# *Incremental Web Crawling as a Competitive Game of Learning Automata*

by

*Svein Arild Myrer*
*Morten Goodwin Olsen*

Master Thesis in
Information and Communication Technology

Agder University College

Grimstad, May 2005

**Abstract**

There is no doubt that the World Wide Web has lived up to it's hype of being the world's central information highway through the past years. An increasing amount of versatile services keeps finding their way onto the Web as information providers continue to embrace the possibilities that the Web can offer. Especially the possibility of producing dynamic content has been an accelerant factor and is the reason why we now conveniently can participate in online auctions or see the latest development of our favorite stocks in near real-time from our own living rooms.

However, for automated data mining applications that deploy crawlers to continuously capture the information provided by this new breed of services, the highly dynamic nature of the content is not convenient at all. As a matter of fact, a complete new set of challenges emerges where traditional crawling strategies are shown to be sub-optimal. Accordingly a new class of methods for crawling operations are clearly needed. Nonetheless, the problem area has so far been given limited attention in literature.

In this thesis we address the new problem area of monitoring highly dynamic data sources of different importance. We use the concept of an incremental web crawler as a basis for our novel approach where we consider the incremental crawling task as a continuous learning problem where scheduling of monitoring tasks is combined with parameter estimation in an on-line manner. By mapping the problem to two variants of the so called knapsack problem we propose two solutions based on a machine learning technique known as learning automata.

We show empirically that our proposed solutions continuously improve their performance through a learning process and that they are capable of operating in non-stationary environments. We also show their performance in comparison to alternative algorithms where, most notably, our schemes are shown to outdo the traditional uniform crawling scheme by factors up to 550% in certain situations.

# Preface

This thesis is submitted in partial fulfillment of the requirements for the degree Sivilingeniør / Master of Science at Agder University College, Faculty of Engineering and Science. This work was carried out under the supervision of associate professor Ole-Christoffer Granmo.

First of all we wish to thank our main supervisor Ole-Christoffer Granmo for first-class guidance throughout the project period. His insight and analytical skills has been greatly appreciated in numerous interesting discussions regarding the problem area targeted in this thesis.

We also wish to thank professor John B. Oommen for his role as a co-supervisor and for putting us on the track of the object partitioning solution.

Finally we wish to thank Head of Studies Stein Bergsmark for his advice regarding thesis writing and of course all of our co-students.

Grimstad, May 2005.

Svein Arild Myrer and Morten Goodwin Olsen

# Contents


**1 Introduction** **6**
- 1.1 Thesis definition . . . . . . . . . . 8
- 1.2 Report outline . . . . . . . . . . 8

**2 Learning automata** **10**
- 2.1 Learning automata . . . . . . . . . . 10
  - 2.1.1 Environment . . . . . . . . . . 10
  - 2.1.2 The learning automaton . . . . . . . . . . 12
  - 2.1.3 Stochastic learning automata . . . . . . . . . . 12

**3 The web environment and the objectives of learning** **14**
- 3.1 Web change characteristics . . . . . . . . . . 14
- 3.2 Environment model definition . . . . . . . . . . 15
  - 3.2.1 General environment model . . . . . . . . . . 15
  - 3.2.2 Environment where updated information disappear at once . . . . . . . . . . 16
  - 3.2.3 Environment where changes overwrite information . . . 17
- 3.3 The knapsack problem . . . . . . . . . . 17
  - 3.3.1 The fractional knapsack problem . . . . . . . . . . 18
  - 3.3.2 The binary knapsack problem . . . . . . . . . . 19
- 3.4 The objectives of learning . . . . . . . . . . 19

**4 Related learning algorithms** **21**
- 4.1 Parameter Optimization Problem . . . . . . . . . . 21
  - 4.1.1 Stochastic searching on the line . . . . . . . . . . 21
  - 4.1.2 Learning mechanism . . . . . . . . . . 22
- 4.2 Object Partitioning Problem . . . . . . . . . . 23
  - 4.2.1 Equipartitioning problem . . . . . . . . . . 23
  - 4.2.2 Object Migration Automaton . . . . . . . . . . 23

**5 Proposed learning algorithms** **26**
- 5.1 Competitive Game of Learning Automata (CGLA) . . . . . . 26
  - 5.1.1 Learning automaton description . . . . . . . . . . 27
  - 5.1.2 Competitive game . . . . . . . . . . 29

5.2 Fixed Partitioning Automaton (FPA) . . . . . . . . . . . . . . . 31
5.2.1 Learning automaton description . . . . . . . . . . . . . . 31

**6 Prototype** **35**
6.1 Requirements . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
6.2 Graphical user interface . . . . . . . . . . . . . . . . . . . . . . . 36

**7 Simulation results and discussion** **38**
7.1 Simulation setup . . . . . . . . . . . . . . . . . . . . . . . . . . 38
7.2 Performance metric . . . . . . . . . . . . . . . . . . . . . . . . . 39
7.3 Results and discussion of the results . . . . . . . . . . . . . . . . 40
7.3.1 Identifying good parameters . . . . . . . . . . . . . . . . 40
7.3.2 Comparison to alternative algorithms. . . . . . . . . . . . 44
7.3.3 Adaptability . . . . . . . . . . . . . . . . . . . . . . . . . 50

**8 Conclusion and further work** **54**
8.1 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54
8.2 Further work . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55

# List of Figures


2.1 General environment . . . . . . . . . . . . . . . . . . . . . . . . . . 11

3.1 Probability of discovering an update over time . . . . . . . . . . 16

4.1 Object Migration Automaton . . . . . . . . . . . . . . . . . . . . 24

5.1 Fixed structure stochastic automaton . . . . . . . . . . . . . . . 28
5.2 Fixed Partitioning Automaton . . . . . . . . . . . . . . . . . . . . 34

6.1 Prototype . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36

7.1 Update probability distribution . . . . . . . . . . . . . . . . . . . 39
7.2 Varying $N$ in the Competitive Game of Learning Automata . 41
7.3 Small $N$ and $f$ resolutions with Fixed Partitioning Automaton 42
7.4 Large $N$ and $f$ resolutions with Fixed Partitioning Automaton 43
7.5 Retrieved importance-value when changes overwrite information 46
7.6 Retrieved importance-value when updated information disappear at once . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46
7.7 Number of polls for the proposed solutions . . . . . . . . . . . 50
7.8 Automata performance in a non-stationary environment . . . 51

# List of Tables


2.1 Response set models . . . . . . . . . . . . . . . . . . . . . . . . 11

3.1 Fractional knapsack problem in relation to the incremental crawling task . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18

3.2 Binary knapsack in relation to the incremental crawling task . 19

5.1 Classes of $\boldsymbol{R}$ in the Fixed Partitioning Automaton . . . . . . . 31

6.1 Explanation of simulation settings . . . . . . . . . . . . . . . . . 36

# Chapter 1

# Introduction

Data mining applications often utilize crawlers (or spiders or robots or agents) in order to maintain an indexed and searchable local copy of the data sources which they monitor. An important property of this local copy or repository is to maintain data that are up to date at any given time. Ideally this means that changes in the data sources should be detected instantly and the local copies updated accordingly. This is however not feasible to accomplish for applications that monitor a large set of data sources, such as search engines, as they are most likely to encounter restrictions in form of available bandwidth and/or computational power. A general focus has therefore been to find and incorporate policies for crawling that can meet the various demands of data mining applications, such as the degree of consistency of the local copies [1].

There are two main strategies in use for keeping the local copy up to date. One is to simply repeat the initial crawling process and replace the old copy with the new one when the desired number of pages have been downloaded. The other is to update the local copy by selectively update parts of it in an incremental and continuous fashion. This re-crawling strategy is referred to as incremental crawling and crawlers that operates in this mode are so called incremental crawlers.

The concept of an incremental crawler was introduced by Junghoo Cho and Hector Garcia-Molina in [2], where they presented a general architecture for this type of crawlers. The architecture suggested that available resources should be selectively distributed through a refinement process, granting more resources to monitor some data sources and less to others. In [2] they also investigated the implications of an incremental crawler. Important issues such as bandwidth savings, faster discovery and retrieval of new data and improvement of the freshness of the local copy were pointed out to be the main advantages of the incremental scheme.

There have been many previous studies regarding incremental crawlers [3, 4, 5, 2, 6] since the concept was introduced. These efforts have in most cases been motivated by the need of search engines where maximizing the "freshness level"[1], or related metrics, are considered important.

However, a growing trend on the Internet for the past years is that information is becoming more and more dynamic [7]. This poses a new challenge for data mining operations since information now only might be available for a limited period of time as it either rapidly changes, gets replaced or becomes part of the deep Web[2]. The problem is especially significant for services where it is vital to deliver up to date information and to maintain a complete update history.

Companies that deliver such services often deploy so called continuous query engines that continuously monitor data streams in order to respond with relevant data to the query maker when given criteria are fulfilled. As an example we can look at the business intelligence industry. Their customers subscribe to certain information which are of value to them. A query can for instance be "notify me when my company is mentioned in the media". In order to respond to such type of queries the query engines have to interpret data streams emerging from a number of data sources. These data streams are most commonly created by a crawler that periodically visits and retrieves data from the sources of interest as the data sources seldom offer a push-based data feed.

For this new problem area traditional crawling strategies are generally sub-optimal. Their focus is to maximize the percentage of the local copy that is consistent at all times. In order to do so, fast changing data sources are not prioritized as these only contribute to keep consistency in a very limited period of time. A new class of methods are therefore needed.

We have in this thesis focused our work on solving the incremental crawling task when monitoring highly dynamic data sources of different importance. To our knowledge [8] and [9] are the only other related work done on this area.

[8] showed that traditional crawling schemes are not optimal when we monitor fast changing data sources. They also proposed new metrics that reflected the state of the information retrieved instead of the state of the local copy. Their suggested algorithm were designed to target only data sources of with very high probability of change.

[9] expanded the work in [8] by formalizing several update models for

[1] The freshness level give the percentage of the local copy that is up to date at any given time

[2] Data from the deep Web is dynamically produced only in response to a direct request to searchable databases

fast changing data sources. They also suggesting a basic greedy algorithm for the distribution of available resources.

Both of these approaches focus strictly on the scheduling of monitoring tasks. They assume that some important parameters, such as the frequency of change of the monitored data sources, are given and does not address methods for estimation of these parameters. The practical effectiveness of the proposed algorithms is in other words restrained by the effectiveness of unknown underlying algorithms. Accordingly, these approaches only offer a partial solution to the incremental web crawling problem.

In contrast to previous solutions, we will in this thesis consider incremental web crawling as a continuous learning problem where scheduling of monitoring tasks is combined with parameter estimation in an on-line manner. In particular our goal is to create an adaptive scheme that is able to respond as quickly as possible to behavioral changes in the monitored data sources. This is for instance important when reacting in real-time to e.g. stock market changes or increases of maximum bids of online auctions.

Our novel approach maps the incremental crawling task to the so called knapsack problem. We propose two solutions to the resulting knapsack problems based on the concept of learning automata. We also show the effectiveness of the suggested solutions in a non-stationary environment and in comparison with other algorithms in a stationary environment.

The thesis definition and further outline of the thesis is given in the next section.

## 1.1 Thesis definition

The final thesis definition is formulated like this:

*The students will approach the new problem area of monitoring highly dynamic data sources of different importance in the context of an incremental web crawler. By mapping the incremental crawling task to the knapsack problem the students will explore the use of learning automata in order to solve this problem. They will furthermore evaluate the performance of the resulting learning algorithms in a non-stationary environment and by comparing them with other relevant algorithms in a stationary environment.*

## 1.2 Report outline

This thesis is outlined in the following way:

**Chapter 1** is the introduction which is the current chapter.

**Chapter 2** covers the basic theory about learning automata as a foundation for understanding the later proposed solutions.

**Chapter 3** defines the environment model which our learning algorithms are going to operate in. This chapter also maps the incremental crawling task to two variants of the knapsack problem and defines the objectives of learning.

**Chapter 4** summarizes previous learning automata solutions to the parameter optimizing problem and the object partitioning problem.

**Chapter 5** extend the learning automata solutions summed up in chapter 4 and proposes solutions for the fractional and binary knapsack problem in the context of the incremental crawling task.

**Chapter 6** describes the prototype developed for simulation purposes.

**Chapter 7** presents the simulation setup, the chosen performance metric, the results from our simulations and a discussion of the results.

**Chapter 8** gives the conclusion of our work. We also look at possible further work this thesis may be a basis for.

# Chapter 2

# Learning automata

This chapter introduces the basic background theory of learning automata and is fundamental for understanding our proposed solutions.

## 2.1 Learning automata

The work on learning automata was pioneered by M.L Tsetlin and his co-workers in the Soviet Union in the 1960s. There have since then been many advances in both the theory itself and the application of the theory [10]. Learning automata are a part of the field of artificial intelligence known as machine learning where the behavior of computer programs is not fixed, but the result of a learning process. In the field of psychology they define learning as "behavioral modification especially through experience or conditioning." [11]. For a learning automaton the behavioral modification happens through repeatedly interaction with an environment and the feedback this results in.

