



Solving the bin packing problem

by

Anis Yazidi

Supervisor: Morten Goodwin Olsen

Project report for IKT-407 in Autumn 2006

Agder University College
Faculty of Engineering and Science

Grimstad, 26 November 2006

Status: Final

Keywords: Distributed crawler, Learning automata, Bin packing problem, Stochastic environment, Knapsack problem.

Abstract:

Through the last years, the Internet has been growing at a high pace raising new challenges to web crawlers. Various approaches have been developed aiming at refreshing efficiently the data repository. Traditional approaches where pages are downloaded at regular intervals are shown to be suboptimal. To overcome the limitations of traditional periodic crawlers, incremental crawlers come to stage.

However, most emphasis has been given to the deployment centralized incremental crawlers neglecting the distributed version of the problem. Obviously, as the size of the internet grows, it becomes imperative to tackle such a version of the problem.

From this perspective, our work attempts to fill the shortage in this field.

In this paper, we make use of the concept of incremental distributed crawler to monitor dynamic web data. Our approach underlies on discrete learning automata to keep track of the web pages polling frequencies.

A fascinating aspect of our scheme is mapping the problem to the Bin packing problem which, to the best of our knowledge, has not been mapped to web crawling previously.

We believe that our scheme can be applied to handle a range of other resource allocation problems where the weights of items are dynamically changing through the time.

[This work is licensed under the Creative Commons Attribution-ShareAlike License \(http://creativecommons.org/licenses/by-sa/2.5/\).](http://creativecommons.org/licenses/by-sa/2.5/)

Table of Contents

1	Introduction.....	3
2	Problem description.....	4
3	Background.....	5
3.1	Web Crawling Policies.....	5
3.2	A variant of the Knapsack problem: Subset Sum Problem.....	5
3.3	Bin packing problem.....	6
3.4	Learning automata.....	6
4	Solution.....	8
4.1	Requirements.....	8
4.2	Design Specification.....	8
4.3	Implementation.....	12
4.4	Validation and Testing.....	15
4.3.1	Static environment:.....	15
4.3.2	Dynamic environment:.....	22
5	Discussion.....	22
6	Further work.....	23
7	Conclusion.....	24
	References.....	25

1 Introduction

Crawlers are core elements of many services running on the Internet. In fact, by keeping a local copy of the web data, they facilitate the search of relevant information.

One of the key challenges of a web crawler is to maintain its repository up to date. However, due to limitations of bandwidth and computer resources, this task is not feasible in a timely manner. Therefore the crawler should decide carefully which pages to download at what time. Indeed, downloading an unchanged page leads to a waste of valuable crawling resources. Furthermore, the dynamicity of the web put additional burden on the crawling process. In fact, the web pages are often updated with different periods, which makes extremely challenging to choose the right time to download.

In [13], Narendra and Thathachar presented the learning automaton as a solution to finding the optimal action in a random environment. Since this time, learning automata has gained a significant attention. Learning automata seems promising for our project because it was shown highly effective when operating in dynamic environments.

In this paper, we consider the problem of optimizing the allocation of distributed web crawler resources when the polling capacity is limited. Our proposed paradigm underlies on mapping the problem to a novel version of the Bin packing problem where the characteristics of items are unknown. This assumption seems to be realistic when operating in dynamic environments such as the Web. Our scheme involves connecting every item with a team of learning automata performing a random controlled walk on a discredited solution space.

To the best of our knowledge, this problem has not been addressed before in the literature.

In the field of distributed crawlers, most of the work [11, 12] in the area mainly focused on the architecture design rather than specific resource allocation approaches. In [10], Cho and Garcia-Molina investigated policies to assign URLs to crawlers in a manner that avoids repeated downloads of the same page.

A similar resource allocation approach to our work has been studied in [8]. The authors of this publication modelled the problem of resource allocation in web as a knapsack problem with unknown properties of the materials. However, they addressed only the centralized version of the problem and their study did not embrace the distributed version. In this project we join the claim of [8, 9] stating that learning automata can efficiently estimate the update frequencies of web pages.

The rest of the report is organized as follows.

In section 2: we present the problem description of our project.

In section 3: we survey a necessary background for the understanding of the project.

In section 4: a detailed description of the solution, its implementation and the conducted experiments are presented.

In section 5: we discuss and evaluate the proposed solution. A further work is also considered in this section.

In section 6: we briefly draw conclusions from our project.

2 Problem description

In this project, we address the problem of optimizing the allocation of distributed web crawler resources when the polling capacity is limited.

The problem seems captivating and intriguing since a possible solution to it can provide benefits to a wide range of resource allocation applications.

Our work is mainly motivated by the need of a distributed crawling policy to manage large scale web data. In fact, the problem has arisen remarkably these last years with the dramatic growth of the internet. However, few studies have been performed in the area of distributed crawlers.

The proposed approach centres on mapping the problem to a novel version of the Bin packing problem where the characteristics of items are unknown. We should underline that the items here are the web sites to monitor, and their weights are guesses of the amount of updated pages. In order to estimate the appropriate polling frequency of the web pages we make use of learning automata.

3 Background

3.1 Web Crawling Policies

A survey of the literature discerns two types of web crawlers namely batch and incremental crawlers. A batch crawler polls the monitored web pages with the same period regardless of how often they change [1, 2, 3]. This blind scheme is shown to be sub optimal. In fact, the proposed scheme wastes the monitoring capacity by downloading unchanged data. More work was invested in developing more efficient crawling policies leading to the advent of incremental crawlers. In contrast to a batch crawler, an incremental crawler updates its repository based on the frequencies at which the pages change. In regards to performance, incremental crawlers attain higher freshness of the local copy of the monitored data sources and optimize significantly the use of the available crawling resources.

