition 7 (The Gittins index for a job with age x)**.**

$$\nu(x) = \sup_{t>x} \frac{V \int_x^t f(s) e^{-\gamma s} \mathrm{d}s}{\int_x^t [1 - F(s)] e^{-\gamma s} \mathrm{d}s}, \tag{5.5}$$

The definition can be easily simplified for processes without discounting by setting $\gamma = 0$

$$\nu(x) = \sup_{t>x} \frac{V \int_x^t f(s)\mathrm{d}s}{\int_x^t [1 - F(s)]\mathrm{d}s}. \tag{5.6}$$

Note that also this definition of the Gittins index is interpretable as the maximised equivalent constant reward rate of the job in state x.

5.2 The index as a scheduling rule for a queueing system

The interchange argument of Lemma 8 together with the index Theorem for a SFABP (Theorem 11) can be generalised for different kinds of bandits, including jobs. The generalisations that aim at modelling a queueing system with jobs are handled in the following sections.

One of the generalisations is to allow the job system to have arrivals, that is, new jobs to arrive in the system. This allows the jobs to be identified as jobs of an *M/G/1-queue*, and the Gittins index becomes a *scheduling rule* for the queueing system. If the reader is not familiar with the terms mentioned, a short introduction to queueing systems is provided in Appendix A.

Essentially the setting is to consider a system of jobs allowing simultaneous Poisson arrivals when discounting is almost completely ignored, i.e. the discounting parameter γ is let to approach zero. Suppose that there are n classes of jobs arriving at Poisson rates λ_i $(i = 1, 2, \ldots, n)$, and that class i jobs have independent service times all with the distribution $F_i(t) = \mathbf{P}(\sigma_i \leq t)$, each of them costing c_i per unit time between its arrival time and the time of completion of its service. Furthermore it is assumed that the queueing system is stable, i.e. the *traffic intensity* is at most one $\sum_i \lambda_i \mu_i < 1$, where μ_i is the mean of distribution F_i, the mean service time.

Naturally it is preferable to minimise the costs of the queueing system by choosing an optimal scheduling rule. With the notation of the previous paragraph the costs of the queueing system with n jobs obey equation

$$C = \mathbf{E} \sum_{i=1}^{n} c_i t_i, \tag{5.7}$$

where t_i is the realised completion time of the job. Thus the objective is to find a policy that minimises the sum.

In Section §5.1 it was noticed that an optimal policy with a small discounting parameter γ minimises the expected weighted flow-time (EWFT) of jobs that have rewards of V_i and completion times t_i, that is

$$R = \sum_{i=1}^{n} t_i V_i. \tag{5.8}$$

By parallelling the reward V_i to the cost c_i by setting $V_i = -c_i$ the expressions (5.7) and (5.8) obey exactly the same form. Thus one could easily assume, that a Gittins

index policy could be an optimal scheduling rule for a queueing system described in the beginning of this section. This is, indeed, the case.

Gittins claims in his book [6, §3.13] that the Gittins index minimises the expected weighted flow time (5.7 and 5.8) in a busy period of a M/G/1 queueing system under a certain condition. This condition requires that there must exist $\varepsilon > 0$ such that the Gittins index policy obtained with every discount parameter $\gamma < \varepsilon$ is the same. This condition is discussed in detail in [6, Condition C†].

Theorem 14. *The expected weighted flow-time (EWFT) in a busy period of a M/G/1-queue with job classes i $(= 1, 2, \ldots, n)$ is minimised by the index policy defined by the $\gamma = 0$ indices $\nu_i(x)$ for the individual jobs, ignoring arrivals.*

It is rather annoying that the Theorem 14 could only be established under previously mentioned condition. Fortunately von Olivier has already proven the Theorem with an discount-free approach [8], hence the condition is not actually needed. But foremost, to convince oneself of the optimality of this scheduling rule of an M/G/1-queue it is necessary to understand, why the Gittins index is the optimal index for families of jobs.

5.3 Generalisations of the index policy for a SFABP

The generalisations that are needed to enhance a SFABP to function like a queueing system are following. Firstly it is necessary to generalise a bandit process to a *bandit superprocess*, which itself is a Markov decision process. This is followed by implementation of *precedence constraints* between bandits and the last generalisation allows the system to have arrivals. These generalisations are discussed in the following sections in the respective order to an extent that is feasible for this thesis.

5.3.1 Bandit superprocesses and families of superprocesses

The definition of a bandit process is too narrow for queueing system purposes, because a bandit process cannot represent a job in a queueing system with arrivals "as is", but a generalisation of the process is needed. The modification of a bandit process allows a bandit to have more controls besides the continuation and freeze control.

Suppose that there is a set of decision processes (which may have a larger control set than a bandit process) $\{D_1, D_2, \ldots, D_n\}$ all with the same discount factor β. Every decision process D_i in state x_i has a control set $A_i(x_i)$, of which every control produces some reward and changes in the state of the process. A slightly modified decision process called *bandit superprocess* or simply *superprocess* S_i is formed from D_i by adding the freeze control 0 to the control set $A_i(x_i)$. The freeze control functions on the decision process exactly as on the bandit processes, it leaves the state unchanged and produces zero reward.

Analogously to bandit families a family of decision processes is formed from S_is by requiring that the freeze control is always applied to all but one of the processes in

the set $\{S_1, S_2, \ldots, S_n\}$, the only difference is that it is not sufficient to only choose a process for continuation, but also a control that is not the freeze control, from the control set. Such a family is termed a *family of alternative superprocesses* (FAS), and if there is no restriction on the set of superprocesses available for selection, the family is termed *simple family of alternative superprocesses* (SFAS).

The definition of an index policy (2.21) is not adequate for families of alternative superprocesses, as the definition assumes that a bandit has only one continuation control. Therefore an index policy for superprocesses chooses not only the superprocess with the greatest index but also the control with the greatest index value.

Definition 8 (Index policy for superprocesses)**.** *An index policy for a FAS $\mathscr{F}$ with respect to $\mu_1, \mu_2, \ldots, \mu_n$ is one which at any decision time when S_i is in state x_i $(i = 1, 2, \ldots, n)$ applies control a_j to superprocess S_j, for some S_j and a_j such that*

$$\mu_j(x_j, a_j) = \max_{\{i, a_i \in A_i(x_i)\}} \mu_i(x_i, a_i).$$

The Gittins index for a bandit process was defined with maximisation over the stopping rules (4.2), but in case of superprocesses the definition of the Gittins index has to be modified. The obvious candidate for the index for a β-discounted set $\mathscr{S}_\beta$ of superprocesses is

Definition 9 (The Gittins index for superprocesses)**.**

$$\nu(S, x, a) = \sup_{\{g : g(x) = a\}} \nu_g(D, x) \quad (S \in \mathscr{S}_\beta, x \in \Theta_D, a \in A_D(x)). \tag{5.9}$$

The index for a superprocess is determined by choosing for every state x and action a the supremum of the indices for decision processes D_g which are formed by removing the freeze control from the superprocesses S_g, where g is an arbitrary policy that applies control a in state x. The forwards induction and modified forwards induction policies are defined analogously to the corresponding definitions for bandit processes. Instead of choosing only optimal bandit, one chooses also the optimal control.

Remark 2. Gittins remarks that the definition of the index for superprocesses may not be applicable to an arbitrary set $\mathscr{S}_\beta$ [6, §3.7].

If the family of β-discounted standard bandit processes are included in $\mathscr{S}_\beta$, the index for the family is indeed the index defined by (5.9).

Theorem 15. *If $\mathscr{S}_\beta \supset \Lambda_\beta$ and an index exists for $\mathscr{S}_\beta$ then it must be a strictly increasing function of $\nu(S, x, a)$.*

The proof of the theorem succeeds roughly by calibrating the family $\mathscr{S}_\beta$ with the standard bandit processes of Λ_β [6, Theorem 3.11]. Further it can be shown that the index of a FAS equals the maximum of the indices of the superprocesses in the family and that a policy for $\mathscr{F}$ is an index policy if and only if it is a forwards induction policy. The first statement is proven as in the proof of Lemma 12 and the second statement follows partially from the first statement. For details see [6, Lemma 3.12].

Remark 2 leaves the question open, which superprocesses may be included in $\mathscr{S}_\beta$, still preserving the index (5.9) as an optimal policy? Trivially, if $\mathscr{S}_\beta$ is the family

of β-discounted bandits $\mathscr{B}_\beta$, then the index obeys the form $\nu(S,x,a) = \nu(S,x,1) = \nu(S,x)$ as there are no other continuation controls. Thus $\nu(\cdot,\cdot,\cdot)$ is an index for $\mathscr{S}_\beta$ by Theorem 11.

However, $\mathscr{S}_\beta$ can be extended with many non-bandit type superprocesses, but superprocesses, which lead to optimal policies that do not conform the index $\nu(\cdot,\cdot,\cdot)$ have to be excluded from $\mathscr{S}_\beta$. Gittins illustrates the possibility of such a conflict with a simple example [6, Example 3.13] that is also shortly explained below.

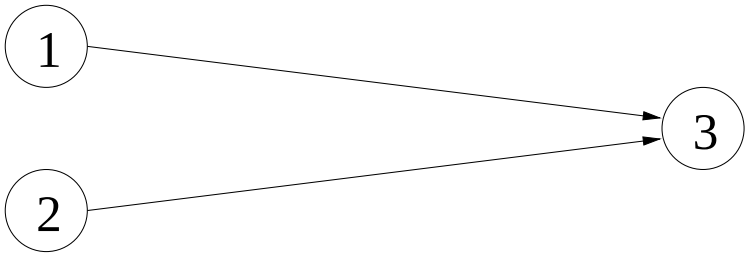


Figure 5.1: Precedence constraints for Example 3

Example 3. Suppose that there are three jobs that have the above illustrated *precedence constraints*, so that job 3 cannot be started until jobs 1 and 2 have been completed. Jobs 1, 2 and 3 require, respectively 1, 2 and 1 unit(s) of service time up to the first decision time after allocation of the job, and then terminate with probabilities $\frac{1}{2}$, 1 and 1 yielding rewards 0, 1 and $M(> 1\frac{2}{3})$. After these times every job behaves like a standard bandit process with parameter 0, whether the process has terminated or not. A reward that is gained at time t is discounted with the factor $(\frac{1}{2})^t$.

The three jobs form a decision process D and, with the addition of the freeze control, a superprocess S. The controls in a control set, other than control 0, may be identified with the jobs available for selection, because the jobs have only one continuation control. The policies of interest are identified by job sequences, with the convention that a job equivalent to a 0 standard bandit process is selected only if no further positive rewards are available.

Under this setup the rewards that would be gained under policy 123 or 213 with initial state x are

$$R_{123}(D,x) = 0 + \frac{1}{2}\left[1\cdot\left(\frac{1}{2}\right)^3 + M\left(\frac{1}{2}\right)^4\right] = \frac{2+M}{32} \quad \text{and}$$
$$R_{213}(D,x) = 1\cdot\left(\frac{1}{2}\right)^2 + 0 + \frac{1}{2}\cdot M\cdot\left(\frac{1}{2}\right)^4 = \frac{8+M}{32}.$$

The reward from job 1 is always zero, and the rewards that occur after the completion of job 1 are gained only with probability of $\frac{1}{2}$. Similarly the total discounts obey equations

$$W_{123}(D,x) = 1 + \frac{1}{2}\left[\frac{1}{2} + \left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^3\right] = \frac{23}{16} \quad \text{and}$$
$$W_{213}(D,x) = 1 + \frac{1}{2} + \left(\frac{1}{2}\right)^2 + \frac{1}{2}\cdot\left(\frac{1}{2}\right)^3 = \frac{29}{16}.$$

The candidates for optimal policies are then

$$\nu(S,x,1) = \nu_{123}(D,x) = \frac{R_{123}(D,x)}{W_{123}(D,x)} = \frac{2+M}{46} \quad \text{and} \tag{5.10}$$

$$\nu(S,x,2) = \nu_{213}(D,x) = \frac{R_{213}(D,x)}{W_{213}(D,x)} = \frac{8+M}{58}. \tag{5.11}$$

It would not necessarily be optimal to continue the process (if job 1 terminates) until the completion of job 3 unless the condition $M > 1\frac{2}{3}$ wasn't stated. Otherwise the optimal stopping time when starting with job 2 would be $t = 2$, since the index of allocating job 2 only and stopping thereafter is $\frac{1}{6}$. With this assumption depending on the value of M, it is always optimal to continue the process until job 3 is completed.