### 2.1.1 Environment

The correlation between actions taken by the automaton and resulting feedback from the environment is considered to be hidden. This includes all time varying changes that may occur. The only way an automaton can explore its environment is through its finite set of available actions. These actions are considered as input to the environment from which the environment produces a random output or response from its set of available responses, as illustrated in figure 2.1. The feedback is either favorable or unfavorable and is probabilistically related to the performed action.

To formally describe the environment we will use the definition stated in [12]. This definition is also commonly used in the literature.

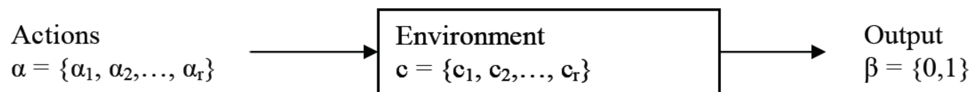


Figure 2.1: A general illustration of an environment taken from [12]

The environment is defined mathematically by a triple $\{\boldsymbol{\alpha}, \boldsymbol{c}, \boldsymbol{\beta}\}$.

- $\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, ..., \alpha_r\}$ represents a finite set of actions being the input to the environment
- $\boldsymbol{\beta} = \{\beta_1, \beta_2\}$ represents an output set, in this case binary.
- $\boldsymbol{c} = \{c_1, c_2, ..., c_r\}$ is a set of penalty probabilities, where $c_i$ of $\boldsymbol{c}$ corresponds to one input action $\alpha_i$.

To elaborate, when an action $\alpha(n)$, belonging to the set $\boldsymbol{a}$, is performed at time $n$ the environment produces a response $\beta(n)$, belonging to the binary set $\boldsymbol{\beta}$. The most basic response set $\boldsymbol{\beta}$ is defined to be $\{0, 1\}$ where 1 indicate a penalty and 0 a reward. The element $c_i$ of $\boldsymbol{c}$ can now be defined mathematically as

$$Pr(\beta(n) = 1|\alpha(n) = \alpha_i) = c_i \qquad (i = 1, 2, ..., r)$$

This definition tells us that the probability of penalty when choosing action $\alpha_i$ is $c_i$. When all elements in $\boldsymbol{c}$ are constant over time we consider the environment to be stationary. Otherwise we consider it to be non-stationary.

So far we have only looked at a binary response set. This is known in literature [12] as the P-model. There exists two other models that define other types of response sets. The characteristics of these two models and the P-model is summarized in table 2.1. The S-model and the Q-model will not be considered further in this paper.

| Model name | Values |
|---|---|
| P-Model | Either 0 or 1 |
| S-Model | Continuous values in the range (0, 1) |
| Q-Model | Finite set of discrete values in the range (0, 1) |

Table 2.1: Response set models

### 2.1.2 The learning automaton

To formally describe a learning automaton we will use the definition stated in [12]. This definition is also commonly used in other literature. The automaton is defined mathematically by the quintuple $\{\Phi, \boldsymbol{\alpha}, \boldsymbol{\beta}, F(\bullet,\bullet), H(\bullet,\bullet)\}$

1. The state of an automaton at any instant $n$, denoted by $\phi(n)$, is an element of the finite set $\Phi = \{\phi_1, \phi_2, ..., \phi_s\}$
2. The output or action of an automaton at the instant $n$, denoted by $\alpha(n)$, is an element of the finite set $\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, ..., \alpha_r\}$
3. The input of an automaton at the instant n, denoted by $\beta(n)$, is an element of the set $\boldsymbol{\beta}$. This set could either be a finite set or an infinite set, such as an interval on a real line. $\boldsymbol{\beta} = \{\beta_1, \beta_2, ..., \beta_m\}$ or $\boldsymbol{\beta} = \{(a, b)\}$ where a,b are real numbers.
4. The transition function $F(\bullet,\bullet)$ determines the automaton state at the instant (n+1) in terms of the state and input to the automaton at the instant $n$, i.e, we get $\phi(n+1) = F[\phi(n), \beta(n)]$. F can also be described as a mapping from $\Phi$ x $\boldsymbol{\beta} \rightarrow \Phi$ and can either be deterministic or stochastic.
5. The output function $H(\bullet,\bullet)$ maps the current state and input into the current ouput. However, if the current output depends on only the current state, the automaton is referred to as a state-ouput automaton. In such a case we replace $H(\bullet,\bullet)$ with $G(\bullet) : \Phi \rightarrow \boldsymbol{\alpha}$. Also expressed as $\alpha(n) = G[\Phi(n)]$ and can also either be stochastic or deterministic.

### 2.1.3 Stochastic learning automata

We have a stochastic automaton if either one of the F and G mappings are stochastic.

If F is stochastic, it can be defined in terms of transition probabilities $f^{\beta}_{ij}$ The value of $f^{\beta}_{ij}$ gives us the probability of moving from $\phi_i$ to $\phi_j$ given input $\beta$. Mathematically this is expressed as:

$$f^{\beta}_{ij} = Pr\left\{\phi(n+1) = \phi_j | \phi(n) = \phi_i, \beta(n) = \beta\right\} \qquad (i, j = 1, 2..., s \;\; \beta = \beta_1, \beta_2, ..., \beta_m)$$

If G is stochastic, it can be defined in terms of output probabilities $g_{ij}$. The value of $g_{ij}$ gives us the probability of performing action $\alpha_j$ given that the automaton is in $\phi_i$. Mathematically this is expressed as:

$$g_{ij} = Pr\left\{\alpha(n) = \alpha_i | \phi(n) = \phi_i\right\} \qquad (i, j = 1, 2..., r)$$

The stochastic learning automaton has two variants, namely the fixed structure automaton and the variable structure automaton. If the probabilities of $f_{ij}$ and $g_{ij}$ does not vary over time we have a fixed structure automata. If this is not the case, we have a variable structure automata. The variable structure automata updates its probability values through a given reinforcement scheme. As it is beyond the scope of this thesis to further explain the properties of fixed structure and variable structure and other variants of learning automata we refer the reader to [12] for an extensive view of the topic.

# Chapter 3

# The web environment and the objectives of learning

To determine the environment and its characteristics is fundamental when designing a learning algorithm. We will in this section first look at web change characteristics that previous studies have discovered. We will then formally define the environment our automata solutions are going to operate in. Furthermore we relate the incremental crawling task to two variants of the so called knapsack problem and define the goals of our learning algorithms based on these variants.

## 3.1 Web change characteristics

For an incremental crawler to effectively prioritize its available resources it must have some notion of the update behaviors of the data sources which it monitors. This problem has been the origin of several larger and smaller scale studies on how the web evolves over time [13, 14, 2, 15, 16] and whether the update patterns of web pages could be modeled mathematically through a unified model.

The findings in [2] suggested that the update patterns of web pages followed a memoryless Poisson process. This assumption was to some extent confirmed in [15] and has been the de facto model for several studies on estimating the frequency of change [17, 18, 6]. However, later studies on web change characteristics has although shown that the assumption of a Poisson process in many cases is not correct. Brewington and Cybernenko undermines the findings of J. Cho and H. Garcia-Molina in [2] by showing that 65 percent of the documents in their observed subset of the Web changes during US working hours (5 a.m. to 5 p.m., Pacific Standard Time) [14] and

accordingly suggest to model web document change as a so called renewal process.

Studies directed toward web pages that serves highly accessed web sites with highly dynamic content, such as news service web sites, real-time stock market figures, show that the update processes can span over a high range of statistical models [13, 16]. We can therefore neither assume an underlying Poisson model nor a renewal process for these kind of situations. Instead a quasi deterministic model were each web page has an associated probability of change at each time slot has been shown to be sufficient to model frequently updated data sources [16, 13]. This model has been the underlying model for the work done in [9, 8] which is directly related to our problem.

## 3.2 Environment model definition

### 3.2.1 General environment model

We model the world wide web as a set of data sources or nodes with independent update behaviors as no studies show any conclusive correlation between the update patterns of different web pages. Note that in this model each node can either represent a single web page or even a set of web pages. Furthermore we disregard the hyperlink structure of the web as the re-crawling process works by a set of already discovered urls. Last we consider the number of data sources to be fixed.

Now, let D be the set of nodes $d_i$ we wish to monitor. As we want to design a learning system for fast changing dynamic environments we apply the quasi deterministic model explained in 3.1 to define the update behavior of these nodes

Consider a sequence of discrete time slots $T = \{t_1, t_2, ..., t_j\}$. Each node has an associated probability $p_{i,j}$ which determines the probability that an update of data source $i$ at time slot $j$ occurs. This probability of change is considered hidden for anyone but the node itself. For simplicity, we have chosen $p_{i,1} = p_{i,2} = p_{i,3} = p_{i,j}$ for any j.

Attached to each node $d_i$ there is a cost denoting the resources needed to visit and download its content. Lets denote this cost as $c_i$. The cost is considered equal for all nodes; $d_i$, $c_1 = c_2 = c_i = 1$ for any i, as typical in other related work [9, 8].

Each node has also an associated importance-value $\omega_i \in [0, 1]$ denoting the importance of capturing any changes to this node. This importance-value is set during the first download of a page by using appropriate relevance measures. The importance-value is considered to be known.

As defined, an update may occur with a certain probability at a given time slot. The model described so far does however not delineate whether the new information is still present or just disappears at the next time slot. We therefore split the environment definition in two in order to address these two important sub-models individually.

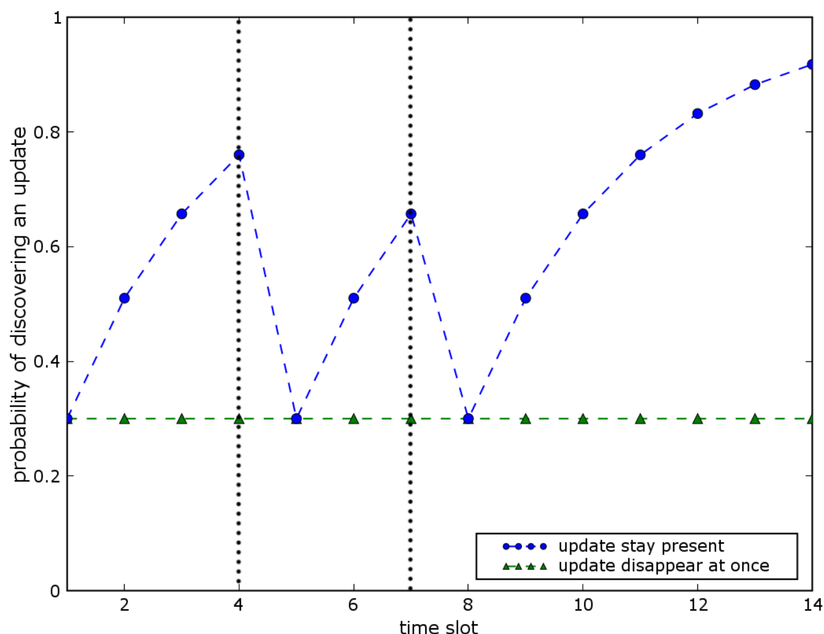


Figure 3.1: Discovery of an update happens in time slot 4 and 7. Figure shows the probability of finding an update if we crawl data source $d_i$ at time j given the probability of update $p_{i,j}$. Note that we operate with discrete time slots so we also have discrete probability values of discovering an update, not continuous as the figure might indicate.

### 3.2.2 Environment where updated information disappear at once

If updates are only present in the time slot they occur, the probability of discovering an update is constant and only related to $p_{i,j}$. This can for instance be time-limited offers at online-shops which only spans the length of the period defined for each $t_i \in T$

We will in this model always have a constant probability of discovering an update at each time slot. This is illustrated in figure 3.1 where $p_{i,j} = 0.3$.

### 3.2.3 Environment where changes overwrite information

In this environment an update overwrites the information posted by the preceding update. This can for instance be the case when monitoring maximum bid for online auctions and stock auctions. Note that when $p_{i,j} = 1$ we get the same situation as in 3.2.2.

With this model the probability of discovering an update will increase from the last time we checked for an update. Let the function $L(d_i)$ give us the latest time slot where we visited node $d_i$. A formal expression for the probability of finding an update of data source $d_i$ in time slot $t_j$ can now be defined as $P(L(d_i), t_j) = 1 - \prod_{L(d_i)}^{t_j}(1 - p_{i,j})$.

This model illustrated in figure 3.1 where $p_{i,j} = 0.3$

## 3.3 The knapsack problem

The ideal situation for any crawler is to have enough resources available so it can constantly revisit all of its monitored data, capturing all occurrences of updates. The crawling task would then be considered trivial. However, this ideal situation is not feasible to assume in a real world situation due to constraints in form of bandwidth and/or computational power.

The so called knapsack problem is a popular combinatorial optimization problem which arises whenever we meet a resource allocation problem imposed by some constraint. This classical problem is often illustrated by a thief having a knapsack with a given capacity C. He has just broken into a museum and can choose from a set of J items having different volumes, $w_i > 0$ , and different values, $v_i > 0$. How can he fill his knapsack with the most valuable set of items without exceeding the capacity of his knapsack? Or, seen in relation to the incremental crawler, how should we spend our available resources on the data sources we monitor in order to maximize the utilization according to the set goals?