The notion of incremental crawler was first introduced by Junghoo Cho and Hector Garcia in [4]. Their claim is that the web pages updates can be modelled as Poisson processes. Furthermore, in [5] they build estimators for the web pages changes underlying on this assumption. However such estimation of web pages changes requires having a history of the updates which is not practical in many cases. In addition, in [5] it is stated that their Poisson model does not handle a large set of web pages, namely highly and slowly changing web pages. Moreover, they assumed that the change frequency is static which may not be valid in certain cases.

The reliability of the Poisson model has been the subject of critical analysis conducted by Brewington and Cybernenko in [6]. In fact, they showed that a large amount of tested web pages does not follow the Poisson Process.

J. Cho and A. Ntoulas proposed a sampling-based policy in [7] for the incremental crawler. The approach relies on estimating the number of web pages updated in a web site based on using random samples. Hence, this approach draw conclusions exclusively from the samples available in the current cycle and does not make use of the update pages history, although it represents a relevant information susceptible of yielding better accuracy.

Recent studies [8, 9] have regarded incremental crawling as a continuous learning process. They mapped the problem of optimizing the frequencies of revisiting the monitored web pages under restricted capacity constraints to a version of knapsack problem where the volume of items are variable, following unknown distribution. These works [8, 9] showed that learning automata seem particularly promising when applied in dynamic environments such as the Web.

3.2 A variant of the Knapsack problem: Subset Sum Problem

The knapsack problem is a classical NP hard problem. The knapsack problem can be described as following: a thief breaks into a museum carrying a knapsack with limited capacity. He found n items of various size and value. How should he choose which items to steal in order to maximize his haul without exceeding the capacity of his knapsack?

From the literature, we can distinguish two variants of the Knapsack problem, namely the 0-1 and the fractional knapsack problems. In the 0-1 knapsack problem each item must be taken or left in entirety. Whereas, in the fractional knapsack problem the thief can take fractions of items.

The Subset Sum Problem is defined formally in [14] as given a set of n items and a knapsack, with

$$w_j = \text{weight of item } j$$

c = capacity of the Knapsack

Select a subset of the items whose total weight is close to, without exceeding, c .
i.e

$$\text{Maximize } z = \sum_{j=1}^n w_j x_j$$

$$\text{Subject to } \sum_{j=1}^n w_j x_j \leq c$$

$$x_j = 0 \text{ or } 1, \quad 1 \leq j \leq n$$

Where

$$x_j = \begin{cases} 1 & \text{if item } j \text{ is selected,} \\ 0 & \text{otherwise} \end{cases}$$

The Subset Sum problem can be solved either heuristically or exactly.

3.3 Bin packing problem

The bin packing problem is one of the famous NP hard problems that arises in a variety of practical problems. The bin packing problem has been described as a set of items of different volume must be packed into a finite number of bins in a way that minimizes the number of used bins [20].

Numerous heuristic solutions have been proposed to the bin packing problems. [21] presents various algorithms to solve the problem in a near optimal way. Empirical and analytical results showed the superiority of the Best Fit Decreasing compared to other packing algorithm.

Because of this, we make use of the Best Fit Decreasing algorithm.

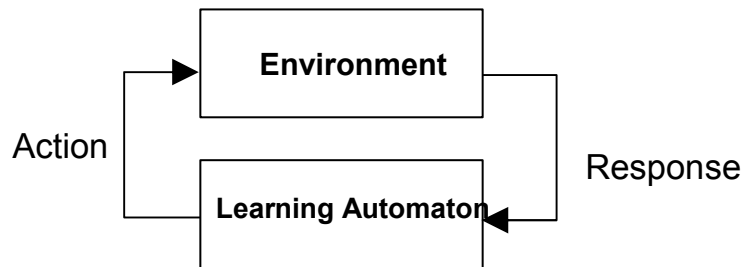
The Best Fit decreasing algorithm is performed by first sorting the items into decreasing order then packing one at a time in the bin that have the least amount of space left. If no bin matches, we open a new bin and put the item into this.

Stochastic versions of the bin packing problem were studied in [22, 23], they supposed that the items follow known distributions. Our problem is different from these studied problems. In fact, we suppose that the items have unknown distributions

3.4 Learning automata

The notion of learning automata has been first introduced by M.L Tsetlin [15]. Since this time, a significant amount of work has been reported in the area of learning automata. The appeal of learning automata is its ability to build schemes that mimic human behaviour when acting in a non stationary environment.

Learning automaton is defined as a finite state machine that interacts with a random environment and learns continuously the optimal actions based on the feedback of the environment.



Learning automata acting in an environment

To formally describe a learning automaton we will use the definition stated in [24]. This definition is also commonly used in other literature. The automaton is defined mathematically by the quintuple $\{ \Phi, \alpha, \beta, F(\cdot, \cdot), G(\cdot, \cdot) \}$

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

The automaton is stochastic if either one of the F and G mappings are stochastic.

There are two types of stochastic automaton:

- Fixed structure stochastic LA
- Variable structure stochastic LA

In our project we make use of fixed structure automaton. More details about the design of such an automaton will figure in the design chapter.