Clearly, the overall optimal policy is 213, because the expected reward from job 3 is the same for both policies 123 and 213, but the only other nonzero reward from job 2 occurs in policy 213 earlier than in policy 123. The indices (5.10) and (5.11) give the optimal policy only if M is smaller than 21. If $M > 21$ then $\nu(S,x,1) > \nu(S,x,2)$ which conflicts with the optimal policy. Therefore S is not a superprocess that could be included in $\mathscr{S}_{\frac{1}{2}}$.

Gittins suggests that under the following condition the optimal policies of $\mathscr{S}_\beta$ are compatible with index policies defined by (5.9). The condition is also necessary for the index theorem for superprocesses.

Condition C. Consider a SFAS $\{S,\Lambda\}$, where Λ is a standard bandit process with parameter λ_0. Suppose that when S is in state x it is optimal to select S and apply control a. If for all $x \in \Theta_D, \lambda \in \mathbb{R}$, and SFASs $\{S,\Pi\}$ (Π a standard bandit process with parameter λ) for which it is optimal to select S in state x, this implies that it is optimal to apply a to S in state x. In other words, if it is optimal to choose the process S, the optimal control is always the same, independent of the other processes. If S has such a property, then it will be said to satisfy Condition C.

Remark 3. Condition C does not hold for Example 3. To show that consider a SFAS $\mathscr{F}$ of the superprocess S of the example and a standard bandit process Λ with parameter λ and discrete decision times. Further suppose that and $M = 32$ and the discount parameter is still $\beta = \frac{1}{2}$. Thus the feasible policies for $\mathscr{F}$ are following.

g_1: Allocate the standard bandit process only. Yielded reward is

$$R_{g_1}(\mathscr{F}) = \frac{\lambda}{\gamma}. \quad (\text{where } \gamma = \tfrac{1}{2})$$

g_2: Allocate the superprocess with policy 213 and then continue the standard bandit process. The reward obtained from this policy is

$$R_{g_2}(\mathscr{F}) = R_{213}(S) + \lambda\left[\frac{1}{\gamma} - W_{213}(S)\right] = R_{213}(S) - \lambda W_{213}(S) + \frac{\lambda}{\gamma}.$$

g_3: Otherwise the same setting as in the previous alternative, but now apply policy 123. The reward obtains form

$$R_{g_3}(\mathscr{F}) = R_{123}(S) + \lambda\left[\frac{1}{\gamma} - W_{123}(S)\right] = R_{123}(S) - \lambda W_{123}(S) + \frac{\lambda}{\gamma}.$$

By choosing not too great λ the policies g_2 and g_3 are more preferable than g_1. This is avoided by requiring $\lambda < \frac{17}{23}$. The rewards of the other two policies obey following expanded form

$$R_{g_2} = \frac{5}{4} - \left(\frac{29}{16} - \frac{1}{\gamma}\right)\lambda \quad \text{and}$$
$$R_{g_3} = \frac{17}{16} - \left(\frac{23}{16} - \frac{1}{\gamma}\right)\lambda.$$

If $\lambda < \frac{1}{2}$ it holds $R_{g_2} > R_{g_3}$, but in contrary if $\frac{17}{23} > \lambda > \frac{1}{2}$ the opposite statement holds. Thus the optimal control of the superprocess indeed does depend on the parameter of the standard bandit process, which violates the Condition C, as expected.

However, if the condition holds, it is possible to generalise the index theorem for superprocesses.

Theorem 16. *The classes of index policies, forwards induction policies, and optimal policies are identical for a simple family $\mathscr{F}$ of alternative superprocesses S_i $(i = 1, 2, \ldots, n)$ satisfying Condition C.*

To show the equality of index policies and forwards induction policies is possible in a similar way as in the proof of Theorem 11. Intuitively the equality is clear, because the allocated bandit will not be changed until its index drops below its initial value, while other indices remain unchanged. This applies also to later index theorems in this thesis. The proof of the Theorem is provided in [6, Theorem 3.15].

After having discussed the superprocesses Gittins introduces the stopping option of a bandit superprocess [6, §3.9]. Gittins proves that the superprocesses that have an *improving stopping option* automatically fulfil Condition C, but because certain precedence constraints are also sufficient to satisfy the condition, it is not necessary to discuss the stopping option at all. Therefore the next chapter is about precedence constraints.

5.3.2 Precedence constraints

This section is motivated by the proof of the index theorem for FABPs that have precedence constraints. Consider a FABP $\mathscr{F}$ of superprocesses B_j $(j = 1, 2, \ldots, n)$ just respect to precedence constraints, that is, constraints that deny access to some project before one or more projects are first continued and have reached some certain states. In practice, the precedence constraints are realised by not allowing policies that violate the precedence constraints to be applied to the family $\mathscr{F}$.

Assign a valid policy g to the the FABP $\mathscr{F}$ to obtain a bandit process $\mathscr{F}_g$. Further take a stopping time τ for the bandit process, denote the stopping subset of the bandit's state space at time τ with Θ_0 and denote the process times of the bandits B_i in $\mathscr{F}_g$ with τ_i $(i = 1, 2, \ldots, n; \tau_1 + \tau_2 + \ldots + \tau_n = \tau)$. Thus the process time τ_i is the time at which bandit B_i reaches the stopping subset Θ_0 when policy g is used, and τ_i is thus considered $g - measurable$. g-measurable it is in the sense that whether or not $\tau_i \geq t$ depends only on the history of $\mathscr{F}_g$ before the process time of B_i exceeds t.

Note that although τ_is are considered g-measurable, the values are well defined whether or not policy g is used. It is simply the process time of B_i up to the first time when $\mathscr{F}$ would have reached Θ_0 under g. Some other policy, say h, defines other process times that are considered g-measurable if and only if they equal the process times τ_i defined by g. The truncated FABP defined by truncating $\mathscr{F}_g$ at time τ is said to be *h-measurable* if τ_i is *h-measurable* for all i.

Then consider a modified forwards induction (FI*) policy κ for $\mathscr{F}$ and denote the time at the end of the first stage of $\mathscr{F}_\kappa$ with stopping time σ (> 0). Further denote at time σ truncated FABP with $\mathscr{K}$ and the corresponding residual FABP with $\mathscr{G}$. The next Theorem proves the index Theorem for superprocesses with precedence constraints, and the proof of the next Theorem is similar to the proof of Theorem 11, where $\mathscr{K}$ and $\mathscr{G}$ take the places of bandit process B_k^* and the residual SFABP $\mathscr{G}$ respectively. It is good to note that σ and realisations of $\mathscr{K}$ and G are well-defined whether or not policy κ is followed up to time σ. Any policy g applied to $\mathscr{F}$ and to the realisation of $\mathscr{K}$ fix the residual FABP $\mathscr{G}$, and define a policy e for $\mathscr{G}$ and a promotion rule for the bandit process $\mathscr{G}_e$ with respect to σ.

This discussion of different policies and measurabilities is now exercised by allowing one to change the used policy of a bandit process without changing the process times τ_j to conform the new policy. This is denoted with an extended notation $R_{hg\tau}(\mathscr{F})$, which is the total over $j = 1, 2, \ldots, n$ of the expected rewards from the occasions before the process time τ_j at which control 1 is applied to bandit process B_j, where *τ and τ_j are defined with respect to the policy g* as in the previous paragraph, but *h is the policy followed*. In symbols

$$R_{hg\tau}(\mathscr{F}) = \mathbf{E}\sum_{i=0}^{\infty} \beta^{t_i} I_{hg\tau}(t_i) r_h(t_i),$$

where t_i ($i = 0, 1, 2, \ldots$) is the ith decision time under policy h, and $I_{hg\tau}(t_i) = 1$ or 0 depending on whether control 1 is applied at time t_i to a bandit process B_j whose process time is less than, or greater than or equal to τ_j. Also

$$W_{hg\tau}(\mathscr{F}) = \mathbf{E}\sum_{i=0}^{\infty} I_{hg\tau}(t_i) \int_{t_i}^{t_{i+1}} \beta^t \mathrm{d}t, \quad \text{and} \quad \nu_{hg\tau}(\mathscr{F}) = \frac{R_{hg\tau}(\mathscr{F})}{W_{hg\tau}(\mathscr{F})}.$$

By the definition it is clear that $\nu_{gg\tau}(\mathscr{F}) = \nu_{g\tau}(\mathscr{F})$, so it follows

$$\sup_h \nu_{h\kappa\sigma}(\mathscr{F}) \geq \nu_{\kappa\sigma}(\mathscr{F}) = \nu(\mathscr{F}). \tag{5.12}$$

Demanding that (5.12) holds proves to be an useful condition for a FABP.

Condition D. The FABP $\mathscr{F}$ is said to satisfy Condition D if for any initial state and FI* policy κ,

$$\sup_h \nu_{h\kappa\sigma}(\mathscr{F}) = \nu(\mathscr{F}).$$

The following generalisation of the index theorem shows that the previous condition is sufficient for Condition C to hold, but before the Theorem it is shown that the previously mentioned Example 3 violates, as expected, also Condition D.

Remark 4. The above construction of an index fixes the realisation of the bandit process, and thus it can be assumed that job 1 is completed.

If $M > 21$ the usual index of the bandit process D is obtained with policy 123

$$\nu(D) = \nu_{123}(D) = \frac{2+M}{46},$$

but the equivalent constant reward rate from policy 213, if the job 1 is completed, equals

$$\nu_{h\kappa\sigma}(D) = \nu_{\{213\}\{123\}4}(D) = \frac{4+M}{16} \Big/ \frac{15}{8} = \frac{4+M}{30}.$$

Evidently the second expression is always greater than the first one, thus the supremum of Condition D is strictly greater than the index $\nu(D)$. Thus Condition D does not hold for Example 3.

Theorem 17 (The Index Theorem for a FABP with Precedence Constraints)**.** *The classes of index policies, forwards induction policies, and optimal policies are identical for a family $\mathscr{F}$ of alternative superprocesses with no arrivals (regarded as a SFAS), and which satisfies Condition D.*

Proof. The equivalence of the first two classes follows from Lemma 12 and the rest of the proof is similar to the proof of the Theorem 11. Like already mentioned above, in the proof the truncated family of alternative bandit processes $\mathscr{K}$ takes the place of the truncated bandit process B_k^* and the residual bandit process $\mathscr{G}$ is replaced with a residual family of alternative bandit processes which is not necessarily simple, but denoted with the same symbol $\mathscr{G}$.

Like in the proof of Theorem 11 take an arbitrary policy g for $\mathscr{F}$ that with the realisation of $\mathscr{K}$ fixes the random FABP $\mathscr{G}$, and defines a policy h for $\mathscr{G}$ and a promotion rule p for the bandit process $\mathscr{G}_h$ with respect to stopping time σ. This notation allows again one to divide the total expected reward generated by g on $\mathscr{F}$ in two parts

$$\begin{aligned} R_g(\mathscr{F}) &= \mathbf{E}\sum_{i=0}^{\infty} \beta^{t_i} I_{g\kappa\sigma}(t_i) r_g(t_i) + \mathbf{E}\sum_{i=0}^{\infty} \beta^{t_i}(1 - I_{g\kappa\sigma}(t_i)) r_g(t_i) \\ &= R_{g\kappa\sigma}(\mathscr{F}) + \mathbf{E} R_{hp}(\mathscr{G}). \end{aligned}$$

In addition to policies like $g\kappa\sigma$, which apply a certain policy at decision times that are derived from another policy, consider the policy $\kappa\sigma h$ for $\mathscr{F}$ (note that the stopping time is in the middle), which coincides with the FI* policy κ up to time σ, and then applies policy h to the residual FABP $\mathscr{G}$. This policy is feasible, i.e. it does not violate precedence constraints, because

- κ is feasible by definition, it does not violate the precedence constraints within $\mathscr{K}$ nor between $\mathscr{K}$ and $\mathscr{G}$.
- under the policy $\kappa\sigma h$ the process times of each bandit process in $\mathscr{F}$ when control 1 is first applied to any bandit process in $\mathscr{G}$ are no less, for any realisation of $\mathscr{K}$, than they are under policy g.
- g is feasible and therefore does not violate the precedence constraints within $\mathscr{G}$.