There exist several variants of the knapsack problem. All of these relate somewhat to the incremental crawling task as they impose the same constraints and goals. However, the two variants we are going consider are the fractional knapsack problem and the binary knapsack problem as these deal with single instances of each item of each kind, just as a crawler only deals with one web page of each kind, not multiple instances of the same web page.

### 3.3.1 The fractional knapsack problem

| Symbol | Knapsack | Web |
|---|---|---|
| $C$ | Knapsack size | Crawler capacity |
| $x_i$ | Fraction of item i | Polling rate of web page |
| $v_i$ | Value of item i | Update rate of page weighted against its importance value |
| $w_i$ | Volume of item i | Cost of polling web page i |
| $J$ | Number of items | Number of web pages |

Table 3.1: Fractional knapsack problem in relation to the incremental crawling task

With the fractional knapsack problem the thief does not necessarily need to put a whole item in his knapsack, only a fraction of it. Each of the items the thief wishes to steal can be related to an external data source, such as a web page, which is a candidate for polling and downloading. The value of an item can be seen as a page's probability of change weighted against its given importance-value. Additionally, a polling frequency of a certain web page is similar to selecting a certain fraction of an item. As an example, if we choose a fraction of 100% of an item, we choose to poll it every time slot. Similarily, choosing a fraction $\frac{1}{N}$ of a web page means that we poll a page on average every $\frac{N}{1}$ time slot. Furthermore the size of the knapsack is similar to the available capacity of a crawler and the weight of an item is seen in relation to the cost of polling a given web page.

Table 3.1 summarizes the relations between the fractional knapsack problem and the incremental crawling task.

A general solution for solving the fractional knapsack problem is by using the following greedy algorithm: *Take as much as possible of the material that is most valuable per unit volume. If there is still room, take as much as possible of the next most valuable material. Continue until the knapsack is full.* However, the above strategy can clearly not be used in our mapping of the fractional knapsack problem as the item values are connected to the update rates of the monitored web pages which are unknown. The item values must then be considered to be random variables with unknown distribution that also may change over time. This extension of the fractional knapsack has to the best of our knowledge not been addressed in the literature before and we will in section 5.1 propose a solution for filling such a knapsack.

### 3.3.2 The binary knapsack problem

| Symbol | Knapsack | Web |
|---|---|---|
| $C$ | Knapsack size | Crawler capacity |
| $x_i$ | 1 if item i is put in knapsack, 0 otherwise | 1 if web page i is always polled, 0 otherwise |
| $v_i$ | Value of item i | Update rate of page weighted against its importance value |
| $w_i$ | Volume of item i | Cost of polling web page i |
| $J$ | Number of items | Number of web pages |

Table 3.2: Binary knapsack in relation to the incremental crawling task

The binary knapsack problem enforces us to either entirely put an item in the knapsack or not put it in at all. We must still, as with the fractional knapsack problem, consider the item values as random variables with unknown distribution as they are connected to the unknown update rates of web pages here as well. As we in section 3.2 defined the cost of polling a web page to be equal for all web pages, the binary knapsack problem related to the incremental crawling task is solved by identifying the most valuable web pages to put in the knapsack. All resources will then be spent on polling these pages. A proposed solution for solving this problem is presented in section 5.2.

## 3.4 The objectives of learning

The objectives of our learning algorithms are naturally connected to the common constraints and goals imposed by the knapsack problem which we related to the incremental crawling task in 3.3

First, let $y_{i,j}$ denote whether we choose to visit node i at time j. This is a binary value and is defined as

$$y_{i,j} = \begin{cases} 1 \text{ if node i is crawled at time instance j} \\ 0 \text{ otherwise.} \end{cases}$$

As we consider each node to have equal cost the resource constraint can now be formally expressed as

$$\sum_{i \in D_i} y_{i,j} = C$$

where C denotes the total number of monitoring tasks available at time j, or the capacity of the knapsack.

We define a poll to be the process of downloading a web page and putting it in a the local repository. An successful poll is considered to happen when we download a web page that has been altered since the last time it was polled. We call this as a hit and define it as a function $H(\bullet, \bullet)$

$$H(d_i, j) = \begin{cases} 1 \text{ if } d_i \text{ was updated since the last visit} \\ 0 \text{ otherwise.} \end{cases}$$

We can now formally define the goal of maximizing the retrieved importance-value as

$$\sum_{d_i \in D} \omega_i y_{i,j} H(d_i, j)$$

working under the constraint of C at each time slot j.

# Chapter 4

# Related learning algorithms

This chapter summarizes two previously proposed learning algorithms for solving the so called parameter optimization problem and the object partitioning problem by using learning automata. As we in chapter 5 extend these algorithms to solve the fractional knapsack problem and binary knapsack problem in relation to the incremental crawling task, it is essential to get an understanding of how these algorithms operate.

Section 4.1 summarize the parameter optimization problem and a learning automata solution initially presented by Oommen in [19].

In 4.2 we go on and sum up the object partitioning problem and learning automata solution presented by Oommen and Ma [20].

## 4.1 Parameter Optimization Problem

The parameter optimization problem plays a role whenever it is wishful to accomplish a given task with minimum cost or maximum benefit, and the cost or benefit is dependent on one or more incoming parameters, as in image processing and neural networks [19].

### 4.1.1 Stochastic searching on the line

One possible solution to the parameter optimization problem was introduced by John B. Oommen in [19] by demonstrating a learning mechanism determining a point on a line in a random environment where the environment gives possible erroneously guidance by directing the movement of the point.

The scheme involved discritizing the space and performing controlled random walks on this space. The solution presented is $\epsilon$-optimal, meaning that if there exists a parameter providing the best result, the schema will con-

verge toward a value arbitrarily close to this parameter. The next section elaborates the learning mechanism.

### 4.1.2 Learning mechanism

As mentioned, the goal of the learning mechanism is to find the optimal value of a parameter $\lambda$ where the unknown value of which it should adapt to is $\lambda^*$. Even though the mechanism does not know the value of $\lambda^*$ it receives guidance from an intelligent environment which is capable of informing the mechanism with the information of whether the chosen $\lambda$ is too big or too small.

However the environment might respond with biased feedback as long as the feedback is correct with a probability $p > 0.5$. $p$ reflects the effectiveness of the environment, as $p$ close to 1 will indicate that the environment gives the correct feedback most of the time. In contrast $p$ close to 0.5 will indicate that the environment gives the correct feedback on average only every second time. The larger $p$ is, the faster the mechanism will converge.

Additionally, the mechanism assumes that $\lambda$ is a number in $[0, 1]$ and discritizes the $[0, 1]$ interval into $N$ steps $\{0, \frac{1}{N}, \frac{2}{N}, ..., \frac{(N-1)}{N}, 1\}$ where $N$ is the resolution of the schema. This means that a larger $N$ will give a more accurate $\lambda$, however it might converge slower than a small $N$.

The mechanism can be presented as an automaton defined as the following:

1. The states are defined as $\phi = \{\phi_1, \phi_2, ..., \phi_N\}$
2. The actions are defined as $\alpha = \{\alpha_1, \alpha_2, ..., \alpha_N\}$
3. The input $\beta$ is defined as $\{0, 1\}$ where 0 suggests a decrease of $\lambda$ and 1 suggests an increase
4. The transition matrix of $F(\phi(n), \beta(n))$ is defined as:
$$\beta(n) = 1 \begin{cases} \phi(n+1) = \phi(n) + 1 & \text{if } 0 < \phi(n) < N - 1 \\ \phi(n+1) = \phi(n) & \text{if } \phi(n) = N \end{cases}$$
$$\beta(n) = 0 \begin{cases} \phi(n+1) = \phi(n) - 1 & \text{if } 0 < \phi(n) < N \\ \phi(n+1) = \phi(n) & \text{if } \phi(n) = 0 \end{cases}$$
5. The output function $G(\phi(n))$ is defined as $\phi_i \rightarrow \alpha_i$

Each state has a corresponding action with an assumption of $\lambda$. A given state, say state $\phi_i$, has the corresponding action $\alpha_i$ with the assumption of $\lambda$ at $\frac{i}{N}$. In other words the states can be organized in a chain $\{0, 1, 2, 3, ..., N-1, N\}$ corresponding to values for $\lambda$ as $\{0, \frac{1}{N}, \frac{2}{N}, \frac{3}{N}, ..., \frac{N-1}{N}, 1\}$.

However if $\lambda$ is not in the interval $[0, 1]$, $\lambda$ can clearly be transformed by a normalizing function. As an example; if the parameter that should be optimized, $\lambda$, lies between $\lambda_{\min}$ and $\lambda_{\max}$, the normalization transitions will be defined as:

- $\lambda_{\text{norm}} = \frac{\lambda - \lambda_{\min}}{\lambda_{\max} - \lambda_{\min}}$
- $\lambda = \lambda_{\text{norm}} * (\lambda_{\max} - \lambda_{\min}) + \lambda_{\min}$

## 4.2 Object Partitioning Problem

The object partitioning problem arises when there exists a set of $W$ objects, $\mathbf{\Omega} = \{A_1, ...A_W\}$, which should be partitioned into a set of $R$ classes, $\{P_1, ..., P_R\}$. The overall intention is to partition the objects which are accessed more frequently together into the same class, and at the same time partition the objects which are less frequently accessed together in different classes. Object partitioning plays a role in situations such as physical distribution of attributes, distributed databases, partition allocation and document placement in a libraries [20, 21, 22], or whenever there exists an underlying physical grouping of the objects $\mathbf{\Omega}$. The object partitioning problem has been shown to be NP-hard [22].

### 4.2.1 Equipartitioning problem

Generally the object partitioning problem does not dictate whether the partitions should be of equal or fixed size. However, a variant of the object partitioning problem is called the equi-partitioning problem. This problem appears when there exists a set of $W$ objects, $\mathbf{\Omega} = \{A_1, ..., A_W\}$, which should be partitioned into a set of $\boldsymbol{R} = \{P_1, ..., P_N\}$ classes with fixed and equal amount of objects in each class.

### 4.2.2 Object Migration Automaton

A solution to the equipartitioning problem using deterministic learning automata was presented by Oommen and Ma in [20]. The solution involved an introduction of a new automaton named the object migration automaton.

The object migration automaton is defined as a quintuple ($\boldsymbol{\alpha}$, $\boldsymbol{\phi}$, $\boldsymbol{\beta}$, F, G) where

1. $\boldsymbol{\alpha} = \{\alpha_1, ..., \alpha_W\}$ is defined as the set of available action, whereas each action represents a class. The abstract objects must choose from one of these actions.

2. $\mathbf{\Phi} = \{\phi_1, ..., \phi_{WN}\}$ is a set of WN states. Each action $\alpha_p$ has N associated states.

3. $\boldsymbol{\beta} = \{1, 0\}$ is the set of inputs from the environment.

4. The transition matrix F is defined as followed:
$$\beta = 0 \begin{cases} \epsilon_k = \epsilon_k - 1 \text{ if } (\epsilon_k \bmod n) \neq 1, & \text{for k=i,j} \\ \epsilon_k = \epsilon_k \text{ if } (\epsilon_k \bmod n) = 1, & \text{for k=i,j} \end{cases}$$
$$\beta = 1 \begin{cases} \epsilon_k = \epsilon_k + 1 \text{ if } (\epsilon_k \bmod n) \neq 0, & \text{for k=i,j} \\ \epsilon_k = \lceil \frac{\epsilon_k}{N} \rceil * N \text{ if } (\epsilon_k \bmod n) = 0, & \text{for k = i,j} \\ & \text{for m=i,j} \\ & \text{for } k \neq \text{m} \end{cases}$$
Where $\epsilon_i$ and $\epsilon_j$ is the states of the abstract object $O_i$ and $O_j$ and the query is $(A_i, A_j)$.

5. The output matrix G is defined by
$p = \lceil \frac{\epsilon_k}{N} \rceil$ for $1 \leq i \leq W$
where $O_i$ is in state $\epsilon_i$ and $\alpha_p$ is the action $O_i$ is in.

The heart of the automaton involves creating an abstract object for each physical objects, and letting the abstract objects move around within a fixed number of classes. Each class represents an action and contains an equal amount of abstract objects. Additionally the actions have a set of states ranging from the most internal state to a so called boundary state.

An object migration automaton with two actions and N states for each action is outlined in figure 4.1. In the figure $\phi_{m1}$ and $\phi_{k1}$ are the most internal states while $\phi_{mN}$ and $\phi_{kN}$ are boundary states. Each state in figure 4.1 contains zero or more abstract objects, while the number of abstract objects in $\phi k$ equals the number of abstract objects in $\phi m$.