4 Solution

4.1 Requirements

- We need first to design a scheme susceptible to provide a solution to our resource allocation problem.
- The solution should be based on learning automata as it is imposed in the project definition
- We should define the environment model with which the automata will interact. In other words, the distribution of the update probability should be studied.
- We need to set a simulation environment to test and evaluate the approach. Thus, we should implement the simulation environment.
- The solution must be able to cope with both dynamic and static environment
- The computational implementation of the approach must be able to handle large web data sets.
- The algorithm should empirically converge in a reasonable time to a near optimal solution
- The automata should yield accurate estimations of the frequencies of change of the of the monitored web pages

4.2 Design Specification

In this section we propose a novel scheme for provisioning the resources of a distributed crawler. We have mapped our resource allocation problem to a variant of the Bin packing problem where the items weights are of unknown distribution.

The problem involves n web sites of different weights to assign to a certain number of distributed crawler nodes in a way that minimize the number of used crawlers.

For the sake of clarity, we dress up the table below explaining the relation between the Bin packing problem and our crawling problem.

Distributed Crawler	Bin packing
Web Site	Item
Proportion of pages to download from a given web site during the current cycle	Item weight
Distributed crawler node	Bin
The polling capacity of a crawler node	The capacity of the Bin

To clarify further these notations we should underline that:

- The proportion of pages to download from a given web site correspond to the sum of polling frequencies of all its web pages
- The polling capacities of the distributed crawler nodes are restricted and identical.

- We consider that the time is discrete. In each time slot an instantiation of every web site weight will be observed.

At this stage, a legitimate question emerges: How to estimate the web sites weights?

How to estimate the web sites weights?

As it is mentioned previously, a web site weight represents the sum of polling frequencies of its web pages.

Each web site consists of a number of web pages that are updated with different periods. We make use of learning automata to adaptively estimate the polling frequencies of web pages that maximize the number of detected updates.

From this perspective, the polling frequency of every web page will be connected to a learning automaton. “Each automaton increases the polling frequency when an update is detected and decreases the polling frequency when an assumed update did not occur” [16] .That is how our scheme provides adaptive behaviour and learns the optimal polling frequencies from the environments responses.

Applying this fashion, the polling frequencies of monitored web pages will move towards their corresponding update frequencies.

As suggested in [9], the polling frequencies of web pages will be learnt using a learning automaton scheme which involves performing a controlled random walk on a discretized space. The table below summarizes the significations of the adopted notations:

Symbol	meaning
L	Number of web sites to crawl
$\Omega=\{S_1, S_2, \dots, S_L\}$	Set of web sites to crawl
$N(S_i)$	Number of pages of the web site S_i , where $1 \leq i \leq L$
X_{ij}	The polling frequency of the i^{th} page of the web site S_j , where $1 \leq i \leq L$ and $1 \leq j \leq N(S_j)$
W_i	The “weight” of the web site S_i , where $1 \leq i \leq L$
C	Distributed crawler node capacity

Every guess X_{ij} , $1 \leq i \leq N(S_i)$ and $1 \leq j \leq L$, is a discrete value among N points:

$\left\{ \left(\frac{1}{N}\right)^\lambda, \left(\frac{2}{N}\right)^\lambda, \dots, \left(\frac{N-1}{N}\right)^\lambda, 1 \right\}$, where N is the resolution of the learning scheme and λ determines the linearity of the discredited solution space.

$W_i(t)$ is defined as the cumulative sum of polling frequencies of S_i web pages.

Formally $W_i(t) = \sum_{i=1}^{N(S_i)} X_{ij}(t)$.

The automaton design:

Consider the page P_{ij} stating for the i^{th} page of the web site S_j
 Let LA_{ij} be the automaton connected to the page P_{ij} and let $s_{ij}(t)$ be its current state .

In regards to these notations the amount $x_{ij}(t)$ would be equal to $\left(\frac{s_{ij}(t)}{N}\right)^\lambda$

Let $V_{ij}(t)$ be the feedback of the environment. The feedback can either favourable or unfavourable:

- Whenever $V_{ij}(t) = 1$ a reward is handed out .Such favourable feedback indicates that the downloaded page is up to date compared to our local copy which is out of date .

Hence, the automaton is rewarded since he chose an action that updated our local copy and therefore improved the freshness of our local repository.

The reward would be increasing the polling frequency of the page.

- Whenever $V_{ij}(t) = 0$ a penalty is handed out .Such unfavourable feedback indicates that our local copy is as fresh as the recrawled web page .Hence , the automaton is penalized since he chose an action resulting in a waste of the crawling resources . The reward would be increasing the polling frequency of the page.

The state of the automaton is updated as described below:

$$\begin{aligned} s_{ij}(t+1) &:= s_{ij}(t) + 1 && \text{If } V_{ij}(t) = 1 \text{ and } 1 \leq s_{ij}(t) < N \\ s_{ij}(t+1) &:= s_{ij}(t) - 1 && \text{If } V_{ij}(t) = 0 \text{ and } 1 < s_{ij}(t) \leq N \\ s_{ij}(t+1) &:= s_{ij}(t) && \text{Otherwise} \end{aligned}$$

The proposed scheme:

We introduce two-phase algorithm for solving this variant of the Bin packing problem.

The First Phase

Roughly speaking, the first phase of our algorithm corresponds to $t = 0$.

First, the weights of the monitored web sites should be initialized. This is achieved by assigning initial guesses of the polling frequencies to the monitored web pages, then computing the web sites weights. The initial distribution of the polling frequencies of pages, in regards to their domains (.com .edu etc..) , will be studied in details in the simulation chapter.