Thus $\kappa\sigma h$ is feasible. The target of this setting is to prove that

$$R_{\kappa\sigma h}(\mathscr{F}) \geq R_g(\mathscr{F}), \tag{5.13}$$

i.e. that it is profitable to modify a policy towards an FI* policy. In this purpose it is noted that it is possible to split $R_{\kappa\sigma h}(\mathscr{F})$ in according to processes $\mathscr{K}$ and $\mathscr{G}$

$$R_{\kappa\sigma h}(\mathscr{F}) = R_{\kappa\sigma}(\mathscr{F}) + \mathbf{E}R_{h0}(\mathscr{G}),$$

where 0 denotes the null promotion rule with respect to time σ. From Condition D it follows that

$$R_{g\kappa\sigma}(\mathscr{F}) = W_{g\kappa\sigma}(\mathscr{F})\nu_{g\kappa\sigma}(\mathscr{F}) \leq W_{g\kappa\sigma}\nu(\mathscr{F}),$$

and by definition

$$R_{\kappa\sigma}(\mathscr{F}) = W_{\kappa\sigma}(\mathscr{F})\nu_{\kappa\sigma}(\mathscr{F}) = W_{\kappa\sigma}(\mathscr{F})\nu(\mathscr{F}).$$

The remaining part of the proof follows faithfully the proof of Theorem 11. Statement (5.13) is proven in a similar way as statement (4.8), in the proof $R_{\kappa\sigma}(\mathscr{F})$ and $R_{g\kappa\sigma}(\mathscr{F})$ replaces $R_f(B_k)$ and $*g$ respectively and instead of using part (i) of Lemma 12 one should refer to the discussion of Theorem 15 and [6, Lemma 3.12]. With this argumentation the inequality $\nu(\mathscr{G}) \leq \nu(\mathscr{F})$ becomes strict and the proof is concluded as the proof of Theorem 11 was concluded in the paragraphs following the proof of statement (4.8). □

5.3.3 Arborescent precedence constraints

Example 3 showed that the Gittins index does not always produce optimal policies. Gittins claims that the essential problem in the setting is that there are initially two ways, so to speak, of making progress towards the large reward of project 3 [6, §3.11]. This section shows, that if such precedence constraints are disallowed, the index produces optimal policies.

The precedence constraints of the Example were illustrated in Figure 5.1, with two arcs leading from projects 1 and 2 to the project 3. The *in-degree* of that *directed graph* was two, because there were two arcs that led to the project 3. If the precedence constraints can be expressed with a directed graph, whose in-degree is at most one and which has no circuits, the graph is called arborescent or an *out-tree*. Figure 5.2 illustrates some example directed graphs.

The precedence constraints forming out-trees allow one to divide the FABP $\mathscr{F}$ to *sub-families* $\mathscr{F}_j (j = 1, 2, \ldots, m)(1 \leq m \leq n)$, with the property that there are no precedence constraints between bandit processes belonging to different sub-families. Thus $\mathscr{F}$ may also be regarded as a simple family of alternative bandit superprocesses, the superprocesses being $\mathscr{F}_j$s. Note that the number of jobs available for service at time zero is at most m.

The advantage of arborescent precedence constraints is that a FABP having such constraints fulfils automatically Condition D and therefore satisfies the conditions of Theorem 17 [6, §3.11].

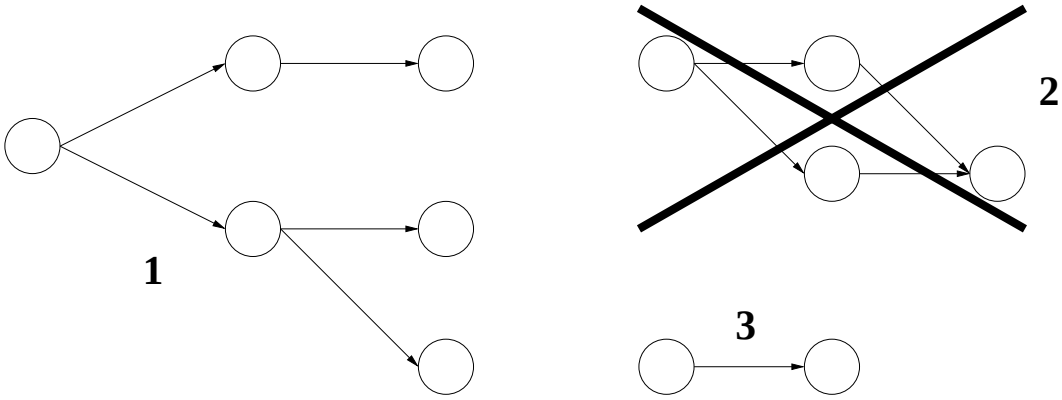


Figure 5.2: Precedence constraints forming out-trees. 1 and 3 represent valid trees $\mathscr{F}_1$ and $\mathscr{F}_3$, while 2 has in-degree of two, and thus is an invalid out-tree.

Suppose that a FABP $\mathscr{F}$ is divided into m *minimal* sub-families, each sub-family having one single initial job, which has to be completed before any of the other jobs in the sub-family may be started. The minimal sub-families of $\mathscr{F}$ are denoted with $\mathscr{F}_j$ $(j = 1, 2, \ldots, m)$ and their corresponding initial jobs with J_j. After completion of an initial job J_j, the sub-family is divided into further minimal sub-families according to the number of jobs that become available after completion of J_j. For example, after completing the initial job of the first sub-family 1 in Figure 5.2, the sub-family is further divided into two sub-families. Note that the number of jobs available after the completion of the initial job of a sub-family can be arbitrary, even zero. If all sub-families include only the initial jobs, there are not any precedence constraints in the FABP $\mathscr{F}$ and the family is actually a SFABP.

It is necessary to define an index for an initial job J_j.

Definition 10 (Sub-family index)**.** *If the state of the initial job J_j is x_j, the sub-family index is defined as*

$$\nu^\dagger(J_j, x_j) = \nu(\mathscr{F}_j, x_j),$$

so that the index for an initial job equals the best index of the sub-family $\mathscr{F}_j$.

Remember that the index of the family is defined as the best equivalent constant reward rate of the family, therefore the index $\nu^\dagger(J_j, x_j)$ is the reward rate of J_j itself and possibly further processes that are preceded by J_j, thus

$$\nu^\dagger(J_j, x_j) \geq \nu(J_j, x_j).$$

A policy for $\mathscr{F}$ is described as a *minimal sub-family index policy* if it at any decision time allocates a non-completed project J_j that conforms the index defined by $\nu^\dagger(J_j, x_j)$. It is possible to generalise the index theorem for this index policy.

Theorem 18. *The classes of minimal sub-family index policies, forwards induction policies, and optimal policies are identical for a family $\mathscr{F}$ of alternative jobs with no arrivals and arborescent precedence constraints.*

The proof of the Theorem succeeds conveniently with help of the index theorem (Theorem 11) and is originally based on a journal article by Glazebrook [7]. Gittins provides also an alternative proof for this Theorem [6, Theorem 3.22].

Before the main part of the proof, it is shown that the minimal sub-family index policy may actually be constructed by reducing the FABP by following instructions.

(i) From the FABP find every sub-family, whose only precedence constraint is the initial job. Such subfamilies always terminate the branches of an out-tree and are termed *terminal* sub-families.

(ii) Construct bandit processes by applying index policy to each of the terminal sub-families and replace every terminal sub-family by the corresponding bandit process. This kind of bandit process completes just when every constituent bandit process has completed.

(iii) If the simplified FABP still has precedence constraints, the stages (i) and (ii) are repeated until $\mathscr{F}$ is reduced to SFABP.

(iv) Form a bandit process B by associating this SFABP with an index policy.

The reward process generated by B is the same as that generated by $\mathscr{F}$ under a minimal sub-family index policy. This minimal sub-family index policy is given by the nested index policies defined by step (ii).

The proof of the Theorem succeeds by showing inductively that this construction yields an optimal policy for the FABP $\mathscr{F}$.

Proof. The first task is to show the optimality of the above defined nested index policies. Like already stated, the proof succeeds by induction on the number of nodes d. The nodes are jobs that precede any other job, i.e. are all jobs excluding jobs that are not *leaves*.

Suppose first that $d = 1$. This equals the situation that the FABP is only a single terminal sub-family. After completion of the initial job the FABP is reduced to a simple FABP and Theorem 11 applies directly, thus it is optimal to schedule the job preceded by the initial job according to the index. If there are two jobs with equal index, the job with the lowest subscript is chosen. This is exactly what is done by the step (ii) in the construction above, and therefore the nested index policies are optimal for $d = 1$.

Secondly it is supposed that the nested index policies are optimal for $d \leq k$ and case $d = k + 1$ is observed. As soon as any job which precedes another job is completed, the system reduces to the case $d = k$, thus the nested index policies hold from that time onwards. Thus the nested index policies are also optimal when $d = k + 1$. This completes the induction.

The optimality of the nested index policies is now proven, but further discussion is needed in order to show that the nested index policies indeed are index policies for $\mathscr{F}$ and that they are also forwards induction policies. The last equality follows again from the discussion of Theorem 15 and the fact that nested index policies are index policies also follows from the previously mentioned Theorem and from the fact that it is an index policy. This discussion concludes the Theorem. □

With the above result it is clear that the sub-family index policies are optimal for FABP:s with arborescent precedence constraints, but the question about allowing

arrivals is still left open. The next section concentrates on removing this limitation by considering them as precedence constraints in an out-tree.

5.3.4 Arrivals

The main idea of accepting arrivals in a FAS is to place the arriving jobs to the out-trees of completing jobs. In this setting any existing job may be taken as a precedence constraint for the arriving jobs.

The arriving jobs are handled by the following procedure. Suppose that the arriving jobs arrive in the system only when some certain job, in this case job B, reaches some certain subset of its state space, say Θ_C. The subset Θ_C must be countable. The arriving jobs form a set or batch, so to say, that is denoted with $\mathscr{T}(B,x)$ and always includes a standard job process with parameter $-M$ ($M \gg 1$). If a job process B has the properties described, it is termed to generate *random out-tree of successors* or, equivalently, to have *job process- (or bandit-)linked arrivals.*

This setting may sound quite restrictive, but it actually isn't that limited, because there may be a set assigned to every job B and state x, i.e. every state x can be considered as a completion state. Thus the completion state $x \in \Theta_C$ of the job B is can be regarded as random, so in general the jobs preceded by job B are random. In addition the batch $\mathscr{T}(B,x)$ may also be random, and there is no restriction why job B could not be included in the batch, so in practice it is possible that the job B is continued before and after the arrival of the batch $\mathscr{T}(B,x)$.

Nevertheless every arriving job of the batch $\mathscr{T}(B,x)$ must have similar properties as the original job in the system, and the arriving jobs may have further their own completion states and corresponding successor jobs. Further all the precedence constraints must be arborescent, thus precedence constraints between batches are not allowed.

The arrival process is illustrated by the following example.

Example 4. Suppose that there is a system of alternative superprocesses that have precedence constraints and arrivals. Initially at time t_0 there are only two jobs B_1 and B_2, which belong to different classes. Suppose that the job belonging to the class pictured with light grey in Figures 5.3 and 5.4 has higher index, so that is processed first. But after some time the job B_1 reaches a *completion state* x_1, at which a batch of new light-class jobs arrive, $T(B_1,x_1) = \{B_1, B_3, B_4\}$. The batch includes also job B_1, which is a precedence constraint for the two other jobs. Thus the job is continued further at once. When completed, the scheduler chooses the next job with the highest available index. Suppose that it is job B_3 and then B_2, and assume further that there is no arrivals during job B_3. But when job B_2 reaches state x_2, again a new job arrives in the system, namely B_5, augmented with the remaining part of B_2, which is continued to the end. The scheduler then chooses B_5 and B_4 in this order, which ends the busy period. The realised process is illustrated by the bottom line of Figure 5.3 and Figure 5.4 illustrates the precedence constraints and the realised process by an arborescent graph.