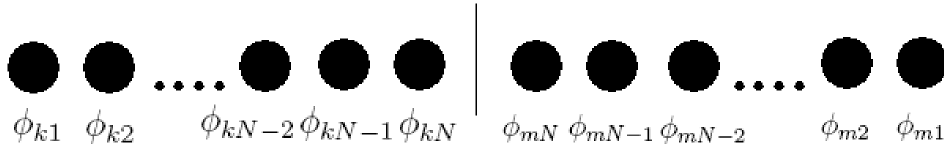


Figure 4.1: Object Migration Automaton with two actions and 2N states

When two physical objects are accessed together, the abstract objects are rewarded if they lie in the same class, and punished otherwise. If two abstract objects are rewarded they move one state closer toward the most internal state. However if the two objects are punished, say $O_i$ and $O_j$, there are several possible outcomes depending on the states of the abstract objects.

1. If neither $O_i$ nor $O_j$ are in boundary states they move one step closer to the boundary state.

2. If exactly one of the abstract objects is in a boundary state, only the other object is moved toward the boundary state of its class.

3. If both are in boundary states one of the abstract objects, say $O_i$ will be moved to the boundary state of the class of $O_j$. In addition an abstract object from the class of $O_j$, which is in boundary state and is not $O_j$, will be moved to the class of $O_i$.

When every object have been positioned in internal states the automaton has converged and the objects can be physically organized according to the position of the abstract objects of the automaton.

In [20] the object migration automaton is shown to be expedient and indicated to be $\epsilon$ optimal in addition to quicker than the best known equipartitioning algorithms in literature.

# Chapter 5

# Proposed learning algorithms

In this chapter we present algorithms that provide adaptive schemes for the recrawling-policy of an incremental web crawler. Our aim has been to develop algorithms that are able to work in a non-stationary environment and that improves their performance through a learning process.

In section 5.1 we propose a scheme using fixed structure stochastic learning automata connected together in a competitive game making the automata compete for the available resources provided by the crawler environment. This solution is constructed to solve the fractional knapsack problem seen in relation to the incremental crawling task as described in 3.3.1.

Additionally, we propose in section 5.2 a scheme using a Fixed Partitioning Automaton distinguishing which of the monitored data sources that are most valuable and focusing our capacity on polling these sources. Here we propose a solution to the binary knapsack problem seen in relation to the incremental crawling task as described in 3.3.2.

## 5.1 Competitive Game of Learning Automata (CGLA)

The algorithm presented in this section is an extension of the learning algorithm elucidated in section 4.1.2 for solving the parameter optimization problem.

In this algorithm we link fixed structure stochastic learning automata to each of the monitored nodes working together in a competitive game. From a parameter optimization point of view each automaton is then responsible to find a fraction of total available resources that should be used in order to monitor it's corresponding data source. However it is essential that the total number of polls does not exceed the available capacity provided by

the crawler environment. This is ensured by connecting the automata in a competitive game and let them compete for the available capacity.

In relation to the fractional knapsack problem we attempt to fill the fractional knapsack with fraction $x_i$ of each available item $i$ where $1 \leq i \leq n$ and $n$ is the number of elements. Our aim is therefore to maximize the expected total value of the fractions without exceeding the size of the knapsack ($\sum_i x_i \leq C$) by connecting the fraction amount of each item with a learning automaton.

### 5.1.1 Learning automaton description

For the sake of clarity we will in this section only focus on explaining the functionality of one automaton interacting with an environment. This means that we will only focus on adapting to feasible fractions based on the feedback received from the environment and not focus on the functionality of several automata interacting. A connection of the automata will be described in section 5.1.2, where the environment feedback is based on the restrictions by the knapsack size.

To further elaborate, the automaton attempts to adapt a parameter $\lambda$ to the most feasible fraction amount guided by an intelligent environment. Additionally it is implicit that when $\lambda$ is close to $\lambda^*$ the parameter is close to the most feasible solution. As the solution to the parameter optimization problem suggests, we discretize the possible values of $\lambda$ into fractions of $\{\frac{1}{N}, , ..., \frac{(N-1)}{N}, 1\}$ where $N$ is the resolution of the scheme. This means that when the automaton is in state $\phi_i$ it chooses fraction $\frac{i}{N}$. Note that our discretization differs somewhat from the solution summarized in 4.1 as the minimum fraction is defined to be $\frac{1}{N}$ and not 0. A minimum fraction of 0 would result in an absorbing scheme as we never would chose to poll an external data source if we assign this fraction to it. The scheme would therefore not be suitable for a non-stationary environment.

**Learning mechanism**

We consider the environment to produce responses from the binary response set defined by the P-model described in section 2.1.1. Whenever $\beta = 1$ the automaton has chosen to poll a data source which has not been updated since the last poll, which for that reason is considered to be an indication that the fraction amount should be decreased. In contrast, whenever $\beta = 0$, the automaton has decided to poll an external data source which has been updated since the last poll, and this is interpreted as the fraction amount is exactly correct or should increase.

As a node $d_i$ has a corresponding importance-value $\omega_i \in [0, 1]$ , and we wish to prioritize the nodes with the highest values of $\omega_i$, the transitions should take the received feedback, $\beta$, and node's importance-value into account. It should be easier to increase the fraction of a node with a high importance-value, than increasing the fraction of a node with low importance-value.

As a consequence, the automaton choses a state with a corresponding action higher than the current state whenever $\beta = 0$ with the probability $\omega_i$ and choses the same state with probability $1 - \omega_i$. This ensures that it is easier for the automata to increase the fraction amount whenever $\omega_i$ is high. In contrast whenever $\beta = 1$ the automaton chooses a state with the corresponding lower action with the probability 1. If we assume that the automaton is in state $\phi_j$, $\beta = 0$ leads to the next state being $\phi_{j+1}$ with the probability $\omega_i$, while $\beta = 1$ leads to the next state being $\phi_{j-1}$.

Additionally, the automaton can not choose states higher than $\phi_N$ or lower than $\phi_0$. When the current state of the automaton is $\phi_N$ and $\beta = 0$ the next state will never be $\phi_{N+1}$ but $\phi_N$. Furthermore, whenever the automaton is in state $\phi_0$ and $\beta = 1$ the next state will not be $\phi_{0-1}$ but $\phi_0$.

The transition function $F(\phi(n), \beta(n))$ would therefore be defined as:

$$
\begin{array}{lll}
\beta(n) = 1 \{ & \phi_i(n+1) = \phi_i(n) + 1 & \text{if } \phi_i(n) < N \text{ with the probability } \omega_i \\
\beta(n) = 0 \{ & \phi_i(n+1) = \phi_i(n) - 1 & \text{if } \phi(n) > 1 \\
 & \phi_i(n+1) = \phi_i(n) & \text{otherwise}
\end{array}
$$

while the fraction corresponding with each state would be $\frac{\phi_i(n)}{N}$.

Figure 5.1 shows a graphical representation of the fixed structure stochastic automaton.

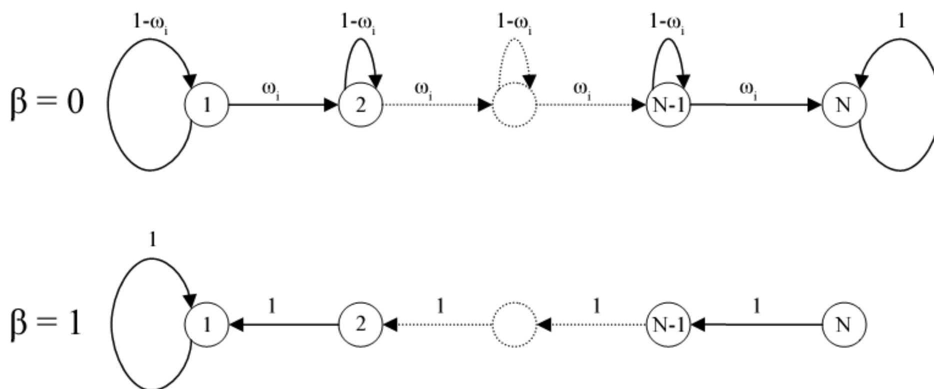


Figure 5.1: Fixed structure stochastic automaton

### 5.1.2 Competitive game

By connecting each item with a learning automaton without any further limitations, some unwanted situations may emerge. It is possible, and even likely, that the knapsack size in general would be exceeded as the sum of each fraction chosen could be greater than the knapsack size. Additionally it is also possible that the entire knapsack is never utilized.

If the automata mostly receive $\beta = 0$, they will at most time slots increase their fraction amounts. This could easily lead to an average of $\sum_i x_i > C$, which clearly is an unwanted situation as the knapsack would be exceeded at most time slots. In contrast, if the automata mostly receive $\beta = 1$, they will at most time slots decrease their fraction amounts. This would in general lead to $\sum_i x_i < C$, which is an unwelcome situation as we wish to fill the knapsack as tightly as possible.

In order to avoid these situations, we organize the automata in a competitive game making the automata compete for the available capacity.

The competition is governed by two situations:

- There is still room in the knapsack
- The capacity of the knapsack has been exceeded

The competition is as following; rewards are handed out whenever there is still room in the knapsack and in contrast punishments are only handed out while the capacity has been exceeded. A direct result of this is that each automata can only decrease the fraction amount whenever the capacity is exceeded and increase the fraction amount whenever there is still room in the knapsack.

This results in a competitive game of learning automata as there is only a certain amount of capacity available and each automata competes for the available capacity.

In order to clarify we present the possible scenarios in the game of automata.

**Scenario 5.1.1** ***The knapsack is not exceeded***
*The knapsack is not exceeded, meaning that the sum of each fraction is less or equal than the capacity of the knapsack ($\sum_i x_i \leq c$). Whenever each automaton receives a reward they increase their fraction amount. Punishments are never handed out when the knapsack is not exceeded.*

**Scenario 5.1.2** ***The knapsack is exceeded***
*The knapsack is exceeded, meaning that the sum of each fraction is greater than the capacity of the knapsack ($\sum_i x_i > c$). Whenever each automaton*

*receives a punishment they decrease their fraction amount. Rewards are never handed out when the knapsack is exceeded.*

**Scenario 5.1.3** ***The knapsack is nearly filled***
*The knapsack is nearly filled and there is only room for one automaton to increase its fraction amount without the knapsack being exceeded. Several outcomes can emerge from this situation depending on how many automata increase their fraction amount.*

- *If two or more automata receive hits they will increase their fraction amount. This will result in the knapsack being exceeded and only punishments will be handed out of which the automata can again only decrease their fraction amount. This is the most common outcome and will result in the sum of the fractions fluctuating around the knapsack size.*
- *Only one automaton receives a hit and increase its fraction amount. In this scenario the knapsack will not be exceeded and additional automata may increase their fraction amount at the next time slot.*
- *If no automata receive hits, no automata will increase their fraction this will result in the total knapsack size not being exceeded.*

In scenario 5.1.3 it is clearly most beneficial that only one automaton increases its fraction amount, since this is the only outcome where the knapsack is filled and the outcome giving the most value. In other words only one automaton can increase its fraction amount without the knapsack being exceeded demonstrating the competition for the available capacity between the automata.

By extending the transition function $F(\phi(n), \beta(n))$ presented in 5.1.1 with the knapsack limitations, we are able to arrange the automata as a competitive game.

$$\begin{array}{lll} \beta(n) = 0 & \{\ \phi_i(n+1) = \phi_i(n) + 1 \text{ with probability } \omega_i & \text{if } \phi_i(n) < N \text{ and } \sum x_i \leq C \\ \beta(n) = 1 & \{\ \phi_i(n+1) = \phi(n) - 1 & \text{if } \phi_i(n) > 1 \text{ and } \sum x_i > C \\ & \phi_i(n+1) = \phi_i(n) & \text{otherwise} \end{array}$$

## 5.2 Fixed Partitioning Automaton (FPA)

In this chapter we extend the Object Migration Automaton summarized in chapter 4.2 in order to solve the binary knapsack problem in relation to the incremental crawling task described in section 3.3.2. We call the new automaton the Fixed Partitioning Automaton, as we design it to partition objects into fixed sized classes.

To elaborate we wish to partition the set of nodes, $\boldsymbol{D}$, into $\boldsymbol{R}$ classes where $\boldsymbol{R} = \{P_1, P_2\}$ and $P_1$ has the size $M$ and $P_2$ has the size $N - M$. In this case $N$ is the number of nodes or the size of $\boldsymbol{D}$ and $M$ the size of the knapsack.

Note that this problem resembles the equi-partitioning problem. However the equi-partitioning problem is defined as partitioning N objects into $\boldsymbol{R}$ classes where each class in $\boldsymbol{R}$ is of fixed *and* equal size. In contrast $P_1$ and $P_2$ are of fixed, but not necessarily equal, size.

The overall goal in the Fixed Partitioning Automaton in the relation to the incremental crawling task is to partition the nodes into classes according to their probabilities weighted against their importance-value. We wish to partition the $N - M$ nodes with the smallest values and $M$ nodes with the highest values into different classes, and utilize the available capacity in such a way that the $M$ nodes with the highest values are polled at each time slot.