In the term of this initialization phase, we apply the Best Fit Decreasing algorithm to spread the web sites to the distributed crawler nodes.

The Second Phase

This phase corresponds to the time slots ulterior to time slot 0. ($t > 0$)

Our guesses of the web sites weights are not deterministic and evolve over time in a perpetual track of the optimal values. The task becomes more challenging and intriguing when the optimal weights change over the time. Such changes of the weights over the time may create some problems. In fact, overloads in some bins are likely to occur.

Indeed, it is possible that a bin capacity would be exceeded as the sum of the weights of the packed items could be greater than the capacity.

When a bin capacity is exceeded we present two actions to handle this event.

Hence , when an overload occurs we suggest :

- **Either:** Exclude some items from the overloaded bin and respread them to the other bins
- **Or:** Decrease the weights of the packed items in the overloaded bin.

But which alternative to choose is a key problem.

Which alternative to choose?

Here again, we intend to use Learning automata to decide the appropriate action to perform in the case of overload.

The key finding of the proposed automaton design should be:

- Prioritizing the alternative suggesting to decrease the weights
- Adopting the “excluding web sites” alternative when the previous alternative fails.

In more clear words, we intend to keep as much as possible the items in their current bins. Besides, excluding some web sites from the overloaded bin will be performed only when the overload persists seeming to be impending.

Under these assumptions, we will connect a learning automaton to every bin.

Each automaton will increase the probability of choosing to exclude some web sites when an overload occurs and decrease the same probability when an overload did not occur.

In order to prioritize the alternative suggesting decreasing the weights, we impose on the automaton the following restrictions:

- The automaton should not be permitted to select a state below a predefined threshold. Furthermore, the threshold should have a low value.
- The automaton state should be initialized with the threshold.

A properly adjusted automaton scheme can be defined as following:

Consider the bin B_i stating for the bin number i .

Let LA_i be the automaton connected to the bin B_i and let $1, \dots, N$ be its states

Let $s_i(t)$ be the current state of LA_i

Let $Prob_i$ the probability of choosing the alternative suggesting to exclude some items

from the bin when it is overloaded. $Prob_i$ is defined as $Prob_i = \left(\frac{s_i(t)}{N} \right)^2$

Let Th be the threshold state of the automaton. Obviously, Th is a given threshold equal for all the automata.

Let $V_i(t)$ be the feedback of the environment:

- Whenever an overload occurs, $V_i(t) = 1$.
- Whenever an overload did not take place, $V_i(t) = 0$.

The state of the automaton is updated as described below:

$$\begin{aligned} s_i(t+1) &:= s_i(t) + 2 && \text{If } V_i(t) = 1 \text{ and } Th \leq s_{ij}(t) < N-1 \\ s_i(t+1) &:= s_i(t) - 1 && \text{If } V_i(t) = 0 \text{ and } Th < s_{ij}(t) \leq N \\ s_i(t+1) &:= s_i(t) && \text{Otherwise} \end{aligned}$$

After clarifying how to choose the right alternative, we go now through the explanation of the two proposed actions:

1- The first alternative: Exclude some items from the overloaded bin

We suppose that all the nodes of the distributed crawler are interconnected with a network backbone. Thus it is possible to transfer a web site from a node to another.

In this case, when the capacity is exceeded we will exclude some websites from the overloaded crawler node. Besides the choice of the web sites to exclude should also result in a maximization of use of the bin.

This problem can be seen as a variant of the knapsack problem called Subset Sum Problem. Given a set of items $Q = \{w_a, w_b, w_c, \dots\}$ packed in the same bin, find a subset Y of Q whose cumulative sum is maximum without exceeding the capacity of the knapsack (or bin).

The problem can be solved efficiently by one of the exact algorithms described in [14]

2-The second alternative: Decrease the weights of the packed items

According to this alternative, when the bin is overloaded, we will choose to reduce the polling frequencies of some web pages in order to make all the items fit in the considered crawler node. We center our greedy algorithm on this remark which relates to the theory of probability:

The more a page polling frequency is penalized in the previous time slots the more probable it would be so in the next time slot.

Hence, it is judicious to have a history of rewards and penalties to reduce just the polling frequencies of the most penalized pages. To store the history of rewards and penalization we attach to every page a circular register of length L .In other words, we will store the last L rewards and/or penalizations .If a new reward or penalization is handed out, this last action will be placed in the head of the registry and all the (L-1) remaining actions will be shifted by one cell.

In the term of each time slot, we will get an eventual set of excluded items to repack. We use the Best Fit Decreasing algorithm to respread again the moving web sites to the different nodes of the distributed crawler.

Remark:

It seems to be crucial to underline that both : the algorithms for solving the Subset Sum Problem and the Best Fit Decreasing algorithm interoperate because they aim both at maximizing the use of the bin. This could be seen as a coherent and harmonious aspect of our scheme.

For instance, this is would not be the case if we had used instead of the Best Fit Decreasing algorithm the Worst Fit Decreasing algorithm. In fact, Worst Fit Decreasing algorithm places the items in the emptiest existing bin whereas all the algorithms for solving the Subset Sum Problem aim at maximizing the use of the bin.

4.3 Implementation

Our proposed solution was implemented using java.

The figure 1 illustrates the UML class diagram that describes our program structure.