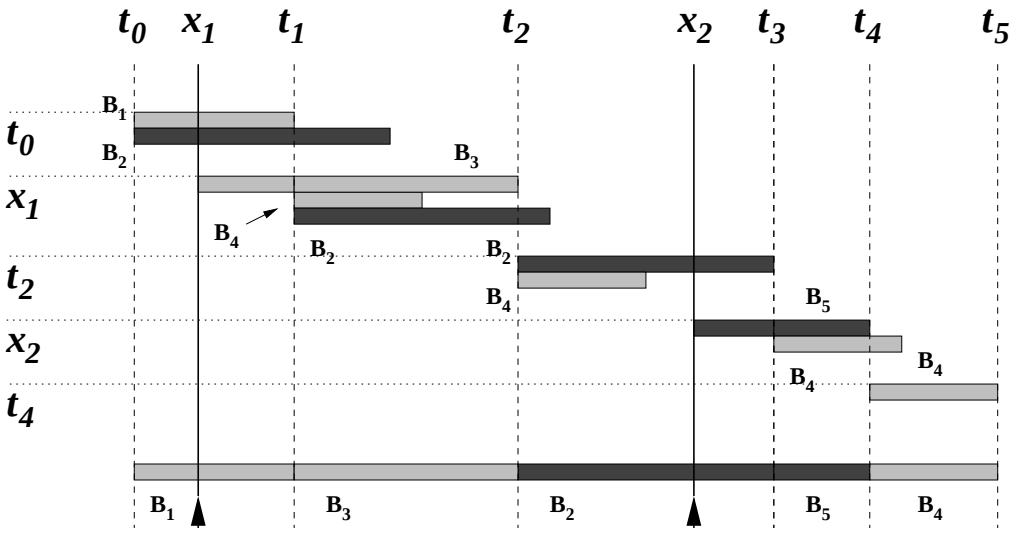


Figure 5.3: The scheduling and arrival processes of Example 4.

It is not clear if Condition D remains satisfied as new batches of jobs arrive in the system, but if the Condition D holds, Gittins states [6, p. 75] that the proof of Theorem 17 holds and may be proven as in case of no arrivals with the exception that there is no fixed limit on the number of the jobs in the system. Further he claims that the result applies also on the index theorem for stoppable jobs mentioned in the end of Section §5.3.1. The discussion leads with [6, Proof 1 of Theorem 3.22] to the final generalisations of the index theorem.

Theorem 19. *The classes of minimal sub-family index policies, forwards induction policies, and optimal policies are identical for a FABP all of whose precedence constraints are defined by an out-tree and the presence of a job-process linked arrival process.*

Unfortunately this theorem covers only arrival processes that are dependent on the completion of existing jobs in the system, thus excluding arrival processes which are independent of the job being processed and which have independent inter-arrival times.

An important exception to this restriction are arrival processes that are memoryless. Further suppose that the compositions of the different batches are identically distributed and independent of previous history. The Poisson arrival process can be regarded as taking place in process time for the various jobs with assumption that the process does not depend on the particular job. Alternatively the arrival process can be regarded as taking place in real time, but it makes no difference, whether the first or second statement is assumed. Because Poisson process is memoryless the arrival process is the same in both cases.

As mentioned earlier in this section, it is possible to include the completed project in the batch that arrives in the system. Therefore one may handle the arrivals *during* the processing of a job by including the remaining portion of the job in the set of successors, as a precedence constraint for the other successors. Note though, that this also alters the minimal sub-family indices of the arriving jobs. Thus the theorem applies also when the arrival process is Poissonian.

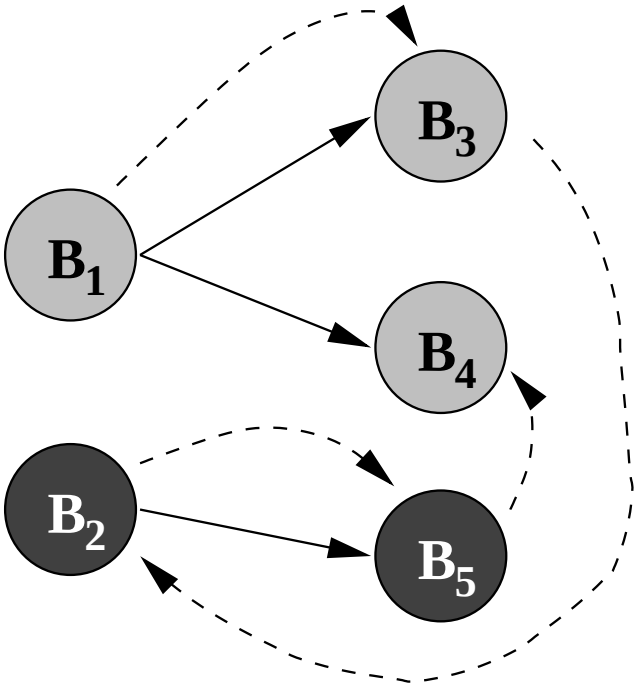


Figure 5.4: The precedence constraints and the scheduling of the jobs discussed by Example 4.

Theorem 20 (The Index Theorem for a FABP with Poisson arrivals)**.** *For a FABP with Poisson arrivals and any arborescent precedence constraints (possibly random), the classes of minimal sub-family index policies, forwards induction policies and optimal policies are identical.*

Note that this Theorem holds even if the process happens in discrete time with the modification that Poisson arrival process is replaced with Bernoulli arrival process.

The following section concludes the theory by showing that the sub-family index policies and the Gittins index policies are equal.

5.3.5 Equality of the sub-family indices and the indices

In the previous section it was shown that sub-family indices are optimal policies for FABPs with Poisson arrivals and arborescent precedence constraints, and that the index policies minimise the EWFT of a M/G/1-queue (Theorem 14) that can be alternatively be regarded as a FABP. Fortunately it can be shown that under the same assumptions as the Theorem 14 was given, it is possible to show that policies defined by sub-family indices equal the normal Gittins index policies. For that is is satisfactory to show that the following theorem holds.

Theorem 21. *The sub-family index $\nu^\dagger(x)$ is an increasing function of the index $\nu(x)$ as x takes different values and discounting is ignored ($\gamma = 0$).*

Proof. For this proof it is necessary to assume the same condition that was assumed for Theorem 14, i.e. the index policy defined by a small enough discount factor must equal the non-discounted policy. In following the index $\nu_i^\dagger(x)$ is defined as the supremum over all policies and stopping times of the average reward rate up to a stopping time obtainable from a sub-family, which is initiated by class i job of age x. The class i refers to the type of the arriving job by means of Section §5.2. Further the cost per unit time c_i associated to class i $(= 1, 2, \ldots, n)$ is treated as reward as in the section mentioned above.

In addition the notation includes vector $T = (T_1, T_2, \ldots, T_n)$, in which T_i is the truncation age of class i jobs besides the initial job, whose truncation age is denoted by y. Thus one may define $R_i(x, y, T)$ as the expected total reward from the minimal sub-family initiated by a class i job of age x up to truncation ages y and T, and $W_i(x, y, T)$ as the corresponding expected total time. With this notation the index $\nu_i^\dagger(x)$ may be defined as follows

$$\nu_i^\dagger(x) = \sup_{\{y>x, T>0\}} \frac{R_i(x, y, T)}{W_i(x, y, T)}. \tag{5.14}$$

If $y = T_i$ the expressions $R_i(x, y, T)$ and $W_i(x, y, T)$ may be simplified by writing

$$R_i(x, T_i, T) = R_i(x, T) \quad \text{and} \quad W_i(x, T_i, T) = W_i(x, T).$$

Now suppose that class i job is processed until it either reaches age y or is finished, whichever occurs first, and denote the age of the initial job at the end of this period by S. Further denote the number of class j $(= 1, 2, \ldots, n)$ jobs arriving during the period by $N_j(S)$. Thus the expected total reward $R_i(x, y, T)$ may be rewritten as follows.

$$R_i(x, y, T) = c_i \mathbf{P}\{S \leq y\} + \sum_{j=1}^{n} R_j(0, T) \mathbf{E}[N_j(S)],$$

where the first term is the expected reward of the initial job and the latter term the expected reward from all other jobs. The expected number of arrivals may be rewritten by means of corresponding arrival rates λ_j. Thus

$$\begin{aligned} R_i(x, y, T) &= c_i \mathbf{P}\{S \leq y\} + \sum_{j=1}^{n} R_j(0, T) \mathbf{E}\{\mathbf{E}[N_j(S)|S]\} \\ &= c_i \mathbf{P}\{S \leq y\} + \sum_{j=1}^{n} R_j(0, T) \mathbf{E}[\lambda_j (S - x)]. \end{aligned}$$

Now the probability $\mathbf{P}\{S \leq y\}$ and expectation of $S - x$ may be expressed alternatively by means of the distribution function $F_i(s)$ so that

$$R_i(x, y, T) \frac{c_i[F_i(y) - F_i(x)]}{1 - F_i(x)} + \sum_{j=1}^{n} \lambda_j R_j(0, T) \frac{\int_x^y [1 - F_i(s)] \mathrm{d}s}{1 - F_i(x)}.$$

Meanwhile, the expected time $W_i(x, y, T)$ obeys expression

$$\begin{aligned} W_i(x, y, T) &= \mathbf{E}[S - x] + \sum_{j=1}^{n} W_j(0, T)\mathbf{E}[N_j(S)] \\ &= \left[1 + \sum_{j=1}^{n} \lambda_j W_j(0, T)\right] \frac{\int_x^y [1 - F_i(s)]\mathrm{d}s}{1 - F_i(x)}. \end{aligned}$$

Substituting $R_i(x, y, T)$ and $W_i(x, y, T)$ in (5.14) gives

$$\begin{aligned} \nu_i^{\dagger}(x) = \sup_{\{y>x, T>0\}} \Bigg\{ &\left[\frac{c_i[F_i(y) - F_i(x)]}{\int_x^y [1 - F_i(s)]\mathrm{d}s} + \sum_{j=1}^{n} \lambda_j R_j(0, T)\right] \\ &\times \left[1 + \sum_{j=1}^{n} \lambda_j W_j(0, T)\right]^{-1} \Bigg\}. \end{aligned}$$

Now the definition of the Gittins index $\nu(x)$ (5.6) is recalled and adapted for class i jobs by

$$\nu_i(x) = \sup_{\{y>x\}} \frac{c_i[F_i(y) - F_i(x)]}{\int_x^y [1 - F_i(s)]\mathrm{d}s.}$$

This expression is part of the latest expression for $\nu_i^{\dagger}(x)$, and by substituting $\nu_i(x)$ to the expression following is obtained

$$\nu_i^{\dagger}(x) = \sup_{\{T>0\}} \frac{\nu_i(x) + \sum_{j=1}^{n} \lambda_j R_j(0, T)}{1 + \sum_{j=1}^{n} \lambda_j W_j(0, T)},$$

which clearly shows that $\nu^{\dagger}$ is an increasing function of ν. □

Thus the Gittins index policies do minimise the EWFT of an M/G/1-queue.

Chapter 6

Experimental comparison of different policies in a MAB system

In this section the policies introduced in Sections §2.5, §2.6 and §3.1 are compared with a simple experiment conducted with help of *Mathematica* computing environment. The general setup of the experiments requires following initial variables to be set properly.

- n The number of projects
- m The number of states that every project has
- β The discount factor inside interval $(0, 1)$.
- r The random rewards for every state and project (matrix size $n \times m$)
- p The transition probability matrices ($m \times m$ matrices for every n projects)
- $\mathbf{P}$ The policy-implementing probability transition matrix (matrix dimensions $m^n \times m^n$)

The Mathematica source files in Appendix B also provide tools for generating the transition probability matrices and rewards randomly. The transition probabilities are drawn from an exponential distribution with the expectancy of one, and then normalised. This approach produces relatively small but also greater probabilities, which proved to be interesting, when the probabilities are combined with uniformly randomised rewards. For example, if a state with high reward is assigned to transition probabilities that lead to a quick transition away from the state, the state was not then necessarily preferable. However, in this experiment, the data was chosen by hand to illustrate the properties of the policies and Gittins indices more predictably.