To clarify we set up a table to explaining the relation between the presented automaton and the binary knapsack.

| Class | Size | | Description |
|---|---|---|---|
| $P_1$ | $M$ | (Knapsack size) | Objects to be placed in binary knapsack |
| $P_2$ | $N - M$ | (Number of objects - M) | Objects not to be placed in binary knapsack |

Table 5.1: Classes of $\boldsymbol{R}$ in the Fixed Partitioning Automaton

### 5.2.1 Learning automaton description

The heart of our automaton solution involves, for each item, to create an abstract object to move around within the automaton. Each abstract object is located within two fixed classes ($\boldsymbol{R} = \{P_1, P_2\}$), where $P_1$ contains the items currently chosen to be in the knapsack while $P_2$ contains the items chosen to be left out of the knapsack. Each abstract objects must be either

in class $P_1$ or $P_2$. Additionally an object in $P_1$ is connected to action $\alpha_1$ while an object in $P_2$ is connected to $\alpha_2$. In our case the actions ($\alpha_1$ and $\alpha_2$) determine whether the associated nodes should be polled or not.

Each action class has a fixed number of states ($\boldsymbol{\phi} = \{\phi_{i0}..\phi_{iN}\}$) where $\phi_{i0}$ is the most internal state and $\phi_{iN}$ is the boundary state. The closer an abstract object is to the internal state, the more certain it is that the object is located in the correct class. In contrast, the closer an abstract object is to the boundary state the less certain it is that the associated class is correct. Initially each abstract object is placed in the boundary states of its class.

In difference to the Object Migration Automaton, the presented scheme does not involve distributing the access of the items into tuples of two and comparing the accessed items. In our proposed scheme each abstract object works independently as long as the abstract objects are not located in boundary states. The solution to the equi-partitioning problem requires that all object are accessed and that there exists an underlying optimal solution. Our case differs from the equi-partitioning solution as the actions connected to each class within the automaton decides when each objects should be accessed, whereas the Object Migration Automaton does not controll any of the accesses of the objects.

It is therefore essential to divide the accesses between the classes $P_1$ and $P_2$. This means that most of the capacity, used to access objects, should be focused on $P_1$ as the goal is to fill $P_1$ with the most valuable objects. However to ensure that the scheme at all times has the possibility to improve, it is necessary that some part of the available capacity is used on the items not in the $P_1$.

We approach the problem by defining a constant $f \in [0, 1]$ to help us divide the available capacity between the two classes. The size of $P_1$ and the number of objects in $P_2$ accessed at each time slot should sum to the available capacity. This means that $M + f(N - M) = C$, where $C$ is the size of the knapsack. In other words all the objects in $P_1$ should be accessed while only $f(N - M)$ of the objects in $P_2$ should be accessed all the time. The objects in $P_2$ are therefor accessed with a round robin scheme selecting $f(N - M)$ objects at each time slot.

Whenever the objects are accessed, they receive input from the environment as $\beta \in \{1, 0\}$. When an external source has been updated since last access $\beta$ will always be equal to 0, in contrast when an external source has not been updated $\beta$ will always be equal to 1.

As a value of a node is decided by both the probability of change and the defined importance-value, we must ensure that both the $\omega_i$ of the node $i$ and the $\beta$ received when object $i$ is taken into consideration in the transition function. For this to be the case we must use $\omega_i$ as part of the rewards in

the transaction function.

We use the following scheme in order achieve our goals:

**When $\beta$= 0:**

1. *If $O_i$ is in class $P_1$*
   Move $O_i$ one state closer to the most internal state with the probability $\omega_i$ if possible.
   Stay in the current state with probability $1 - \omega_i$.

2. *If $O_i$ is in class $P_2$ and not in a boundary state*
   Move $O_i$ one state closer to the boundary state with the probability $\omega_i$.
   Stay in the current state with the probability $1 - \omega_i$.

3. *If $O_i$ is in class $P_2$ and in a boundary state*
   $O_i$ should move from $P_2$ to $P_1$ with the probability $\omega_i$ and switch place with an object placed in $P_1$. The candidates to move from $P_1$ to $P_2$ is the objects located in the boundary states of $P_1$ and at the current time slot received $\beta = 1$. If no objects are in boundary states the objects closest to the boundary state are candidates. An arbitrary object from the candidates are selected, say $O_j$, and $O_i$ and $O_j$ switch places. If no objects in $P_1$ at at the current time slot received $\beta = 1$, no transition will occur.
   Stay with the probability $1 - \omega_i$

In contrast if $O_i$ receives an input of $\beta = 1$ the rules are almost reversed, however no importance-value is considered.

**When $\beta$= 1:**

1. *If $O_i$ is in class $P_2$*
   Move $O_i$ one state closer to the most internal state if possible

2. *If $O_i$ is in class $P_1$ and not in the boundary state*
   Move $O_i$ one state closer to the boundary state

3. *If $O_i$ is in class $P_1$ and in a boundary state*
   $O_i$ should move from $P_1$ to $P_2$ and switch place with an object placed in $P_2$. As in the latter rule the candidates to move from $P_1$ to $P_2$ are in the boundary state of $P_1$ and at current received $\beta = 0$ or the objects

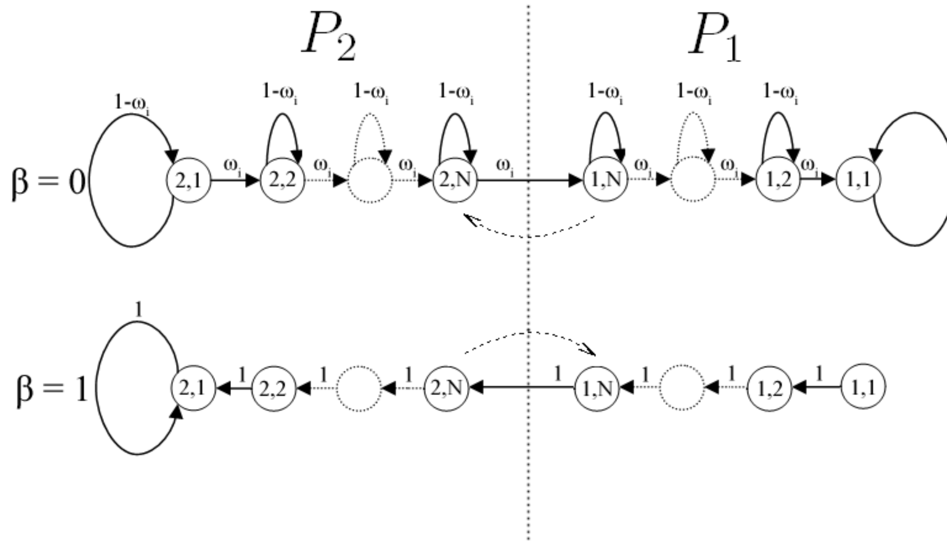


Figure 5.2: Fixed Partitioning Automaton

closest to the boundary state. An arbitrary object from the candidates is chosen and switched place with $O_i$. If no objects at the current time slot received $\beta = 0$, no transitions will occur.

Figure 5.2 shows a graphical representation of the Fixed Partitioning Automaton with the boundary states $1, N$ and $2, N$, while the most internal states being $1, 1$ and $2, 1$. Note that in order for an object to move from $P_1$ to $P_2$ or reversed, a corresponding object must move the other way.

# Chapter 6

# Prototype

To evaluate the performance of our learning algorithms we implemented a test environment that simulated the environment a crawler interacts with.

## 6.1 Requirements

We set the following requirements

- Must simulate web page update patterns so the learning algorithms can be evaluated in a realistic environment.
- Must support both fixed and non-stationary environments in order to show the properties of the proposed solutions in different settings.
- Must support visualization of the simulation results in order to get a convenient look on how the learning algorithms improve their performance over time.
- Must support a possibility to deploy different reinforcement schemes so we can easily see how the learning algorithms perform with respect to these.
- Must be able to set the number of web pages that shall be monitored so we can see how the algorithms perform when dealing with different numbers of data sources.
- Must be able to set the capacity of the crawler so we can see how the algorithms perform when they are met with the constraints of the crawler capacity.
- Must be able to serialize the simulation runs for later merging and analysis.

## 6.2 Graphical user interface

The graphical user interface and the underlying simulation environment was implemented with Python and the WxPython package. Figure 6.1 shows the application. The general options for the simulation are listed in table 6.1.

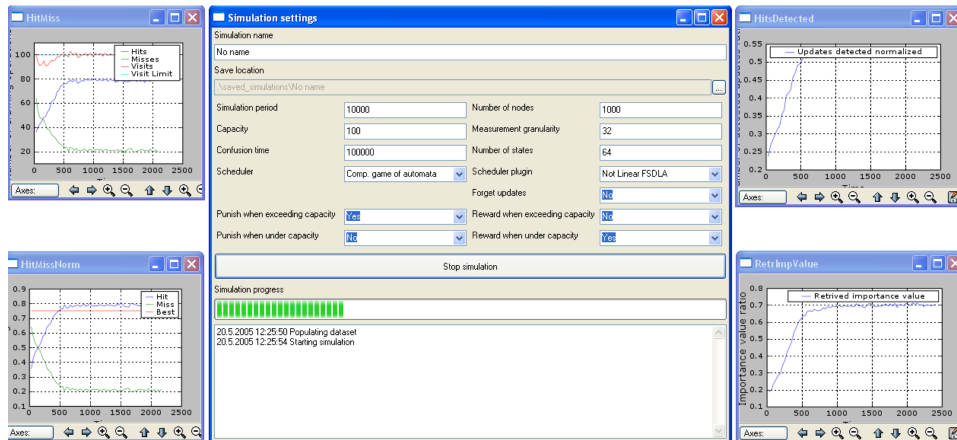


Figure 6.1: Simulation settings

| | |
|---|---|
| **Simulation name** | Name given to the simulation. |
| **Save location** | Location where simulation result will be saved. |
| **Simulation period** | Number of discrete time slots - length of simulation. |
| **Number of nodes** | Number of autonomous nodes simulating web pages. |
| **Confusion time** | Environment's penalty probabilities change after the given time has progressed. |
| **Number of states** | Number of states in the active learning automata. |
| **Scheduler** | Which scheduler to use. |
| **Scheduler plugin** | Each scheduler may have a set of available learning algorithms. These are the actual learning automata. |
| **Forget updates** | If updates disappear at once or remain until the next update occurs. |
| **"Reinforcement"** | Define when penalties and rewards shall be regarded. |

Table 6.1: Explanation of simulation settings

The application is able to produce a set number of autonomous nodes which the simulated crawler interacts with. The crawler is controlled by a scheduler that interacts with the learning algorithm. At each time slot the scheduler feeds the crawler with pages to visit based on the decision of the learning algorithm. The scheduler then forwards feedback to the learning

algorithms based on the outcome of the action inflicted. During simulations the application also visualized real time progress of several measurements.

# Chapter 7

# Simulation results and discussion

In this chapter we first present the setup and performance metric chosen for the simulations conducted. This is covered accordingly in section 7.1 and 7.2

We then go on and present the actual simulation results, combined with a discussion of the results, in section 7.3

## 7.1 Simulation setup

**R=$|D|$:** Number of monitored nodes we wish to monitor for change. This value is set to 1000 if otherwise not stated.

**C:** The capacity available each time slot, equivalent to the knapsack size. This is set to 100 for all simulations.

**Update probability distribution** As defined in section 3.2 each node has an attached probability $p_{i,j}$ which denotes the probability that an update occurs at time j. The update probability distribution is chosen according to a zipf distribution [23], as done in other work [8, 16]. We create our update probability distribution by letting $\theta$ randomly vary between 0 to 2 so we remain a skewed distribution, but get a larger part of the nodes to have a mid-range probability of update, as illustrated in figure 7.1. This is in contrast to related work [8] which used an extremely skewed distribution in their experiments. We feel that our approach yields a more realistic setting, although it gives the problem more complexity as it makes it harder to prioritize between the nodes for our learning algorithms.

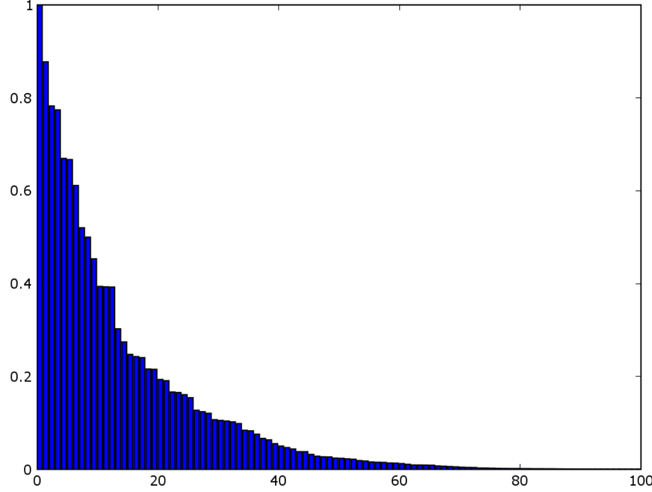


Figure 7.1: Update probability distribution of 100 nodes

**Importance-value distribution** The importance-value of a node, $\omega_i$, is chosen in relation to its probability of change in a random biased manner. This assumption is done according to the findings in [24] were more dynamic pages are shown to be more popular (indicating more importance) than other pages.

**Expected Change-capacity ratio** is the expected number of updates $\sum_{i \in P} p_{i,j}$, that occur at time instant j, over the number of monitoring tasks available, C, formally expressed as $(\frac{\sum_{d_i \in D} p_{i,j}}{C})$.