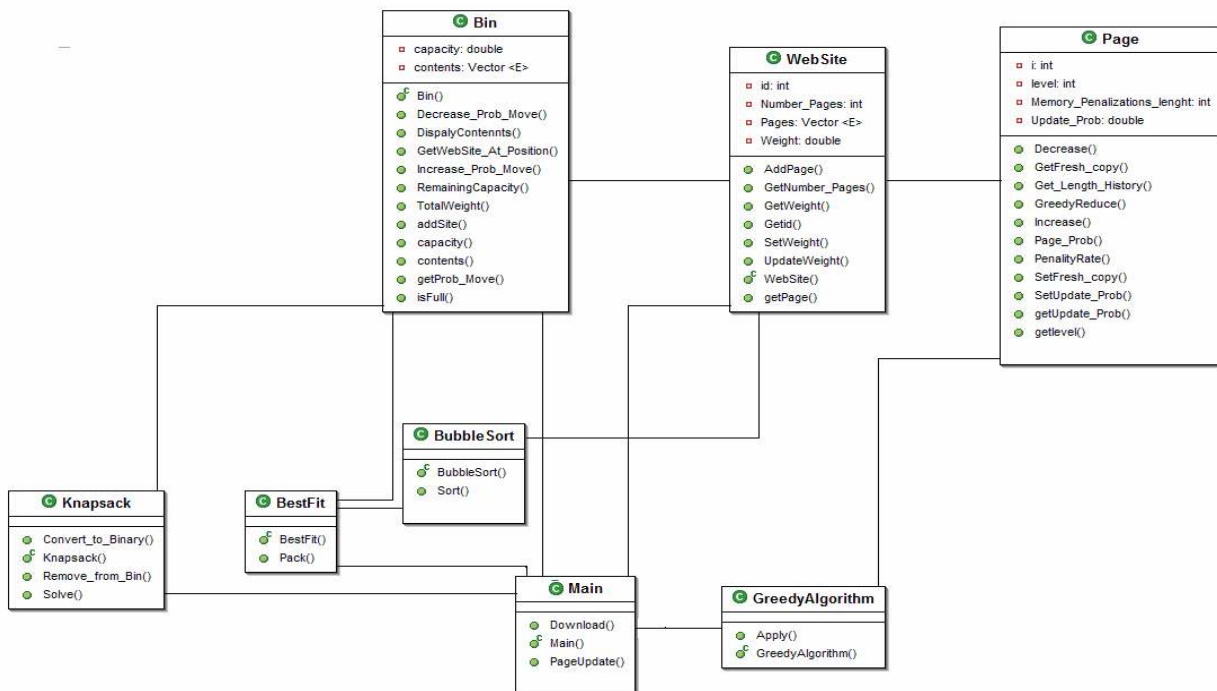
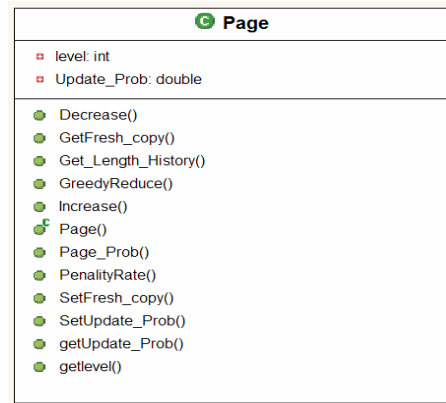


Figure 1-Class Diagram

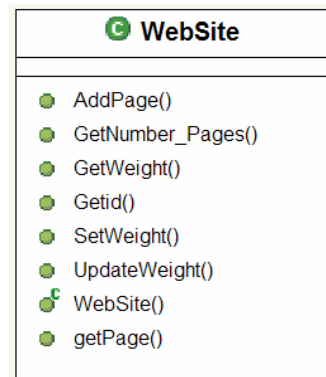
The starting point of our program is the initializing the bin-packing scheme .The user is prompted to specify the main parameters of the simulation namely the crawler capacity ,the number of web sites to monitor, the number of web pages of each web site ,the distribution of the update probability and the initial values of the polling frequencies .

Once all these parameters are defined, our program proceeds by creating objects of the type `WebSite`. Pages are assigned to web sites manipulating the use of a vector to range all the objects of the type `Page` belonging to the same web site.

Clearly, each web page is represented by an object of the type `Page` .In addition each web site is represented by an object of the type `WebSite`.



The next step is to compute the initial weights of the monitored web sites. This is achieved using the method `Getweight` which returns the weight of web site.



Then we perform best fit decreasing algorithm to pack the available items according to their initial weights.

As it is known, the best fit decreasing algorithm consists of two phases. First we arrange the items in a decreasing order. To fulfil this requirement we implemented the sorting algorithm Bubble sort, also known as exchange sort. Then, we perform the best fit algorithm to pack the items into the bins. (Note that Best Fit is distinct from Best Fit Decreasing).

Hence, to implement Best Fit Decreasing algorithm we developed the two classes: Best Fit and Bubblesort.

The Best Fit Algorithm is explained below:

Weight (The weight of the item to pack)

IBF (Index of the best fit bin)

IBF= -1

I=0

Min=Capacity

```

FOR EACH BIN IN THE VECTOR BINS {
  If (BIN.RemainingCapacity () >= Weight )
      If (BIN.RemainingCapacity () - Weight < Min)
          {
              Min= BIN.RemainingCapacity ()-Weight ;
              IBF =i;
          }
      I++;
}

If (IBF = -1)
{
    Create a new Bin to pack the item
}
Else
{
    Pack the Item in the Bin indexed by IBF
}

```

After the performing the initialization phase as described above, we start iterating. During every time slot, the LA learn the polling frequencies

Every web page is connected with learning automaton. The two methods Increase () and decrease () implemented in the class Page permit to adjust the polling frequency with regards to the environment responses.