Experiment data In this experiment the following transition probability matrices were used.

$$p_1 = \frac{1}{10}\begin{bmatrix} 0 & 0 & 0 & 10 \\ 0 & 4 & 5 & 1 \\ 7 & 1 & 2 & 0 \\ 1 & 1 & 5 & 3 \end{bmatrix} \quad p_2 = \frac{1}{20}\begin{bmatrix} 15 & 3 & 0 & 2 \\ 7 & 2 & 7 & 4 \\ 0 & 2 & 5 & 13 \\ 1 & 3 & 3 & 13 \end{bmatrix}$$

$$p_3 = \frac{1}{20}\begin{bmatrix} 1 & 0 & 0 & 19 \\ 4 & 4 & 12 & 0 \\ 2 & 6 & 6 & 6 \\ 0 & 0 & 2 & 18 \end{bmatrix}$$

The rewards were chosen as follows

$$r_1 = [7\ 6\ 3\ 1]\,, \quad r_2 = [3\ 4\ 2\ 5]\,, \quad r_3 = [7\ 3\ 4\ 1]\,,$$

and the applied discount factor was $\beta = 0.9$.

The discussion is divided into two main sections according to whether the method features discounting or not. The first section discusses the discounted case.

6.1 Discounted models

All the discounted iteration models, value iteration (§2.5), policy iteration (§2.6) and the Gittins index policy (§3.1) yielded exactly identical policy, although the Gittins index and even policy iteration were obviously faster than the value iteration. Both iteration models showed correctly that project 2 was the best project, because according to the policy the system locks sooner or later to states, in which only project 2 is allocated. This is natural, because the project 2 has the greatest average reward of the projects and the discount factor is not significant, so there is no point continuing the other projects after they have reached some unpreferable state, which in this experiment is in the both cases state number 4. Note though that it is not always optimal to continue the project with the highest average reward if remarkable discounting is used. To convince the reader about this, a short example is provided.

Example 5. Suppose that there is a family of two projects, of which one is a dummy standard-alike project that yields reward of 1 at every decision time. The second process is supposed to have two states, of which the first state produces immediate reward of 2 and the other 0. Further suppose that the process transfers from the first state to the other with probability of 10% and from the second state to the other with probability of 100%. Further assume that the discount factor is $\beta = \frac{1}{4}$ and the process after every decision time takes always one unit of service time. Thus the average reward of the first project without discount is exactly 1 and of the second project $\frac{10}{11} \cdot 2 \approx 1.82$. Suppose then that the second project is at the state that gives immediate reward of zero. With this initial setting the discounted reward of the first project is $\frac{4}{3}$ and of the second project at most 1 (assuming hypothetically that the state of the project never changes back to the worse state again). Therefore it is optimal to choose the project with smaller average reward.

The stopping criteria (2.16) in the value iteration was the computational accuracy of *Mathematica*, i.e. the iteration was continued until Mathematica considered the condition $v = Lv$ to hold. This is in practice about 15 digits and the iteration took 283 rounds to complete.

The policy iteration was naturally continued until the policy was fixed. This took approximately five iteration rounds, depending on the random starting policy. Although the policy iteration practises matrix inversion, it was considerably faster than the value iteration algorithm, and produced exactly same results.

The experiment results with the absorbing states highlighted are available in Table 6.1. The Gittins index does not provide values for state combinations of the Table, but instead of every single state of every project. The Gittins indices are shown in Table 6.2, the Gittins index policy within Table 6.1 and the ϕ-functions defined by (3.1) are illustrated by Figure 6.1.

By observing the figure one can notice that the states that are associated with the highest index, which is also the index of the project, are not necessarily that attractive when there is no augmentation ($M = 0$). The fact that the stopping option is excluded naturally means that the values are the expected discounted total rewards of the projects with corresponding initial states with infinite time horizon. The non-augmented values are also provided in Table 6.2.

Projects 1 and 3 have some quick transitions to and from some states. For example, if project 1 reaches state 1, it will immediately transfer to state 4, thus making the state 1 less preferable, in spite of a high immediate reward of 7. In contrary, project 2 does not have such properties, and the value functions do not intersect with each other.

Table 6.1: Discounted reward model: Value iteration and Policy iteration give exactly identical results.

			Value iteration		Policy iteration		Gittins index
Project			Discounted	Discounted	Discounted	Discounted	Discounted
1	2	3	value	policy	value	policy	policy
1	1	1	42.80	1	42.80	1	1
1	1	2	40.38	1	40.38	1	1
1	1	3	40.76	1	40.76	1	1
1	1	4	39.62	1	39.62	1	1
1	2	1	44.21	1	44.21	1	1
1	2	2	41.65	1	41.65	1	1
1	2	3	41.99	1	41.99	1	1
1	2	4	41.19	1	41.19	1	1
1	3	1	43.91	1	43.91	1	1
1	3	2	41.22	1	41.22	1	1
1	3	3	41.66	1	41.66	1	1
1	3	4	40.86	1	40.86	1	1
1	4	1	46.29	1	46.29	1	1
1	4	2	43.87	1	43.87	1	1
1	4	3	44.16	1	44.16	1	1
1	4	4	43.52	1	43.52	1	1
2	1	1	44.63	3	44.63	3	3
2	1	2	42.24	1	42.24	1	1
2	1	3	42.52	1	42.52	1	1
2	1	4	41.66	1	41.66	1	1
2	2	1	45.69	3	45.69	3	3
2	2	2	43.19	1	43.19	1	1
2	2	3	43.45	1	43.45	1	1
2	2	4	42.85	1	42.85	1	1
2	3	1	45.47	3	45.47	3	3

continued on next page

continued from previous page

			Value iteration		Policy iteration		Gittins index
Project			Discounted	Discounted	Discounted	Discounted	Discounted
1	2	3	value	policy	value	policy	policy
2	3	2	42.87	1	42.87	1	1
2	3	3	43.20	1	43.20	1	1
2	3	4	42.60	1	42.60	1	1
2	4	1	47.38	3	47.38	3	3
2	4	2	45.00	1	45.00	1	1
2	4	3	45.22	1	45.22	1	1
2	4	4	44.74	1	44.74	1	1
3	1	1	41.95	3	41.95	3	3
3	1	2	39.32	1	39.32	1	1
3	1	3	39.64	1	39.64	1	1
3	1	4	38.67	1	38.67	1	1
3	2	1	43.15	3	43.15	3	3
3	2	2	40.40	1	40.40	1	1
3	2	3	40.69	1	40.69	1	1
3	2	4	40.01	1	40.01	1	1
3	3	1	42.90	3	42.90	3	3
3	3	2	40.03	1	40.03	1	1
3	3	3	40.41	1	40.41	1	1
3	3	4	39.72	1	39.72	1	1
3	4	1	45.09	3	45.09	3	3
3	4	2	42.48	2	42.48	2	2
3	4	3	42.73	2	42.73	2	2
3	4	4	42.18	2	42.18	2	2
4	1	1	39.78	3	39.78	3	3
4	1	2	37.09	3	37.09	3	3
4	1	3	37.51	3	37.51	3	3
4	**1**	**4**	36.25	**2**	36.25	**2**	**2**
4	2	1	41.34	3	41.34	3	3
4	2	2	38.50	2	38.50	2	2
4	2	3	38.88	2	38.88	2	2
4	**2**	**4**	37.99	**2**	37.99	**2**	**2**
4	3	1	41.01	3	41.01	3	3
4	3	2	38.02	3	38.02	3	3
4	3	3	38.51	3	38.51	3	3
4	**3**	**4**	37.62	**2**	37.62	**2**	**2**
4	4	1	43.66	3	43.66	3	3
4	4	2	40.96	2	40.96	2	2
4	4	3	41.29	2	41.29	2	2
4	**4**	**4**	40.58	**2**	40.58	**2**	**2**

Table 6.2: Discounted Gittins indices. Column A: Gittins indices (augmented problem), Column 0: Indices with $M = 0$, Column R: Immediate reward

State ▷	1			2			3			4		
Project ▽	A	0	R	A	0	R	A	0	R	A	0	R
1	**7.00**	3.69	7	**6.00**	3.96	6	**4.66**	3.63	3	**3.32**	3.32	1
2	**3.62**	3.62	3	**4.30**	3.80	4	**3.84**	3.76	2	**5.00**	4.06	5
3	**7.00**	2.17	7	**3.89**	2.34	3	**4.26**	2.27	4	**1.60**	1.60	1

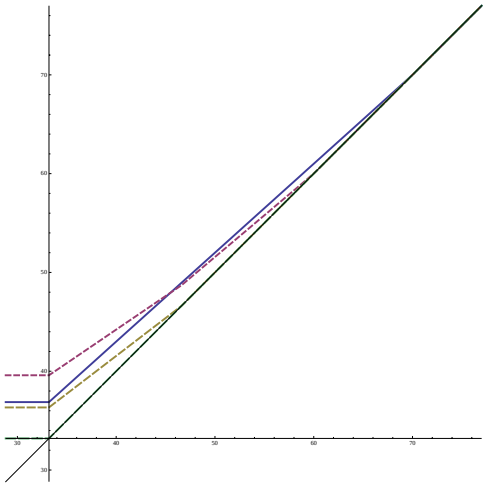
6.2 Average reward models

The result of the same methods, when observing the average reward, was not as consistent as it was in the previous case, although the primary criteria of maximising the average reward was fulfilled by all methods. All the methods produced policies that eventually allocated only project 2, which has the best average reward of the three projects, namely 3.86. Instead the relative rewards and recurrent states varied from method to another and no single evident reason for this was found. The reasons could include e.g. computational inaccuracy or bugs in the program code. The occurred problems are discussed in detail in following paragraphs.

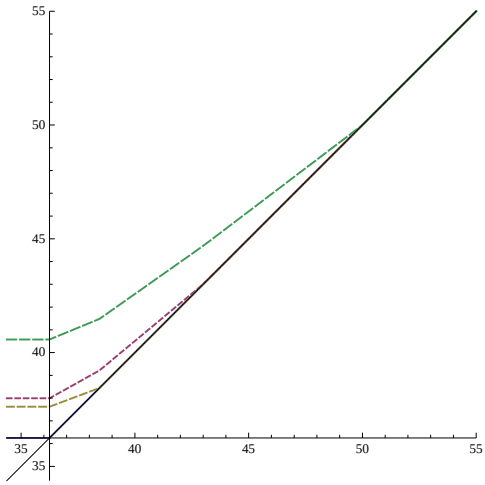
The policy iteration algorithm showed up to be the most problematic case. The algorithm produced almost always different results, depending on the initial randomised policy. Evidently the results produced by the algorithm were not always optimised in means of the relative rewards, because occasionally the algorithm locked into a policy that left projects 1 and 3 untouched after they had reached state 1 that was the state that produces the best immediate reward of seven. Obviously this is not an optimal policy by means of relative rewards.

The relative reward produced by the value iteration algorithm proved to be always (i.e. no counterexamples came upon) better than the rewards produced by the policy iteration algorithm. Unexpectedly, the policy obtained by using the value iteration algorithm almost never allocated project number 3, but project 2 was allocated always when project 1 had reached state 4, no matter what the state of project 3 was.

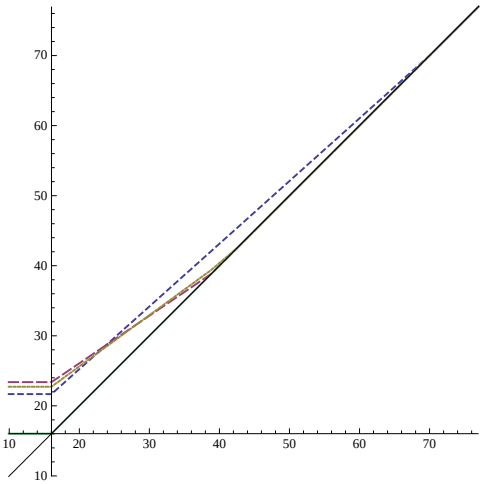
The Gittins index policy in this experiment was produced by a setting that almost ignored the discounting, but not all of it, by using discount factor $\beta = 0.999$. The results were generally better than the result produced by the policy iteration algorithm, but not better than by the value iteration algorithm. The Gittins indices obtained are available in Table 6.4 and the other results results with the recurrent states highlighted are available in Table 6.3.