## 7.2 Performance metric

**Retrieved importance-value** denotes the proportion of the retrieved importance-value over the expected total importance-value that is produced at time instant j.

$$\frac{\sum_{d_i \in D} \omega_i y_{i,j} H(d_i, j)}{\sum_{d_i \in D} \omega_i p_{i,j}}$$

To clarify the above expression we must not only choose to poll node $d_i$ at time j ($y_{i,j} = 1$) but also get a hit ($H(d_i, j) = 1$) in order to add to the retrieved importance-value.

## 7.3 Results and discussion of the results

In this section we present the results of the simulations conducted in order to evaluate different aspects of our proposed solutions.

First we address the issue of finding good values for possible performance critical parameters in 7.3.1.

We go on and evaluate the performance of our proposed solutions in a stationary environment in comparison to alternative algorithms. Our developed learning schemes are evaluated using both of the environment models defined in section 3.2 and according to the performance metric presented in 7.2.

Finally we show how our developed learning algorithms respond to the same environment models as they turn non-stationary in section 7.3.3.

Each of the 3 different aspects addressed are discussed and summarized in their corresponding sections.

### 7.3.1 Identifying good parameters

In this section we address the issue of finding good values for possible performance critical parameters for our automata solutions presented in 5. To elaborate, we empirically wish to show how different values of given parameters affect both accuracy and speed of convergence. We have also used the discovered results as a guideline for choosing parameters for the simulations we present in section 7.3.2 and 7.3.3.

First we present the results for the Competitive Game of Learning Automata solution where we show how the resolution of the automaton, $N$, affects performance.

We then go on and see how $N$ affects the performance of the Fixed Partitioning Automaton. Here we also look at how the capacity distribution, $f$, influences performance.

#### Observations and discussion

**Parameter $N$ of the CGLA** In this section we evaluate the parameter $N$ of the automata presented in 5.1 as a solution to the fractional knapsack problem in the context of an incremental crawler.

The performance is evaluated with respect to 4 different values of $N$: 16, 32, 64 and 128. We believe this range of values will give a good indication of how varying the parameter $N$ affects the results.

Figure 7.2 show the calculated average of 10 simulation runs of each variant of parameter $N$.

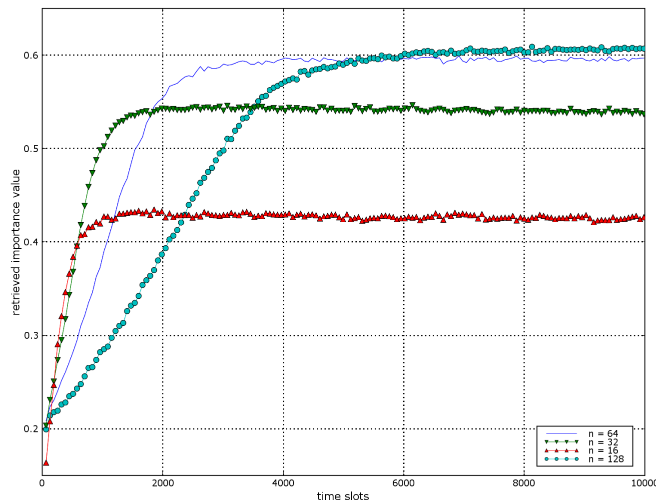


Figure 7.2: Performance with different resolutions of $N$ in the automata

The two most notably observations that can be made in figure 7.2 is that a larger value of $N$ results in better performance with respect to the retrieved importance-value. However, a larger value of $N$ also slows down the speed of convergence, i.e. it takes more time to reach the same level when $N = 64$ as when $N = 32$. In other words, figure 7.2 gives us an indication that there exists a trade-off between the accuracy of the result and the speed of convergence.

The indicated trade-off is although not very surprising. A larger value of $N$ results in a finer granularity of the fractions we can take of each item. We will therefore get a better overall performance as each automaton can get closer to an optimal value.

The different speed of convergence for different values of $N$ is caused by the fact that larger values of $N$ will result in possible longer walks in order to reach the optimal value. A large $N$ will therefore decrease convergence speed.

Both of these observations are in accordance with the observations done for the learning automaton solution of the parameter optimization problem summarized in section 4.1

By observing figure 7.2 we see that the performance rate with respect to the retrieved importance-value does not follow the same model as the convergence speed. I.e the difference in converging speed when $N = 64$ as when $N = 128$ is about 1.5, but the difference in retrieved importance-value

is very small. This observation tells us that there is little accuracy to gain by choosing a large $N$-value if convergence speed is an issue. A $N$-value much smaller than 64, i.e $N = 32$ or $N = 16$, as shown in figure 7.2 will although have a larger impact on the accuracy which cannot be as easily defended by their respectively speed of convergence.

In the context of an incremental crawler it is natural that we want to have as rapid increase in performance as possible. Convergence speed does in other words become an issue here. A feasible solution seems to be when $N = 64$ as the trade-off between good accuracy and good convergence speed is better than shown for the other values of $N$.

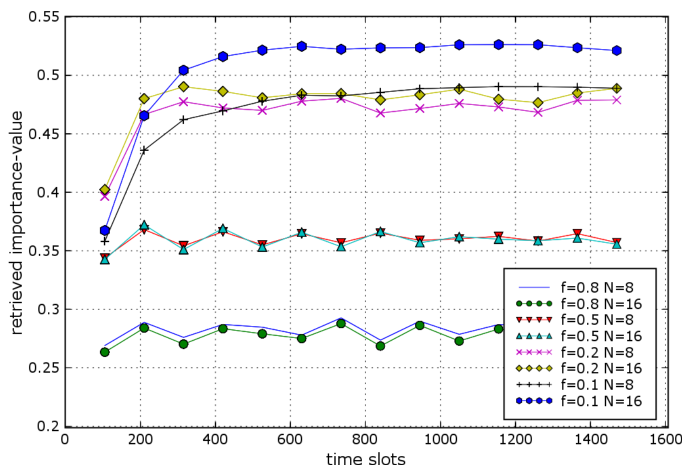


Figure 7.3: Performance when having different values of $N$ and $f$ in the Fixed Partitioning Automaton when $N$ is small

**Parameter $N$ and $f$ of the FPA** In this section we evaluate possible values of the $N$ and $f$ parameters of the automaton presented in section 5.2 as a solution to the binary knapsack problem in the context of an incremental web crawler.

As initial simulations showed that $f$ and $N$ are not mutually exclusive parameters with respect to accuracy and convergence speed, we conducted simulations where performance where shown according to 4 values of $N$ run over 4 values of $f$. $N$ was chosen from the set $\{8, 16, 32, 64\}$ and $f$ was chosen from the set $\{0.1, 0.2, 0.5, 0.8\}$. We believe this range of values will give a good indication of how varying the parameter $N$ over different values of parameter $f$ affects the results.

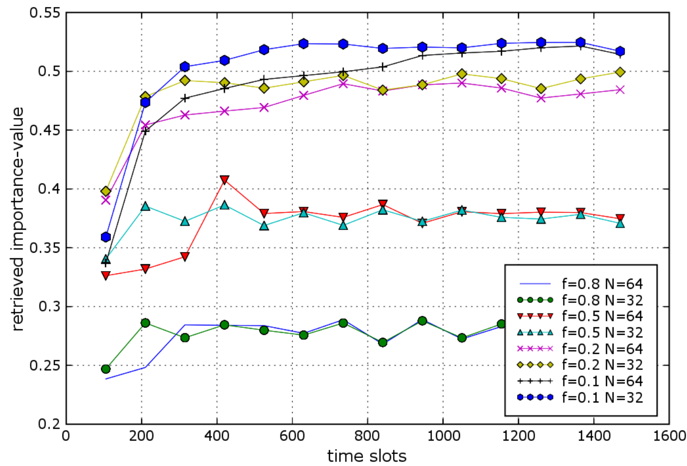


Figure 7.4: Performance when having different values of $N$ and $f$ in the Fixed Partitioning Automaton when $N$ is large

The calculated average of 100 simulation runs where $R = 100$, $C = 10$ and with an environment where updates overwrite information can be observed in figure 7.3 and 7.4.

The most notable observation is how different values of $f$ affect the performance with respect to the retrieved importance-value. As $f$ gets closer to 1 the scheme becomes more and more similar to a uniform scheme as more and more resources are granted to the round-robin queue that governs the $P_2$ partition. This is an early indication that the learning algorithm will outperform the uniform scheme when operating with accurate parameters. In contrast we get a better score of the retrieved importance-value as $f$ becomes 0.1.

As a small value of $f$ would grant less resources to the round-robin queue, used in the learning mechanism to move nodes from the $P_2$ partition, to the $P_1$ partition we would initially expect that the convergence speed would be slower. However, we observe that there is little trade-off between convergence speed when $f = 0.1$ and $f = 0.2$, we only achieve a greater accuracy. The trade-off is however likely to increase as $f$ verges toward 0 as we then would spend less and less resources to sort out the nodes that should belong in the active partition $P_1$. We would therefore get a decline in convergence speed.

We assume $f = 0.1$ to be a good setting as we with this setting seem to score well with respect to both convergence speed and accuracy.

We have so far not addressed the $N$-parameter. As observed in figure 7.3 and 7.4 $N$ seem to contribute in a very limited way to accuracy. However, as

$N$ grows large it contribute less and less. I.e it seems we could just as well have chosen $N = 16$ than $N = 64$ in the case where $f = 0.2$. Although, if we decrease $N$ too much it seems that accuracy also decreases, i.e when $N = 16$ and $N = 0.1$ vs. $N = 8$ and $N = 0.1$.

$N = 16$ seems therefore like a suitable parameter for the FPA.

A decline in performance would although be expected as N gets close to 1 as the exchange between the two partitions would be of a more random manner as the items in the active $P_1$ partition would have no form of ranking since many of them would reside in the same states.

#### Summary

We wanted empirically to show how different values of given parameters affect both accuracy and speed of convergence for our proposed automata solutions. We also wanted to use the discovered results as a guideline for choosing parameters for the simulations we present in section 7.3.2 and 7.3.3.

The only parameter to alter in the Competitive Game of Learning Automata solution was the parameter $N$ which controlled each automaton's resolution. This parameter were shown to affect both convergence speed and accuracy of the solution.

$N = 64$ were shown to have the smallest trade-off between convergence speed and accuracy compared to the other values that where considered.

We demonstrated the performance of the Fixed Partitioning Automata by varying both its $N$ and $f$ parameter with respect to each other. $N$ gives the number of states in each partition while $f$ controls the capacity assigned to each partition.

$f$ had a clear impact on the retrieved importance-value. As $f$ approached 1 and came closer to a uniform crawling scheme the performance degraded severely.

$f$ had also some impact on the convergence speed which most likely get larger as $f$ verge toward zero. The experiments does however not show this explicitly. A reasonable selection of $f$ seemed to be when $f = 0.1$

$N$ had only a small effect on accuracy where a suitable value seems to be $N = 16$ as a lower value would degrade accuracy and a large value would not contribute much to accuracy at all.

### 7.3.2 Comparison to alternative algorithms.

In this section we compare the effectiveness of the proposed solutions to other relevant algorithms when run under the same simulation setups. These policies are:

**Uniform** Resources are distributed uniformly among the nodes we wish to monitor. This results in a round-robin scheme for the re-crawling process.

**Weighted proportional** Resources are allocated in a proportional manner to the $p_{i,j}$ and $\omega_i$ values. This algorithm assumes that the $p_{i,j}$ and $\omega_i$ values are known.

The experiments show the retrieved imoportance-value when the expected change-capacity ratio changes from 1.5 to 4. The expected change-capacity ratio of 1.5 results in having 150% updates over the available capacity, while the expected change-capacity ratio of 4 results in having 400% updates over the available capacity.

We choose $N = 16$ and $f = 0.1$ for the Fixed Partitioning Automaton and $N = 64$ for the Competitive Game Of Learning Automata solution as decided in section 7.3.1.

We first address the performance of the schemes in the environment model where changes overwrite information. Then we do the same for the environment model where updated information disappear at once. Finally we discuss and summarize the results.

### Changes overwrite information

Figure 7.5 shows the effectiveness of the proposed schemes as the expected change-capacity ratio increases in an environment where changes overwrite information.

From figure 7.5 we observe:

1. The retrieved importance-value decreases as the expected change-ratio increases, as a natural consequence of the increase in number of updates.

2. Both proposed solutions perform better than the uniform scheduler. This shows that both schemes are able to adapt to better solutions than uniformly distributing the capacity among the monitoring nodes, retrieving more importance-value.

3. The Competitive Game of Learning Automata always performs better than the weighted proportional scheme in this experiment, while the Fixed Partitioning Automaton only performs better with larger values of expected change-ratio.

4. The Competitive Game of Learning Automata always retrieves more information than the Fixed Partitioning Automaton in this experiment.