The principles of the environment simulations are described below:

```

Pi (Probability of update of the page i)
Fresh copy i (Boolean variable that indicates if the web page copy is up to date or out of date)

If (Pi > Random (0, 1)) {
    Fresh copy i = False // When the page is updated our copy becomes out of date
}

If ( Fi > Random(0,1)) {
    Download the page i;
}

Else Do Nothing

```

In addition, the principals of our learning scheme, which invokes the environment is outlined below:

```

Fi (Polling frequency of the page i)

If ( Page i is Downloaded ) {

    If( Fresh copy i =True ) {
        Decrease Fi ;
    }

    Else {
        Increase Fi;
        Fresh copy i = True ;
    }

}

```

During every run , we first estimate the new values of the polling frequencies of the monitored web pages ,then every web site is weighted again using the method UpdateWeight(). Each bin capacity is computed using the method TotalWeight() . If a Bin is overloaded, we make use of its connected learning automaton to decide the appropriate action to perform.

The alternative of excluding some items will be chosen with the probability getProb_Move() . We choose to exclude some items from the overloaded bin when getProb_Move() >random (0 ,1) . In contrast, when getProb_Move() > random (0 ,1) , we choose to reduce the polling frequencies to overcome this unwanted situation.

In the case of the first alternative, we will apply an exact algorithm of the Maximum Subset Sum Problem. To achieve this goal, we implemented the class Knapsack Problem which, given the overloaded bin , returns a vector of the excluded items . This is achieved using the method Solve.

The implementation of an exact algorithm for solving the Maximum Subset Sum Problem is described below:

W = (w ₁ ,w ₂ , ..., w _n)	(the items vector)
C	(the capacity of the Knapsack)
x	(an integer such as $0 \leq x \leq 2^{n-1}$)
X = (x ₁ , x ₂ ,....., x _n)	(the binary representation vector, of length n, of the integer x .)

Remark: The component x_i of the vector X indicates the presence or absence of the item w_i .

```

Max :=0;
For (x=2n-1, x>0 ;x --) {

If ( <X, W> ≥ Max) && ( <X, W> < C ) )

    // Comment: Note that <X, W> =  $\sum_{i=1}^n x_i w_i$ 

    {
    Max :=<X,W> ;
    X* :=X ;
    }
}

```

Maximum Subset Sum = Max ;

Remark: For every i , $1 \leq i \leq n$, the component x_i^* of the vector X^* indicates the presence or absence of the item w_i .

The second eventual alternative is to reduce some of the polling frequencies to overcome the overload. This is performed using the class GreedyAlgorithm. In order to store the story of the last penalizations we use a circular register of a fixed length. We start first by reducing the polling frequencies of the most penalized and we stop the endless loop when the total weight of the bin becomes less than the capacity.

In the end of each run we collect the excluded web sites in the vector called OutgoingWebSites (in the main class) then we respread them again using Best Fit Decreasing algorithm.

4.4 Validation and Testing

In this chapter, we present the results of the conducted simulations.

Since we don not dispose of real data sets to crawl, we simulated a test environment where web pages changes occur in a stochastic manner.

To evaluate our proposed scheme we define two environments:

- Static environment
- Dynamic environment

4.3.1 Static environment:

Recent studies on the web dynamics have shown a strong relationship between the update probability of a web page and its top-level domain. These studies were first pioneered by Cho and Garcia-Molina who examined the update pattern of web pages through experiments conducted on 720,000 web pages.

A similar larger scale experiment had been conducted in [17] on 151 million pages crawled once every week for a period of 11 weeks.

Both studies highlight that the web pages change at different rates with accordance to their domains. In fact, they claim, that the domains .edu and .gov are the most static whereas .com and .net are the most dynamic. For instance, Cho and Garcia-Molina confirm through experiments that 50% of the com domain changes every 11 days while it takes 4 months for the .gov domain to reach the same amount of changes.

A-Fixed probability according to top levels domains

We make use of the previous assumption to create our update probability distribution. Thus the web pages update probabilities are chosen fixed according to their top levels domains: A choice of the update probability distribution is described as following:

	Update probability
.com	0.3
.net	0.25
.edu	0.1
.org	0.05

Remark:

The stated distribution is chosen randomly .Nevertheless the distribution seems to be qualitatively reasonable since it verifies the claim stating that the domains .com and .net are more dynamic than .edu and .org:

▪ Test 1:

Our first test was conducted on 20 web sites belonging to 4 different domains.

	Number of web sites	Number of pages
.com	8	131 , 201 , 341 , 567 , 531 ,36 ,451 ,121
.net	6	124 ,174 ,402 ,322,49 , 615
.edu	4	235,135,95 ,343
.org	2	32 ,123

We chose the crawler capacity 400 pages.

We chose N, the number of automaton states, equal to 8

W initialized all the automata initial states to 1.

We conducted a series of tests with different simulations runs.

The following table summarize the tests results:

	Bin 0	Bin 1	Bin 2	Bin 3
50 runs	320.5	399.875	315.625	0
500 runs	385.75	349.125	382.875	203.0
1000 runs	360.875	379.875	392.5	202.125
5000 runs	374.75	369.875	387.875	191.37

To have a more clear overview of the evolution of the bins number and weights through the time we dress figure 2.