(a) Project 1



(b) Project 2



(c) Project 3

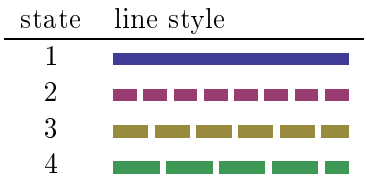


Figure 6.1: Illustration of ϕ-functions

Table 6.3: Average reward model

			Value iteration		Policy iteration		Gittins index
Project			Relative	Av. reward	Relative	Av. reward	$\beta = 0.999$
1	2	3	value	policy	value	policy	policy
1	1	1	3.51	1	3.51	3	1
1	1	2	1.09	1	0.59	1	1
1	1	3	1.38	1	1.04	3	1
1	1	4	0.20	1	0.20	1	1
1	2	1	5.74	1	5.74	3	1
1	2	2	3.33	1	2.82	1	1
1	2	3	3.61	1	3.28	3	1
1	2	4	2.44	1	2.44	1	1
1	3	1	5.86	1	5.86	3	1
1	3	2	3.44	1	2.94	1	1
1	3	3	3.73	1	3.39	3	1

continued on next page

continued from previous page

			Value iteration		Policy iteration		Gittins index
Project			Relative	Av. reward	Relative	Av. reward	$\beta = 0.999$
1	2	3	value	policy	value	policy	policy
1	3	4	2.55	1	2.55	1	1
1	4	1	8.74	1	8.74	3	1
1	4	2	6.32	1	5.82	1	1
1	4	3	6.60	1	6.27	3	1
1	4	4	5.43	1	5.43	1	1
2	1	1	5.91	1	5.91	3	3
2	1	2	3.49	1	2.99	1	1
2	1	3	3.78	1	3.45	3	1
2	1	4	2.61	1	2.61	1	1
2	2	1	8.15	1	8.15	3	3
2	2	2	5.73	1	5.22	1	1
2	2	3	6.01	1	5.68	3	1
2	2	4	4.84	1	4.84	1	1
2	3	1	8.26	1	8.26	3	3
2	3	2	5.85	1	5.34	1	1
2	3	3	6.13	1	5.8	3	1
2	3	4	4.96	1	4.96	1	1
2	4	1	11.14	1	11.14	3	3
2	4	2	8.72	1	8.22	1	1
2	4	3	9.01	2	8.67	3	1
2	4	4	7.83	1	7.83	1	1
3	1	1	2.74	3	2.74	3	3
3	1	2	0.32	1	-0.18	1	1
3	1	3	0.60	1	0.27	3	1
3	1	4	-0.57	1	-0.57	2	1
3	2	1	4.97	1	4.97	3	3
3	2	2	2.55	1	2.05	1	1
3	2	3	2.84	1	2.50	3	1
3	2	4	1.66	1	1.66	1	1
3	3	1	5.09	1	5.09	3	3
3	3	2	2.67	1	2.17	1	1
3	3	3	2.95	1	2.62	3	1
3	3	4	1.78	1	1.78	2	1
3	4	1	7.97	1	7.97	3	3
3	4	2	5.55	1	5.04	1	2
3	4	3	5.83	1	5.50	3	2
3	4	4	4.66	1	4.66	1	2
4	1	1	0.37	**2**	0.37	3	3
4	1	2	-2.05	**2**	-2.55	3	3
4	1	3	-1.77	**2**	-2.10	3	3
4	1	4	-2.94	**2**	-2.94	**2**	**2**
4	2	1	2.60	**2**	2.60	3	3
4	2	2	0.18	**2**	-0.32	3	2
4	2	3	0.47	**2**	0.14	3	2

continued on next page

continued from previous page						
			Value iteration		Policy iteration	Gittins index
Project			Relative	Av. reward	Relative	Av. reward	$\beta = 0.999$
1	2	3	value	policy	value	policy	policy
4	2	4	-0.71	**2**	-0.71	**2**	**2**
4	3	1	2.72	**2**	2.72	3	3
4	3	2	0.3	**2**	-0.2	3	2
4	3	3	0.59	**2**	0.25	3	3
4	3	4	-0.59	**2**	-0.59	**2**	**2**
4	4	1	5.60	**2**	5.60	3	3
4	4	2	3.18	**2**	2.67	3	2
4	4	3	3.46	**2**	3.13	3	2
4	4	4	2.29	**2**	2.29	**2**	**2**

Table 6.4: Gittins indices with discount factor $\beta = 0.999$. Column A: Gittins indices (augmented problem), Column 0: Indices with $M = 0$, Column R: Immediate reward

State ▷	1			2			3			4		
Project ▽	A	0	R	A	0	R	A	0	R	A	0	R
1	**7.00**	3.57	7	**6.00**	3.57	6	**4.77**	3.57	3	**3.57**	3.57	1
2	**3.86**	3.86	3	**4.36**	3.86	4	**4.03**	3.86	2	**5.00**	4.86	5
3	**7.00**	1.77	7	**3.95**	1.78	3	**4.29**	1.78	4	**1.77**	1.77	1

Chapter 7

Conclusions and further work

The objective of this thesis was fulfilled to an extent that was quite satisfactory. The author now understands the construction of the Gittins index way better that he could have understood that without writing the thesis, and hopes that this thesis will make the topic more understandable by a reader to whom the topic is previously unknown.

However, in the beginning of the writing work it was assessed that it would be great if this thesis could go a little beyond the work published on this topic, or to be easily found. For example, this could include generalisation of the schedulable queueing processes beyond the M/G/1-queue. It would also have been nice to add an experiment on queueing with the Gittins index. The main reason, why these questions were not answered in this thesis, is the lack of time and the wish to understand the main theory behind the Gittins index well enough.

Further work on this topic naturally should continue the unfinished work. In practise this means further experimenting and scheduling of N-server queueing systems, different arrival processes and different objective functions other that EWFT. The work should also provide guidelines, how the indices could be evaluated in queueing process in practise.

Appendix A

Queueing systems

In this section the most common queueing arrival and service processes are introduced with some queueing disciplines. In the first section the standard notation [9] of queueing systems [1] is introduced and it is followed by description of different arrival and service processes. In the last section some essential queueing disciplines are introduced.

A.1 The standard notation of a queueing system

The standard way to describe a queueing system was introduced by David. G. Kendall in 1951 [9] and has following form

$$A/S/m/c/p,$$

where each letter describes a certain part of the queueing system.

- A Arrival process (e.g. M, Markovian) defines the distribution which the arrivals obey.
- S Service process (e.g. D, deterministic) defines the distribution that describes the duration of service in a server.
- m Number of servers $(1, 2, \ldots, \infty)$ in the system
- c System slot count
- p Size of the customer population

The quantity c stands for all the slots in the system, both servers and queueing slots. If there are m servers and c slots in the system, then there are at most $c - m$ waiting slots in the system. If c or p is undeclared, they are assumed to be infinite.

A.2 Arrival and service processes

The processes of customers arriving and being processed in the system are characterised through their arrival and service rates λ and μ. In addition the character of the arrival and service processes is usually described with a single letter, that stands for a certain type of probabilistic distribution, which in most cases is one of the following three kinds.

M Markovian, memoryless, Poisson distributed
If a process follows Poisson distribution, the occurrences are fully independent of each other, i.e. remaining time to next occurrence is not dependent on the elapsed time.

G general distribution
If the distribution type is general, there are no restrictions on the type of the distribution. The distribution is characterised through statistical measures like mean and variance.

D deterministic distribution
A deterministic distribution is a constant, fully predictable distribution, e.g. an arrival every second.

A.3 M/M/1-queue

One of the the simplest queueing system is a one-server system with Markovian arrival and service processes. Because both processes are Markovian, it turns out that the number of customers $N(t)$ in the system at time t obeys the geometric distribution. Furthermore it is possible to determine many quantities like *average customer time in the system*, *average number of customers in queue* and *probability of n customers in the system* with simple formulae, needed is only the arrival and service rates λ and μ [1, p. 265].

A.4 M/G/1-queue

Another very common queueing system is a variation of the previous one, the $M/G/1$ queue. Again, there is only one server and the arrival process is Markovian, but the type of the service process is not restricted. Therefore the number of customers in the system does no longer follow the geometric distribution in general. However, there exists a couple of pretty results also for $M/G/1$-queue, like *Pollaczek-Khinchin* formula [1]. The common feature of the results is that most of them include dependence upon the variance of the service time [1, p. 265].

A.5 Queueing disciplines

In this section some fundamental queueing disciplines of scheduling policies are listed, as in Adam Wierman's recent article [20].

FCFS FCFS is also known as FIFO, which stands for First Come First Served and First In First Out respectively. Arriving jobs are served in the order they arrive.

PS Processor sharing (PS) discipline shares the processor time equally to all jobs in the queue, giving the same service rate to each job.

FB Foreground-Background preemptively serves those jobs that have received the least amount of service so far.

LCFS Last Come First Served, or Last In First Out (LIFO) is a discipline that serves non-preemptively the job that arrived most recently.

SRPT Shortest Remaining Processing Time preemptively serves the job with the shortest remaining size.

A.6 Little's Theorem

Sometimes the arrival and service processes are well known and the expected service times can be easily measured, but the expected number of customers in the queueing system remains unknown, or the other way round. Fortunately, independent from the arrival and service processes, there exists a simple and important result known as *Little's Theorem* [1, 10]. First some quantities are described.

Suppose that the sample history of the system is observed from time $t = 0$ to the indefinite future and the values of various quantities are recorded as the time goes by, particularly define

$N(t)$	Number of customers in the system at time t
$\alpha(t)$	Number of customers who arrived in the interval $[0, t]$
T_i	Time spent in the system by the i^{th} arriving customer.

The intuitive notion of the "typical" number of customers in the system observed up to time t is

$$N_t = \frac{1}{t}\int_0^t N(\tau)\mathrm{d}\tau,$$

which is further called *time average of $N(\tau)$ up to time t*. Naturally, N_t changes with the time t, but in many systems of interest, N_t tends to a steady-state N as t increases, that is,

$$N = \lim_{t\to\infty} N_t.$$

This quantity is called *steady-state time average* of $N(\tau)$. It is also natural to view

$$\lambda_t = \frac{\alpha(t)}{t}$$

as the *time average arrival rate* over the interval $[0, t]$. The *steady-state arrival rate* is again received as the limit of λ_t

$$\lambda = \lim_{t \to \infty} \lambda_t.$$

Similarly, the *time average of the customer delay up to time* t is defined as

$$T_t = \frac{\sum_{i=0}^{\alpha(t)} T_i}{\alpha(t)},$$

that is, the average time spent in the system per customer up to time t. Again the steady-state version is defined as

$$T = \lim_{t \to \infty} T_t.$$

If the limit values N, λ and T exist, it turns out that the quantities are related by a simple formula.

Little's Theorem. *The quantities N, λ and T are related to each other through equation*

$$N = \lambda T. \tag{A.1}$$

It is possible to determine any quantity when the other are known.

The original proof written by the author of the theorem is available in a journal article [10].

Appendix B

Mathematica source

B.1 initialise.nb

This file was used to initialise the experiment data with help of the file helper.nb.

First cell runs all cells of helper.nb.

```
(* Run every cell in helper.nb *)
Clear["Global`*"]
helper = Notebooks["helper.nb"];
If[Length[helper] == 0,
  Print["Please open the notebook helper.nb"];];
helper = helper[[1]];
SelectionMove[helper, All, Notebook];
SelectionEvaluate[helper];
```

Initialise the experiment data.

```
(* Initialise the experiment data. *)
(* number of states and projects, and the discount factor *)
states = 4;
projects = 3;
beta = 0.9;
n = states^projects;
(* state combinations *)

projectstates = Tuples[Range[states], projects];
(* Transition probabilies *)
p = {{{0.0, 0.0 , 0.0, 1.0}, {0.0, 0.4, 0.5, 0.1},
    {0.7, 0.1, 0.2, 0.0}, {0.1, 0.1, 0.5, 0.3}},
  {{0.75, 0.15, 0.0, 0.1}, {0.35, 0.1, 0.35, 0.2},
    {0.0, 0.1, 0.25, 0.65}, {0.05, 0.15, 0.15, 0.65}},
  {{0.05, 0.0, 0.0, 0.95}, {0.2, 0.2, 0.6, 0.0},
```

```
    {0.1, 0.3, 0.3, 0.3}, {0.0, 0.0, 0.1, 0.9}}};

(* Respective rewards *)
r = {{7, 6, 3, 1}, {3, 4, 2, 5}, {7, 3, 4, 1}};

(* A random policy *)
{randomPTr, randomRewards, randomPolicy} =
  randomPolicyGenerator[p, r, projectstates, n];
```

B.2 helper.nb

This file includes a set of functions used by other files.