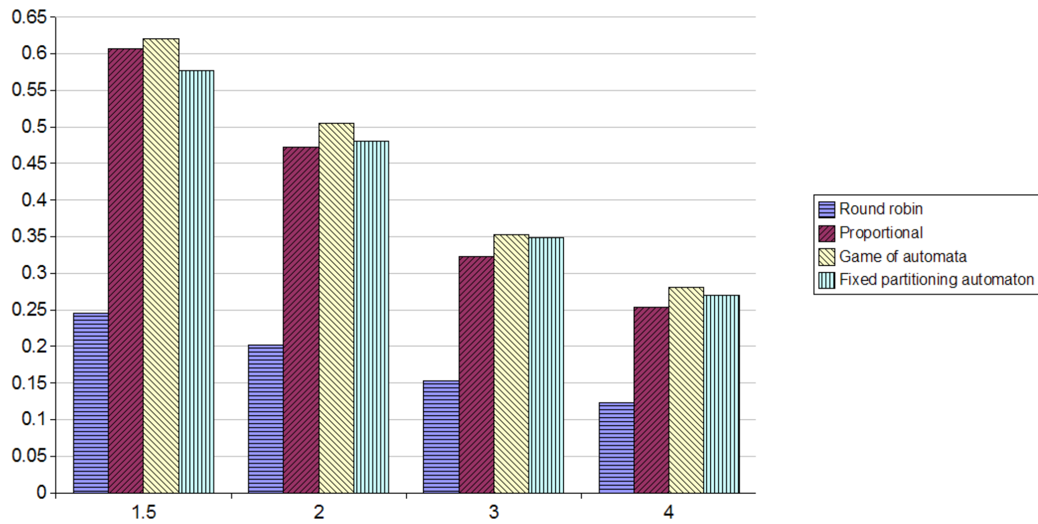


Figure 7.5: Retrieved importance-value when changes overwrite information

### Updated information disappear at once

Figure 7.6 shows the effectiveness of the proposed schemes in an environment where each update disappear at once.

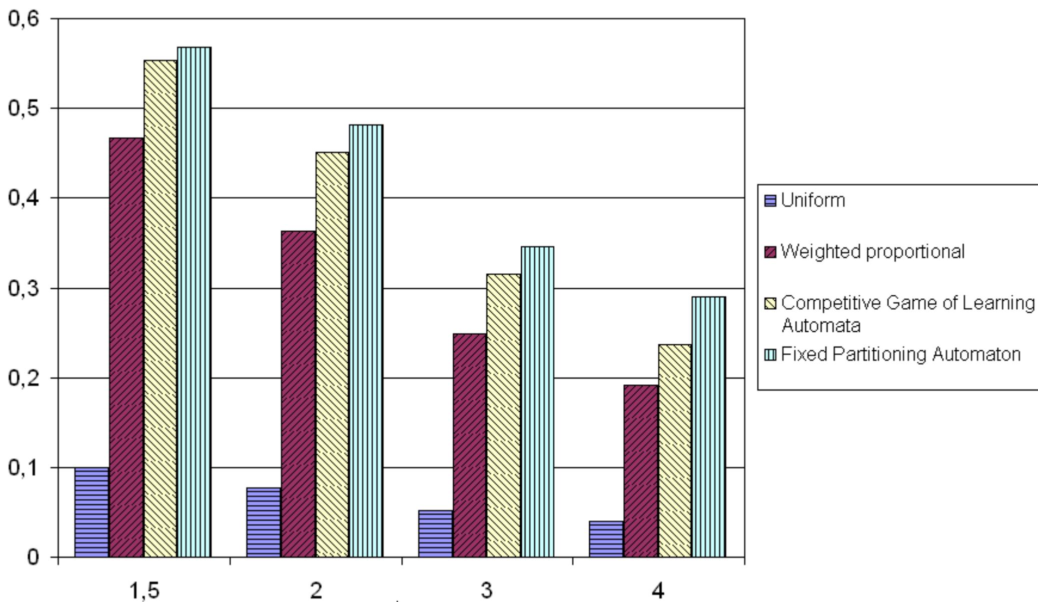


Figure 7.6: Retrieved importance-value when updated information disappear at once

From this experiment we observe:

1. The retrieved importance-value decreases as the expected change-ratio increases, as a natural consequence of the increase in number of updates.
2. Both presented schemes perform better than the uniform scheduler for any expected change-ratio.
3. Both presented schemes are able to retrieve more information than the weighted proportional scheme for any expected change-ratio whenever each updated information disappear at once.
4. In this environment model the Fixed Partitioning Automaton always performs better than the Competitive Game of Learning Automata.

### Discussion

In this section we discuss and evaluate the experiments presented in figure 7.5 and 7.6, comparing both our proposed solutions to other relevant algorithms and to each other.

**CGLA vs. Uniform** As is observable in both the latter experiments, the Competitive Game of Learning Automata performs better than the uniform scheduler. This shows that it is in our experiments always more beneficial, when it comes to the retrieved importance-value, to adapt to fractions of each item based on rewards / punishments and importance-value than to uniformly distribute the capacity among the nodes.

This indicates that the CGLA always will increase the fractions of an item $i$ if item $i$ seems beneficial to poll, as it receives $\beta = 0$. The competitive game is in our experiments shown to prioritizes items which contributes to an increase in the retrieved importance-value over items which do not.

**CGLA vs. Weighted proportional** The graphs also show that the Competitive Game of Learning Automata performs better than the weighted proportional scheduler in both environments.

A possible reason for this is that CGLA might a more aggressive scheme, which implies that it adapts to fractions of items slightly higher than weighted proportional. Slightly higher fractions would lead to slightly higher polling rates which could contribute to an over all higher retrieved importance-value when the automata prioritizes the items with the highest importance-value.

If this is the case, it would be amplified when updated information disappear at once, as each change is not detectable after the time slot it appears and no scheme can benefit from polling an item too late.

**FPA vs. Uniform** It is observable that the Fixed Partitioning Automaton in both of the latter experiments perform better than the uniform scheduler.

This shows that in our experiments it is more beneficial to focus the capacity on the items believed to have the highest values than to uniformly distribute all the polls. This is a good indication that the vastly prioritizing of items believed to have the highest values, as done by the Fixed Partitioning Automaton, results in a better performance than uniformly distribution of all polls.

**FPA vs. Weighted proportional** Our experiments show that the Fixed Partitioning Automaton has a better performance than the weighted proportional scheduler whenever updates disappear at once. Additionally it is observable in the experiments that the scheme has a better performance when updates overwrite information with large values of the expected change-ratio.

This indicates that whenever updates overwrite information the Fixed Partitioning Automaton will gain from an increase of changes as the $P_2$ class will detect more changes. This naturally leads a believe that whenever $\sum p_{i,j}$ is small, the Fixed Partitioning Automaton will detect few changes from the nodes in $P_2$ while when $\sum p_{i,j}$ is large, the scheme will detect many changes from the nodes in $P_2$.

The weighted proportional scheme is in our experiments shown to gain less than the Fixed Partitioning Automaton. A likely reason for this is that it mostly focuses on the nodes with the best values. This implies that the Fixed Partitioning Automaton scores better than weighted proportional whenever the expected change-ratio is large, and is less effective than the weighted proportional when the expected change-ratio is small.

**FPA vs. CGLA when changes overwrite information** By comparing the Fixed Partitioning Automaton and the Competitive Game of Learning Automata, the experiments show that the CGLA will perform better in an environment where changes overwrite information. This can be observed in graph 7.5 where the Fixed Partitioning Automaton retrieves less importance-value than the Competitive Game of Learning Automata.

This indicates that CGLA benefits more from distributing the nodes into fractions and polling the nodes accordingly, than the FPA does from distributing the nodes into two classes an polling accordingly. In other words, when changes overwrite information, the granularity of the polling rates of the CGLA seems to be more beneficial than partitioning into only two classes.

As the expected change-ratio increases, the gap in retrieved information-value between the Competitive Game of Learning Automata and the Fixed

Partitioning Automaton decreases. This is likely do to fact that more updates are detected in the non-prioritized $P_2$ class of the Fixed Partitioning Automaton.

This implies that it is less beneficial to distribute the polling rates as more updates occur. However, as long as updates overwrite information, it is always more beneficial than not doing so.

**FPA vs. CGLA when updates disappears at once** Figure 7.6 shows that the FPA performs better than the CGLA whenever the information in the environment disappears at once.

The results implies that it is most beneficial to focus the majority of the capacity on polling only a few nodes, not distributing the polls among N possible polling rates as done by the CGLA.

A likely reason for this is that the items with the highest probabilities of being changed at time slot $j$ for any $j$, is in this environment always the items with the over all highest probability of being changed. As the goal of the Fixed Partitioning Automaton is to distinguish the items with the over all highest values (therefore highest probability of update and importance-value), it is shown to perform better than CGLA when the updated information disappear at once.

### Summary

Both of our proposed schemes were shown to outdo the uniform scheme and in most cases the weighted proportional scheme which had the advantage of knowing the update probabilities of the monitored data sources. Most noteworthy we outperformed the scheme scheme by factors up to 550% in certain situations.

Additionally the experiments show that the Fixed Partitioning Automaton and the Competitive Game of Learning Automata perform best in their alternate environments. The Fixed Partitioning Automaton manages to score a better retrieved importance-value whenever each updated information disappears at once, as it in this environment is profitable to mainly poll the items with the highest values.

In contrast, when the changes overwrite information, the Competitive Game of Learning Automata is shown to have an over all better performance. This indicates that it in this environment is profitable to distribute the polls among the items and poll the items with close to the same rate as they change.

### 7.3.3 Adaptability

In this section we expose the Competitive Game of Learning Automata solution and the Fixed Partitioning Automaton to an environment where the nodes switch importance-value and update probabilities every 2500th time slot, making the environment non-stationary. Both of the environment models defined in 3.2 have been considered.

We choose $N = 16$ and $f = 0.1$ for the Fixed Partitioning Automaton and $N = 64$ for the Competitive Game Of Learning Automata solution as decided in section 7.3.1.

The goal of these experiments is to empirically show the adaptability of our proposed schemes and see how they respond to a non-stationary environment.

The actual switch which makes the environment non-stationary is done in the following way: *node 1 switches update probability and importance-value with node n, node 2 switches the same values with node n-1, and so on.*

Figure 7.7 and 7.8 show an average of 20 experiments conducted for each of the environment models.

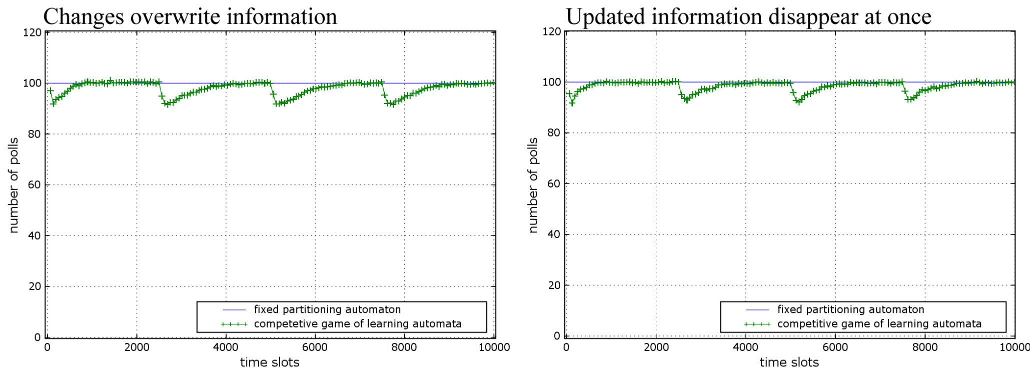


Figure 7.7: Number of polls

#### Observations and discussion

**The crawler capacity** dictates the total number of nodes that can be polled each time slot. With the The Fixed Partitioning Automata we obey this rule by choosing $f$ so $M + f(N - M) = C$. Thus, the number of visits each time slot becomes fixed, as seen in figure 7.7.

The Competitive Game of Learning Automata solution does however take a different approach. As we can observe in figure 7.7 this scheme does not utilize the capacity all the time. This is a consequence of the competitive

nature of the scheme where one goal is to adapt to a number of polls that equals the capacity. Initially we can observe that available resources is not fully utilized as the total number of polls is below the capacity. A similar situation occurs when we see a shift in the environment and most of the automata will be in a situation where their believed fraction is not correct. For an automata this is only observable through the feedback it receives from the environment, and from figure 7.7 it seems like most of the automata after a shift receives an increased level of punishment and thereby decrease their believed fraction. This causes the total number of polls to decrease as well. However, since the total capacity is no longer utilized we can observe that the automaton re-adapt until they once again reach the crawler capacity and on average remains loyal to this constraint. This behavior show that by governing the situations where the automata are allowed to either increase or decrease their believed fraction we can control the average total fraction amount so $\sum x_i < C$.