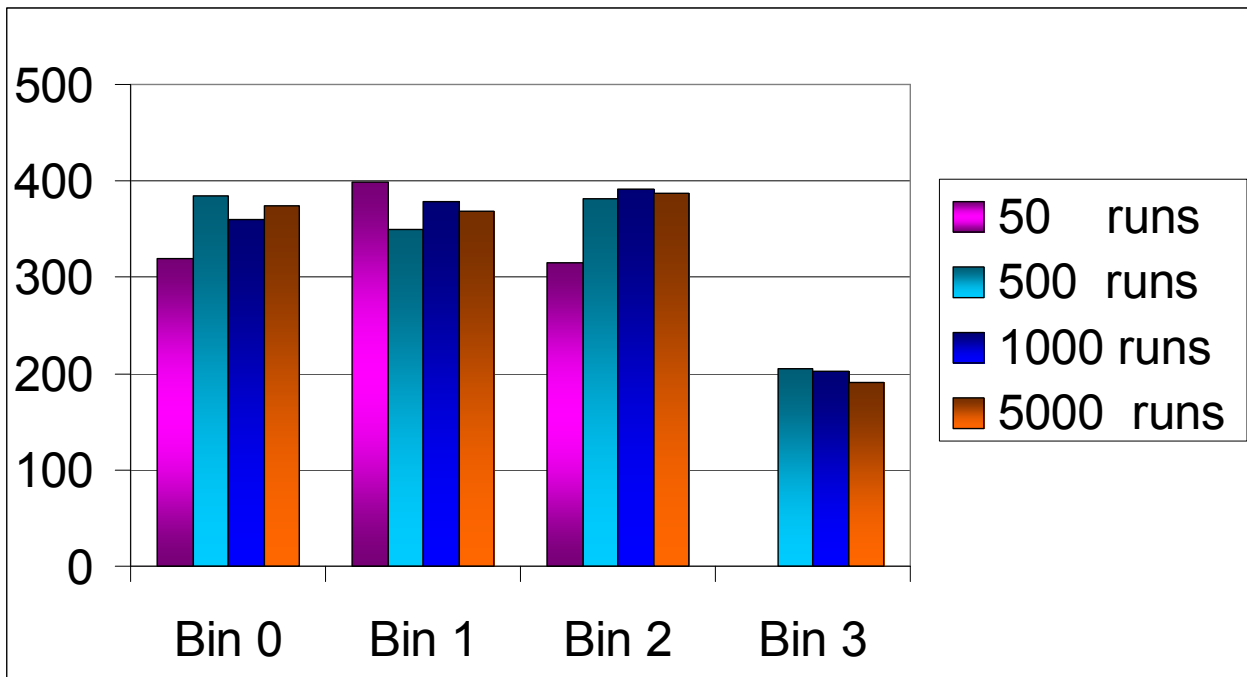


Figure 2

In order to draw conclusions concerning the efficiency of our scheme we compare the obtained results to the optimal solution. However, we should define first what we mean by optimal solution.

The optimal solution to our allocation resource allocation problem requires that every web page polling frequency is equal to its corresponding update frequency (or update probability). Hence the optimal solution to our problem could be obtained by weighting the web sites according to their update probability distributions then spreading them to the crawler nodes using best fit decreasing algorithm.

Remark:

Under these assumptions ,the called “optimal” solution could be sub optimal if the best fit decreasing algorithm does not succeed to pack the items using the minimal number of bins .However the optimality of the packing algorithms does not embrace the focus of our project . (The bin packing problem is NP hard problem. Nevertheless, the best fit decreasing algorithm achieves a near optimal solution)

Therefore, the optimal solution to our resource allocation problem is achieved by using 4 crawler nodes as described below:

	Bin 0	Bin 1	Bin 2	Bin 3
Weight	399.2	399.01	399.7	25.75

With comparison to the optimal solution, we confirm that the results of the test number 1 are near optimal.

In fact, as the automata updated the polling frequencies, the number of bins increased in a steadily manner and reached the optimal desired value: 4 bins.

We also observe that the performance improved through the time and the solution finally converged towards the optimal number of crawlers. The stability was attained over 5000 simulations runs

We underline also for all the simulations runs (50, 500, 1000 ad 5000 runs) the available bins were used in an efficient manner. In fact we can observe that the bins were almost full during all the time steps.

The estimations of the web sites lead by the automata are to some extent accurate. After 5000 runs the automata returned a total weight of the web sites equal to 1323.625. ($1323.625 = \text{total size of the 4 bins}$). Such total weight is more or less equal to the optimal web sites weight: 1223.66 ($1223.66 = 399.2 + 399.01 + 399.7 + 25.75$)

▪ Test 2:

The second test will differ from the first just in the number of the automaton states.

In fact, we will conduct the second test with 16 automaton states.

After 10 000 simulations runs we reached the following stable solution:

	Bin 0	Bin 1	Bin 2	Bin 3
10 000 runs	383.3125	376.5	378.4375	61.25

The result is again near optimal: 4 bins. The number of used bins is the same as the optimal solution.

We observe also that performing with 16 states automaton yield higher accuracy than the 8 states configuration. However, this accuracy is achieved to the detriment of the speed convergence. In fact, it takes almost twice the previously spent time to converge towards a near optimal solution.