Function pTrAfterPolicy determines the transition probability matrix for entire set of state combinations according to a certain policy.

```
pTrAfterPolicy[policy_, projectstates_, p_] :=
  Module[{pTr, n, project},
   n = Length[policy];
   pTr = Table[0., {n}, {n}];
   Do[
    project = policy[[i]];
    Do[
     If[Max[
         Abs[Delete[projectstates[[i]] - projectstates[[j]],
           project]]] == 0,
       pTr[[i, j]] =
         p[[project, projectstates[[i, project]],
          projectstates[[j, project]]]];
       ];,
     {j, n}];,
    {i, n}];
   pTr];
```

Function rewardsAfterPolicy determines the immediate rewards according to the given policy.

```
rewardsAfterPolicy[policy_, projectstates_, r_] :=
  Module[{rewards, n, project},
   n = Length[policy];
   rewards = Table[0., {n}];
   Do[
    project = policy[[i]];
    rewards[[i]] = r[[project, projectstates[[i, project]]]];,
    {i, n}];
   rewards];
```

Function greedyPolicy generates the greedy policy.

```
greedyPolicy[p_, r_, n_, projectstates_] :=
  Module[{gPTr, gRewards, gPolicy},
   gPTr = Table[0., {i, n}, {j, n}];
   gRewards = Table[0., {i, n}];
   gPolicy = Table[0, {i, n}];
   projects = Length[p];
   states = Length[p[[1]]];
   Do[
    rewards = Table[r[[j, projectstates[[i, j]]]], {j, projects}];
    bestproject = Position[rewards, Max[rewards], 1, 1][[1, 1]];
    gPolicy[[i]] = bestproject;
    Do[
     If[Max[
         Abs[Delete[projectstates[[i]] - projectstates[[j]],
           bestproject]]] == 0,
       gPTr[[i, j]] =
         p[[bestproject, projectstates[[i, bestproject]],
          projectstates[[j, bestproject]]]];
       ];,
     {j, n}];
    gRewards[[i]] =
     r[[bestproject, projectstates[[i, bestproject]]]];,
    {i, n}];
   {gPTr, gRewards, gPolicy}];
```

Function e returns a vector, where all the indices listed in the list have equal nonzero values while all other components are zero. The length of the vector is n and the norm is 1.

```
e[list_, n_] := Table[If[MemberQ[list, i], 1.0/Length[list], 0], {i, n}];
```

Function reduceStates searches the transient and absorbing states of given policy and transition probabilities. There may be more than one group of absorbing states.

```
reduceStates[policy_, projectstates_, p_] :=
  Module[{n, pPi, pTr, project, absorbingstates, transientstates},

   (* The number of state combinations is the length of the policy \
    vector *)
   n = Length[policy];

   (* pTrAfterPolicy returns the probability transition matrix \
   according to the given policy *)
   pTr = pTrAfterPolicy[policy, projectstates, p];
```

```
  (* pPi is the probability transition matrix of infinite number of \
  transitions *)
  pPi = Chop[FixedPoint[#.pTr &, pTr, 50000]];

  (* The first guess for the states that are absorbing is to search \
  the possible final states for every initial state. \
  Union removes the duplicates. *)
  absorbingstates =
   Union[Map[
     projectstates[[
       Flatten[Position[e[{#}, n].pPi, x_ /; x != 0]]]] &,
     Range[n]]];

  (* The final state groups, which are unions of other candidates,
  are discarded. Only the smallest groups are accepted *)
  absorbingstates =
   Select[absorbingstates,
    And @@ Table[(Intersection[#, absorbingstates[[i]]] == # ||
         Intersection[#, absorbingstates[[i]]] == {}), {i,
        Length[absorbingstates]}] &];

  (* The group of transient states is supposed to be the complement \
   of absorbing states *)
  transientstates =
   Complement[projectstates, Flatten[absorbingstates, 1]];
  {absorbingstates, transientstates}];
```

Function verify tests if the absorbing and transient state groups really are what expected.

```
verify[absorbingstates_, transientstates_, policy_, projectstates_, p_] :=
  Module[{temp, pTr, verification},

   (* The test must be run for each absorbing state group *)
   verification = Table[False, {i, Length[absorbingstates]}];

   (* The transition probability matrix for one step and infinite \
   number of steps.  *)
   pTr = pTrAfterPolicy[policy, projectstates, p];
   pPi = Table[0., {n}, {n}];
   pPi = FixedPoint[#.pTr &, pTr, 10000];
   Do[
    (* The temp vector represesents the initial state,
    where we start already from absorbing states *)
    temp = e[
       Flatten[
        Map[Position[projectstates, #] &, absorbingstates[[i]]]], n].pTr;

    (* After one transition according to the policy the state of the \
```

```
    project must stay in the absorbing group. *)
    verification[[i]] = (Flatten[Position[temp, x_ /; x != 0]] ==
       Flatten[Map[Position[projectstates, #] &, absorbingstates[[i]]]]);,
    {i, Length[absorbingstates]}];
   (* First boolean value tells if the absorbing states really are \
   absorbing. Second boolean value tells if the redundant states \
   really are redundant according to the transition probability \
   matrix in case of infinite number of transitions. *)
   {And @@ verification,
    projectstates[[Flatten[Position[Map[Norm[#, 1] &, Transpose[pPi]],
        x_ /; x == 0.]]]] == redundantstates}];
```

This function initialises random rewards and transition probabilities.

```
initialize[states_, projects_, {RewMax_, RewMin_}] :=
  Module[{n, m, p, r, projectstates}, n = states^projects;
   projectstates = Tuples[Range[states], projects]; m = projects^n;
   r = Table[
     RandomReal[{RewMin, RewMax}, WorkingPrecision -> 1], {i,
      projects}, {j, states}];
   p = Table[0., {i, projects}, {j, states}, {k, states}];
   p = Table[
     Random[ExponentialDistribution[1]], {i, projects}, {j,
      states}, {k, states}];
   p = Map[#1/Norm[#1, 1]&, p, {2}]; {p, r, n, projectstates}];
```

Function randomPolicy creates a random policy.

```
randomPolicyGenerator[p_, r_, projectstates_, n_] :=
  Module[{pTr, rewards, policy}, pTr = Table[0., {n}, {n}];
   rewards = Table[0., {n}]; policy = Table[0, {n}];
   projects = Length[p];
   Do[project = RandomInteger[{1, projects}]; policy[[i]] = project;
    Do[If[Norm[Delete[projectstates[[i]] - projectstates[[j]],
    project], 1] == 0,
       pTr[[i, j]] =
         p[[project, projectstates[[i, project]],
          projectstates[[j, project]]]];];, {j, n}];
    rewards[[i]] = r[[project, projectstates[[i, project]]]];, {i, n}];
   {pTr, rewards, policy}];
```

Function solveHoward evaluates a policy without discounting.

```
solveHoward[pTr_, rewards_, n_] :=
  Module[{u, w, vars, lhs, rhs, pi, result, value, aValue, h, g},
   (* Clear and initialise unknown variables for Solve. *)
```

```
   h = Table[0, {n}];
   g = Table[0, {n}];
   Do[
    h[[i]] = u[i];
    g[[i]] = w[i];,
    {i, n}];
   (* Variables to be solved (value function h and average reward g) *)
    vars = Flatten[{h, g}];

   (* Left hand side of the equation is union of equations from the \
      Howard equation and additional equations. *)
   (* First equation *)
   lhs = (pTr - IdentityMatrix[n]).g;

   (* Second equation *)
   lhs = Flatten[{lhs, rewards - g + (pTr - IdentityMatrix[n]).h}];
   (* Constraint equation *)

   pi = Chop[FixedPoint[#.pTr &, pTr, 50000]];
   lhs = Flatten[{lhs, pi.h}];

   (* Right hand side is zero. *)
   rhs = Table[0, {Length[lhs]}];

   result = Solve[lhs == rhs, vars];
   value = Flatten[h /. result];
   aValue = Flatten[g /. result];
   {aValue, value}];
```

B.3 gittinscontinuous.nb

Evaluating Gittins indices and value functions.

```
phi = Table[0., {projects}, {states}];
giOrder = Table[0, {projects}, {states}];
(* Following loop solves first the value function and index from the \
project in ascending order *)
Do[
  (* Clear old values *)
  Clear[temp1, temp2, temp3, temp4, temp5, temp6, eqn, res, w];
  temp1 = Table[0., {projects}, {states}];
  temp5 = temp4 = temp3 = temp2 = temp1;
  Do[
   (* temp1 is an unknown vector for Solve *)

   temp1[[i]] = Table[w[j], {j, states}];, {i, projects}];

  (* The dynamic programming equation without max as an expression *)
```

```
    eqn = Table[(r[[i]] + beta*p[[i]].temp1[[i]] - temp1[[i]])[[j]],
      {i, projects}, {j, states}];
  (* Result vector *)
  res = Table[0., {projects}];
  Do[
   Do[
    (* Add the varibale to the equation *)
    eqn[[i, giOrder[[i, j]]]] = phi[[i, giOrder[[i, j]]]] -
       w[giOrder[[i, j]]];,
    {j, currentIndex - 1}];
   (* Solve the equation without max *)
   res[[i]] = Solve[eqn[[i]] == 0, temp1[[i]]];,
   {i, projects}];

  (* temp2 is the results in substitution form *)
  temp2 = Flatten[res, 1];
  (* temp3 is the plain results without substitution *)
  Do[
   temp3[[i]] = temp1[[i]] /. temp2[[i]];,
   {i, projects}];

  (* temp4 is the results operated with Max[.,M] *)
  temp4 = Map[PiecewiseExpand, Map[Max[M, #] &, temp3, {2}], {2}] //
    Simplify;

  (* temp5 pics out the biggest delimiter of the piecewise functions \
    (Gittins indices) *)
  temp5 = Table[
    temp4[[i, j, 1, Length[temp4[[i, j, 1]]], 2,
     Length[temp4[[i, j, 1, Length[temp4[[i, j, 1]]], 2]]]],
    {i, projects}, {j, states}];
  temp6 = Map[Ordering, temp5];
  giOrder = Take[#, currentIndex] & /@ temp6;
  (* Save the solved value function to phi *)
  Do[
   phi[[i, Last[giOrder[[i]]]]] = temp4[[i, Last[giOrder[[i]]]]];,
   {i, projects}];,
  {currentIndex, states}];
giIndices = temp5*(1 - beta);
(* Last lines produce a policy vector that indicates the project with \
the best Gittins index *)
GIPolicy = Table[0, {n}];
Do[stateGiIndices =
   Table[giIndices[[j, projectstates[[i, j]]]], {j, projects}];
  GIPolicy[[i]] =
   Position[stateGiIndices, Max[stateGiIndices], 1, 1][[1, 1]];,
  {i, n}];
(* Further results *)
pTr = pTrAfterPolicy[GIPolicy, projectstates, p];
rewards = rewardsAfterPolicy[GIPolicy, projectstates, r];
```

```
{GIabsorbingstates, GItransientstates} =
  reduceStates[GIPolicy, projectstates, p];
{GIAValue, GIValue} = solveHoward[pTr, rewards, n];
```

The results are drawn.

```
Clear[plots];
Do[plotmax = (giIndices[[i, giOrder[[i, -1]]]] 1.1)/(1 - beta);
 plotmin = giIndices[[i, giOrder[[i, 1]]]]/(1 - beta);
 margin = (plotmax - plotmin) 0.1; phi2 = Append[phi[[i]], M];
 plots[i] =
  Plot[phi2, {M, plotmin - margin, plotmax}, AspectRatio -> 1,
   PlotRange -> {{plotmin - margin, plotmax}, {plotmin - margin,
      plotmax}}, AxesOrigin -> {plotmin, plotmin}, Exclusions -> None,
    PlotStyle -> {{Thickness[0.004]}, {Dashing[{0.01, 0.008}],
      Thickness[0.004]}, {Dashing[{0.015, 0.008}],
      Thickness[0.004]}, {Dashing[{0.02, 0.008}], Thickness[0.004]},
     Black}],
 {i, projects}]
```