**The retrieved importance-value** is the performance metric chosen in order to show the performance of the automaton solution. Figure 7.8 show the development of this value in a non-stationary environment for both of the environment models defined in 3.2

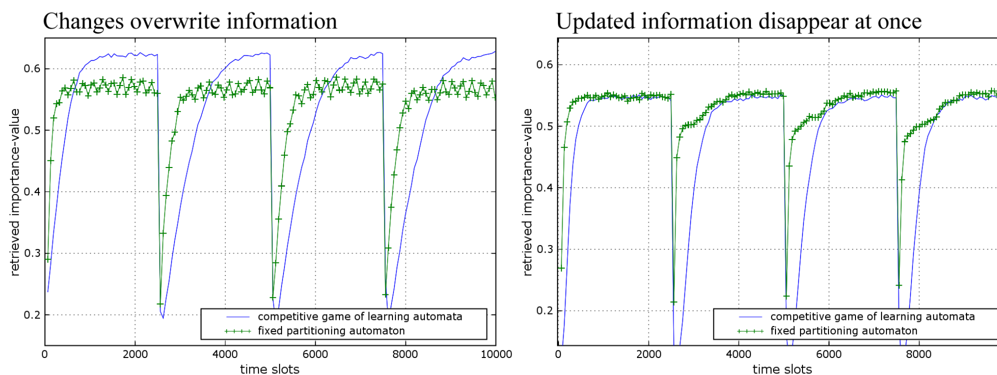


Figure 7.8: Non-stationary environment

We observe that both the Competitive Game of Learning Automata and the Fixed Partitioning Automata improve their performance as time pass until they converge to a certain value. The reached values are in accordance to the values found in section 7.3.2, but we will not discuss these any further as we in this section only will evaluate the adaptive behavior.

After each shift the retrieved importance-value decreases for both schemes as the item values of the nodes change and the currently chosen polling rates become unfeasible. As the expected total importance-values remain the same after a shift, we can assume that the automata solution would adapt to the same level as before the shift occured. Figure 7.8 show both automata solutions reaching the same level, as before the shift, in both environment models and thereby demonstrates the assumed adaptive capability.

In general it can be observed that the Fixed Partitioning Automaton adapts quicker than the Competitive Game Of Learning Automata solution. A possible explanation is that, after a shift, the CGLA does not utilize the available resources 100% as it uses some time to re-adapt to the capacity limit. A more contributing factor would although most likely be that the FPA more quickly can separate the best candidates from the worst as the nodes in the active partition $P_1$ is polled every time slot. Neither does a long walk from the internal state to the boundry state have to take place before an exchange between the $P_1$ and $P_2$ partitions happen as the candidates are picked from the state closest to the boundry state, as described in section 5.2. However, after a the first primary sorting is conducted it becomes harder to separate the nodes from eachother. This is observed in the environment model where changes overwrite information as well as in the model where informations disappears at once. In the prior environment we see this tendency as a rather fluctuating retrieved importance-value. This is caused by an continious exchange between the partitions where the nodes that are difficult to separate are involved. We do however not see this fluctation in the latter environment, but we can observe that it is still hard to separate some nodes due to the slow increase in performance after the automaton has done the initial primary sorting. This environment does however give a bit more precise feedback than the prior as we never can detect an update that happened in the previous time slot which may confuse the automata. Fluctation is therefore not as visible here.

The difference in convergence speed must although also be seen in relation to the behavior of the Competitive Game of Learning Automata. First of all, a long walk is needed in order to re-adapt to fractions that lay far apart. If the chosen fraction is $\frac{1}{N}$, it will in average be polled each $N$ time slot. If the fraction amount optimally should increase, as the automaton receives a reward, it will in average be polled after $\frac{2}{N}$ time slots, then after $\frac{3}{N}$ time slots and so on. If the most feasible fraction is 1 the automata uses in average $\frac{1}{N} + \frac{2}{N} + \frac{3}{N} + ... + \frac{N-1}{N} + 1$ time slots to reach the desirable fraction amount. Note that we in this example assume that only rewards are handed out, an optimal situation in this case.

Secondly we can see it as a direct result of the competitive game where rewards are ignored when the capacity is exceeded and penalties are ignored when we the capacity is not exceeded. As a consequence a lot of possible "learning" is missed which naturally slows the adaptive behavior down.

### Summary

It has been shown that both the proposed automata solution, the CGLA and FPA, are capable of improving their performance over time due to a learning process and that they are capable of operating in a non-stationary environment.

The Fixed Partitioning Automata has allthough shown more rapid adaptive behavior in comparison to the CGLA. This is somewhat caused by the nature of the automata itself as well as the nature of the Competitive Game of Learning Automata solution.

We have also shown that the Competitive Game of Learning Automata solution is loyal to the constraint given by the crawler capacity as it in average stays at or below this limit. This is a consequence of the competitive game where we govern when too give rewards and when too give penalties.

# Chapter 8

# Conclusion and further work

## 8.1 Conclusion

In this thesis we have developed and evaluated two new solutions to the incremental crawling task when monitoring highly dynamic data sources. This is an issue that has not been addressed much in literature and our proposed solutions are a contribution in order to replace traditional crawling schemes that has been shown to be sub-optimal in previous studies.

Our novel approach looked at the incremental crawling task as a continuous learning problem where scheduling of monitoring tasks were combined with parameter estimation in an on-line manner. We also mapped the problem to two variants of the so called knapsack problem and based our solutions on a machine learning technique known as learning automata.

The two variants of the knapsack problem addressed in the context of the incremental crawling task were the binary knapsack problem and the fractional knapsack problem. In both variants we considered the item values to be of unknown distribution and the item weights to be equal.

As a solution to the binary knapsack variant we presented a learning automaton which we named the Fixed Partitioning Automaton. This algorithm is an extension on the Object Migration Automaton previously presented as a solution to the equi-partitioning problem. Our proposed solution is designed to partition the items into two partitions of fixed, but possibly unequal, sizes where one partition contains the most valuable items. Although this functionality has not been formally proved in this thesis, simulations show that the scheme performs very well in an environment where this fixed partitioning strategy is optimal.

The fractional knapsack variant were approached by extending and connecting a learning algorithm used to solve the parameter optimization prob-

lem in a competitive game. This solution were given the name Competitive Game of Learning Automata. By connecting independent automata into a competitive game and govern when rewards and penalties should be regarded, we designed a scheme that were not only able to adapt to a set knapsack / crawler capacity size, but also create a stochastic competition between the automata that adaptively improved performance. The result was a distribution of available capacity / item fractions that in most cases, not regarding the Fixed Partitioning Automaton, outperformed compared strategies.

The evaluation of the performance of our proposed automata solutions were done by using two different environment models; one where we considered updates to disappear at once and another where we considered updates to remain, but just until the next update occurred. The Fixed Partitioning Automata showed greater performance in the prior environment and the Competitive Game of Learning Automata performed best in the latter.

Both of our proposed solutions were shown to outdo the uniform scheme and in most cases the weighted proportional scheme which had the advantage of knowing the update probabilities of the monitored data sources. Most notably we outperformed the round robin scheme by factors up to 550% in certain situations. This confirms the findings in previous studies where traditional crawling schemes were shown to be sub optimal when working with highly dynamic environments. It also confirms the capabilities of our proposed schemes.

Our proposed solutions were also shown to successfully being able to operate in non-stationary environments where the Fixed Structure Automata showed a faster adaptive behavior compared to the Competetive Game of Learning Automata in all investigated situations. This is mainly due to the Fixed Structure Automata nature as more it quickly can separate good candidates from bad candidates as it continuously focus it's efforts on only a small subset of considered items and can thus swiftly do a primary sorting and migration between the partitions if a shift in the environment happens.

## 8.2 Further work

The beauty of research is reflected in it's ability to contribute new thoughts and approaches to the world which other people may use as a basis for new ideas and/or in relation to their existing work. We will therefore in this section briefly introduce a few problems this thesis could be a subject for.

- As of now, the Competetive Game of Learning Automata-solution only consider the current time-slot when it makes the decision to crawl a

page or not. It would be interesting to see if it is possible to design a similar scheme where the automata look further ahead in order to make the same decision and to see whether this affects performance either way.

- We have only focused our work toward two environment models and its possible to expand our work by looking at other relevant models, i.e. a model where changes are cumulatively added.
- We propose a solution to the binary knapsack problem where we consider the item values to be of unknown distribution and the items having equal weight. What if we would consider a situation where the weight is not equal. How could the proposed scheme be expanded/redesigned to deal with this new problem?
- The proposed solutions are only tested using syntethic data sets. It would be interesting to see how the learning algorithms would perform using a real-world data.
- To prove the convergence results that are demonstrated empirically in this thesis are subject for further work.
- Improvement of the suggested schemes by introducing new techniques and algorithms in order to improve speed and accuracy is always valuable.

# Bibliography


[1] J. Cho and H. Garcia-Molina, "Synchronizing a database to improve freshness," 2000, pp. 117–128. [Online]. Available: citeseer.ist.psu.edu/cho00synchronizing.html

[2] ——, "The evolution of the web and implications for an incremental crawler," in *Proceedings of the Twenty-sixth International Conference on Very Large Databases*, 2000. [Online]. Available: citeseer.ist.psu.edu/cho00evolution.html

[3] J. Edwards, K. S. McCurley, and J. A. Tomlin, "An adaptive model for optimizing performance of an incremental web crawler," in *World Wide Web*, 2001, pp. 106–113. [Online]. Available: citeseer.ist.psu.edu/edwards01adaptive.html

[4] J. Cho and A. Ntoulas, "Effective change detection using sampling," 2002. [Online]. Available: citeseer.ist.psu.edu/cho02effective.html

[5] S. K. G. Hadrien Bullot and M. K. Mohania, "A data-mining approach for optimizing performance of an incremental crawler," Swiss Federal Institute of Technology, Lausanne - Indian Institute of Technology, Delhi - IBM India Research Lab, Tech. Rep.

[6] C. M. Grenville, "Bandwidth efficient web object change interval estimation." [Online]. Available: citeseer.ist.psu.edu/666170.html

[7] M. K. Bergman, "The deep web: Surfacing hidden value." [Online]. Available: citeseer.ist.psu.edu/bergman00deep.html

[8] S. Pandey, K. Ramamritham, and S. Chakrabarti, "Monitoring the dynamic web to respond to continuous queries," in *WWW '03: Proceedings of the twelfth international conference on World Wide Web.* New York, NY, USA: ACM Press, 2003, pp. 659–668.

[9] W. I. Sources, "Wic: A general-purpose algorithm for monitoring." [Online]. Available: citeseer.ist.psu.edu/712654.html

[10] M. A. L. Thathachar and P. S. Sastry, "Varieties of learning automata: an overview." *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 32, no. 6, pp. 711–722, 2002.

[11] S. R. K. et al., *The American Heritage Stedman's Medical Dictionary, SECOND EDITION.* H.M.

[12] K. S. Narendra and M. A. L. Thathachar, *Learning automata: an introduction.* Prentice-Hall, Inc., 1989.

[13] V. N. Padmanabhan and L. Qui, "The content and access dynamics of a busy web site: findings and implicatins," in *SIGCOMM*, 2000, pp. 111–123. [Online]. Available: citeseer.ist.psu.edu/padmanabhan00content.html

[14] B. E. Brewington and G. Cybenko, "Keeping up with the changing Web," *Computer*, vol. 33, no. 5, pp. 52–58, 2000. [Online]. Available: citeseer.ist.psu.edu/brewington00keeping.html

[15] D. Fetterly, M. Manasse, M. Najork, and J. L. Wiener, "A large-scale study of the evolution of web pages," *Softw. Pract. Exper.*, vol. 34, no. 2, pp. 213–237, 2004.

[16] J. L. Wolf, M. S. Squillante, P. S. Yu, J. Sethuraman, and L. Ozsen, "Optimal crawling strategies for web search engines," in *WWW '02: Proceedings of the eleventh international conference on World Wide Web.* New York, NY, USA: ACM Press, 2002, pp. 136–147.

[17] J. Cho and H. Garcia-Molina, "Effective page refresh policies for web crawlers," *ACM Trans. Database Syst.*, vol. 28, no. 4, pp. 390–426, 2003.

[18] ——, "Estimating frequency of change," in *Submitted for publication*, 2000. [Online]. Available: citeseer.ist.psu.edu/cho00estimating.html

[19] B. J. Oommen, "Stochastic searching on the line and its applications to parameter learning in nonlinear optimization." *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 27, no. 4, pp. 733–739, 1997.

[20] B. J. Oommen and D. C. Y. Ma, "Deterministic learning automata solutions to the equipartitioning problem," *IEEE Trans. Comput.*, vol. 37, no. 1, pp. 2–13, 1988.

[21] ——, "Fast object partitioning using stochastic learning automata," in *Proceedings of the Tenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, New Orleans, Louisiana, USA, June 3-5, 1987*. ACM, 1987, pp. 111–122.

[22] W. Gale, S. Das, and C. T. Yu, "Improvements to an algorithm for equipartitioning," *IEEE Trans. Comput.*, vol. 39, no. 5, pp. 706–710, 1990.

[23] [Online]. Available: http://www.cs.unc.edu/˜vivek/home/stenopedia/zipf/

[24] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. R. Karlin, and H. M. Levy, "On the scale and performance of cooperative web proxy caching," in *Symposium on Operating Systems Principles*, 1999, pp. 16–31. [Online]. Available: citeseer.ist.psu.edu/article/wolman99scale.html