B.4 policyiteration.nb

Discounted policy iteration

```
(* Initialisation of runtime variables *)
alternatePolicies = Table[0., {projects}, {n}];
alternateRewards = Table[0., {projects}];
iPTr = randomPTr;
iRewards = randomRewards;
iPolicy = Table[0, {n}];
lastPTr = iPTr*0.;
lastRewards = iRewards*0.0;
i = 1;

(* Iterate until the policy (and corresponding transition \
probabilities) remain unchanged *)
While[iPTr != lastPTr,
  (* Progress indicator *)
  Print["Iteration ", i];
  lastPTr = iPTr;
  lastRewards = iRewards;

  (* Value update *)
  V = Inverse[IdentityMatrix[n] - beta*lastPTr].lastRewards;

  (* Create alternative policies for comparation *)
  Do[
```

```
   Do[
    alternatePolicies[[project]] *= 0.0;
    (* Check only transitions to states that differ only in one \
    project. *)
    Do[
     If[Norm[
         Delete[projectstates[[state]] - projectstates[[j]], project],
          1] == 0,
       alternatePolicies[[project, j]] =
         p[[project, projectstates[[state, project]],
          projectstates[[j, project]]]];
       ];,
     {j, n}];
    alternateRewards[[project]] =
     r[[project, projectstates[[state, project]]]];,
    {project, projects}];
   rewards =
    Flatten[Table[
      alternateRewards[[i]] + beta*alternatePolicies[[i]].V, {i,
       projects}]];

   (* Choose the best available decision for every state. *)
   bestProject = Position[rewards, Max[rewards]][[1, 1]];
   (* Update the policy. *)
   iPolicy[[state]] = bestProject;
   iPTr[[state]] = alternatePolicies[[bestProject]];
   iRewards[[state]] = alternateRewards[[bestProject]];,
   {state, n}];
  i++;
  ];
(* Save the results to variables. *)
DPIPolicy = iPolicy;
DPIValue = V;
DPIPTr = iPTr;
DPIrewards = iRewards;
{DPIabsorbingstates, DPItransientstates} =
  reduceStates[DPIPolicy, projectstates, p];
```

Average reward policy iteration

```
(* Initialisation of runtime variables *)
alternatePolicies = Table[0., {projects}, {n}];
alternateRewards = Table[0., {projects}];
(* Generate random initial policy. *)
{randomPTr, randomRewards,
   randomPolicy} = randomPolicyGenerator[p, r, projectstates, n];
iPTr = randomPTr;
iRewards = randomRewards;
iPolicy = randomPolicy;
```

```
lastPTr = iPTr*0.0;
lastRewards = iRewards*0.0;
i = 1;

(* Iterate until the policy (and corresponding transition \
probabilities) remain unchanged *)
While[iPTr != lastPTr,
  Print["Iteration ", i];
  lastPTr = iPTr;
  lastRewards = iRewards;

  (* Evaluate policy. *)
  {AV, V} = solveHoward[iPTr, iRewards, n];

  Do[
   (* Create alternative policies for comparation. *)
   Do[
    alternatePolicies[[project]] *= 0.0;
    Do[
     (* Accept only transition to states that differ only in one \
        project. *)

     If[Norm[Delete[projectstates[[state]] - projectstates[[j]],
          project], 1] == 0,
       alternatePolicies[[project, j]] =
         p[[project, projectstates[[state, project]],
          projectstates[[j, project]]]];
       ];,
     {j, n}];
    alternateRewards[[project]] =
     r[[project, projectstates[[state, project]]]];,
    {project, projects}];

   (* Evalueate alternative rewards. *)

   rewards =
    Flatten[Table[alternatePolicies[[i]].AV, {i, projects}]];
   bestProject = Position[rewards, Max[rewards]][[1, 1]];

   (* Choose the best available rewards and update the policy. *)
   (*
   Condition (a) *)

   If[rewards[[bestProject]] > rewards[[iPolicy[[state]]]],

    iPolicy[[state]] = bestProject;
    iPTr[[state]] = alternatePolicies[[bestProject]];
    iRewards[[state]] = alternateRewards[[bestProject]];,

    (* Condition (b) *)
```

```
    rewards =
     Flatten[Table[
       alternateRewards[[i]] + alternatePolicies[[i]].V, {i, projects}]];
    bestProject = Position[rewards, Max[rewards]][[1, 1]];
    If[rewards[[bestProject]] > rewards[[iPolicy[[state]]]],
     iPolicy[[state]] = bestProject;
     iPTr[[state]] = alternatePolicies[[bestProject]];
     iRewards[[state]] = alternateRewards[[bestProject]];
     ];
    ];,
   {state, n}];
  i++;
  ];
(* Save the results to variables. *)
APIPTr = iPTr;
APIrewards = iRewards;
APIPolicy = iPolicy;
APIValue = V;
APIAValue = AV;
{APIabsorbingstates, APItransientstates} =
  reduceStates[APIPolicy, projectstates, p];
```

B.5 valueiteration.nb

Discounted value iteration

```
(* Initial policy is the greedy policy. *)
{gPTr, gRewards, gPolicy} =
   greedyPolicy[p, r, n, projectstates];

(* arrays for policies to be compared *)
alternatePolicies = Table[0., {projects}, {n}];
alternateRewards = Table[0., {projects}];

(* arrays to save the iterated policy *)
iPTr = gPTr;
iRewards = iV = gRewards;
oV = 0.*iV;
iPolicy = gPolicy;

(* Iterate until the discounted value does not change any more. *)
i = 1;
While[iV != oV,

  (* Print iteration count always when i is power of 2. *)
  If[IntegerQ[Log[2, i]],
   Print["Iteration ", i]];
```

```
  (* Create alternative policies. *)
  Do[
   Do[
    alternatePolicies[[project]] *= 0.0;
    (* Accept only transition to states that differ only in one \
    project. *)
    Do[
     If[Max[
         Abs[Delete[projectstates[[state]] - projectstates[[j]],
           project]]] == 0,
       alternatePolicies[[project, j]] =
         p[[project, projectstates[[state, project]],
          projectstates[[j, project]]]];];,
     {j, n}];

    (* alternate rewards *)
    alternateRewards[[project]] =
     r[[project, projectstates[[state, project]]]];,
    {project, projects}];

   (* possible new values for the value function *)
   newValues =
    Table[alternateRewards[[j]] + beta*
       alternatePolicies[[j]].iV, {j, projects}];

   (* the best values are chosen *)
   bestproject = Position[newValues, Max[newValues]][[1, 1]];
   (* the policy is updated *)
   iPolicy[[state]] = bestproject;
   iPTr[[state]] = alternatePolicies[[bestproject]];
   iRewards[[state]] = alternateRewards[[bestproject]];,
   {state, n}];
  oV = iV;
  iV = iRewards + beta*iPTr.iV;
  i++;
  ];

(* Save the results to variables. *)
Print["Iteration ", i];
DVIPolicy = iPolicy;
DVIrewards = iRewards;
DVIValue = iV;
DVIPTr = iPTr;
DVIrewards = iRewards;
{DVIabsorbingstates, DVItransientstates} =
  reduceStates[DVIPolicy, projectstates, p];
```

Average reward value iteration

```
(* Initial policy is the greedy policy. *)
{gPTr, gRewards, gPolicy} =
   greedyPolicy[p, r, n, projectstates];

(* arrays for policies to be compared *)
alternatePolicies = Table[0., {projects}, {n}];
alternateRewards = Table[0., {projects}];

(* arrays to save the iterated policy *)
iPTr = gPTr;
AV = iRewards = iV = gRewards;
oAV = oV = 0.*iV;
i = 1;
iPolicy = gPolicy;

(* do until the values do not change *)
While[Max[Abs[Chop[AV - oAV, 10^(-7)]]] != 0.0,

  (* Print iteration count always when i is power of 2. *)
  If[IntegerQ[Log[2, i]],
    Print["Iteration ", i]];
  (* for every state combination *)
  Do[
   (* for every project *)
   Do[
    alternatePolicies[[project]] *= 0.0;
    (* again for every state,
    search in which states the transition is possible. *)
    Do[
     If[Max[
         Abs[Delete[projectstates[[state]] - projectstates[[j]],
           project]]] == 0,
        alternatePolicies[[project, j]] =
          p[[project, projectstates[[state, project]],
           projectstates[[j, project]]]];];,
     {j, n}];

    (* alternate rewards *)
    alternateRewards[[project]] =
      r[[project, projectstates[[state, project]]]];,
    {project, projects}];

   (* possible new values for the value function *)
   rewards =
    Table[alternateRewards[[j]] + alternatePolicies[[j]].iV, {j,
      projects}];

   (* the best values are chosen *)
```

```
   bestproject = Position[rewards, Max[rewards]][[1, 1]];

   (* the policy is updated *)
   iPolicy[[state]] = bestproject;
   iPTr[[state]] = alternatePolicies[[bestproject]];
   iRewards[[state]] = alternateRewards[[bestproject]];,
   {state, n}];
  oAV = AV;
  oV = iV;
  i++;
  iV = iRewards + iPTr.iV;
  AV = iV - oV;
  ];
(* Save the results to variables. *)
Print["Iteration ", i];
AVIPTr = iPTr;
AVIrewards = iRewards;
AVIPolicy = iPolicy;
AVIAValue = AV;
AVIValue = iV - i*AV;
{AVIabsorbingstates, AVItransientstates} =
  reduceStates[AVIPolicy, projectstates, p];
```

Bibliography

[1] Dimitri Bertsekas and Robert Gallager. *Data networks (2nd ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992.

[2] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control 2nd Edition*, volume 2. Athena Scientific, 2001.

[3] David Blackwell. Discounted dynamic programming. *Annals of Mathematical Statistics*, 36(1):226–235, 1965.

[4] Jerzy Filar and Koos Vrieze. *Competitive Markov decision processes*. Springer-Verlag New York, Inc., New York, NY, USA, 1996.

[5] J.C. Gittins. Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society. Series B (Methodological)*, 41(2):148–177, 1979.

[6] J.C. Gittins. *Multi-armed Bandit Allocation Indices*. Wiley-Interscience series in Systems and Optimization. John Wiley & sons, New York, 1989.

[7] K. D. Glazebrook. Stochastic scheduling with order constraints. *International journal of system sciences*, 7(6):657–666, 1976.

[8] Olivier Gottfried. Kostenminimale prioritäten in wartesystemen vom typ m/g/1. *Elektronische Rechenanlagen*, 14(6):262–271, 1972.

[9] David G. Kendall. Some problems in the theory of queues. *Journal of the Royal Statistical Society. Series B (Methodological)*, 13(2):151–185, 1951.

[10] John D. C. Little. A proof for the queuing formula: L= λw. *Operations Research*, 9(3):383–387, may 1961.

[11] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, New York, NY, USA, 1994.

[12] Sheldon M. Ross. *Applied Probability Models with Optimization Applications*. Dover Publications inc. New York, 1992.

[13] Satinder Singh and David Cohn. How to dynamically merge markov decision processes. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.

[14] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press (Bradford Book), 1998.

[15] Henk C. Tijms. *Stochastic Models: An Algorithmic Approach.* John Wiley & Sons, 1994.

[16] J. Virtamo and S. Aalto. Calculation of time-dependent blocking probabilities. In B. Goldstein, A. Koucheryavy, and M. Shneps-Shneppe, editors, *the Proceedings of the ITC Sponsored St. Petersburg Regional International Teletraffic Seminar Teletraffic Theory as a Base for QoS: Monitoring, Evaluation, Decisions*, pages 365–375, St. Petersburg, Jun. 1998.

[17] Richard Weber. On the gittins index for multiarmed bandits. *The Annals of Applied Probability*, 2(4):1024–1033, 1992.

[18] Peter Whittle. Multi-armed bandits and the gittins index. *Journal of the Royal Statistical Society*, 42(2):143–149, 1980.

[19] Peter Whittle. *Optimal Control: Basics and Beyond.* John Wiley & Sons, 1996.

[20] Adam Wierman. Fairness and classifications. *SIGMETRICS Perform. Eval. Rev.*, 34(4):4–12, 2007.