Here is a comparative brief overview of the results of the two conducted tests

	Optimal solution	Test 1 (N=8 , runs=5000)	Test 2 (N=16 , runs=10000)
Total web sites weight	1223.66	1321	1199.5

The following table summarize the results obtained in the test 1 and 2 compared to the optimal solution.

Web Site Number	Number of web pages	Optimal weight	Test 1 LA N=8 , Runs=5000	Test 2 LA N=16 , Runs=10 000
0	131	$131 \cdot 0.3 = 39.3$	42.125	38.9375
1	201	$201 \cdot 0.3 = 60.3$	55.25	60.4375
2	341	$341 \cdot 0.3 = 102.3$	105.75	101.9375
3	567	$576 \cdot 0.3 = 172.8$	168.125	167.125
4	531	$531 \cdot 0.3 = 159.3$	160.375	155.875
5	36	$36 \cdot 0.3 = 10.8$	10.875	10.0
6	451	$451 \cdot 0.3 = 135.3$	145.625	133.1875
7	121	$121 \cdot 0.3 = 36.3$	38.25	32.9375
8	124	$124 \cdot 0.25 = 31$	34.375	31.3125

9	174	174*0.25=36.75	44.5	38.6875
10	402	402*0.25=100.5	105.75	89.3125
11	322	322*0.25=80.5	85.375	77.25
12	49	49*0.25=12.25	15.25	10.3125
13	615	615*0.25=153.75	154.125	145.4375
14	235	235*0.1=23.5	41.625	29.0
15	135	135*0.1=13.5	21.75	15.8125
16	95	95*0.1=9.5	15.75	10.5
17	343	343*0.1=34.3	58.25	41.375
18	32	32*0.1=3.2	4.125	2.0625
19	123	123*0.1=12.0	15.75	8.0

Performance metrics:

To evaluate the performance of the automata we define the following performance metric:

$$R = \frac{\text{Number of updated web pages}}{\text{Number of visited}}$$

R= Number of visited

This metric indicates the rate of “successful” downloads. We mean by a successful download a download which reveals a changed page and consequently improves the freshness of our local repository.

The maximum possible value of R is 1. It corresponds to ideal case where all the downloads result in new information.

R depends empirically on the resolution of the automaton. A larger value of N improves significantly the performance. This is merely explained by the fact that a larger value of N yields better accuracy and therefore more successful downloads are performed.

Figure 3 illustrates that the 16 states automata perform better than the 8 states automata. However this is achieved in the detriment of the convergence speed.

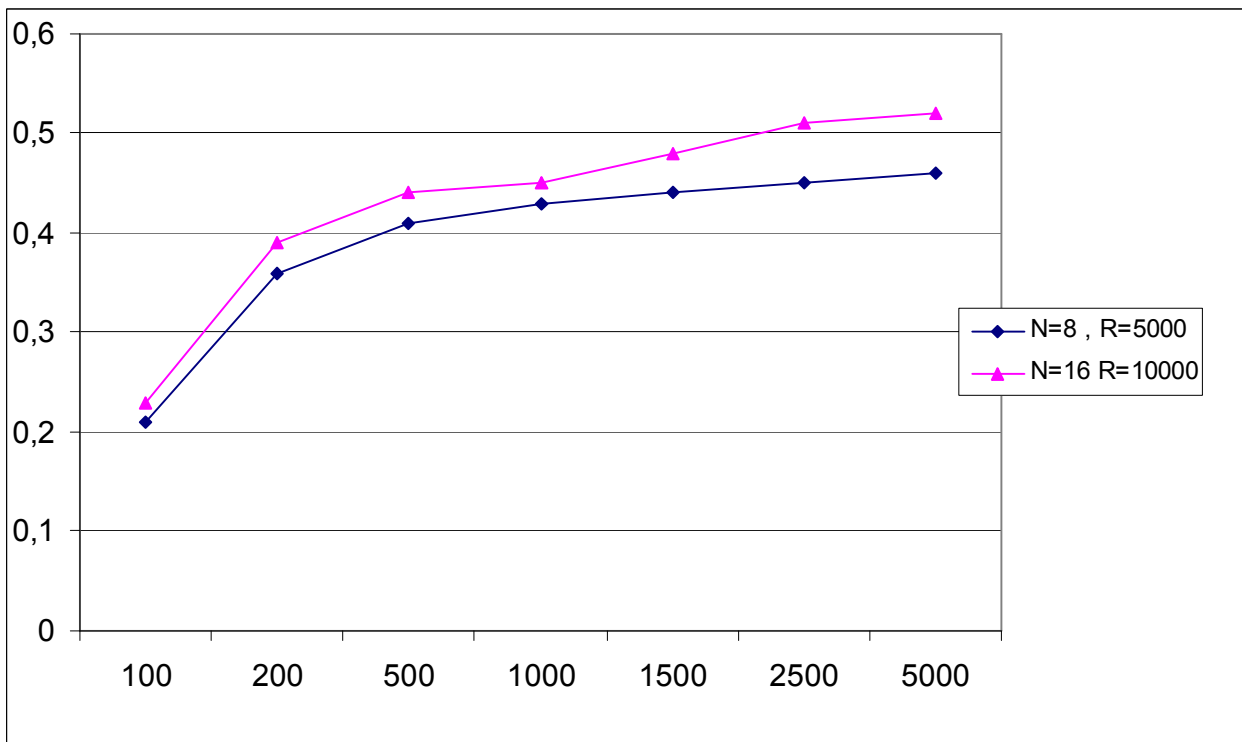


Figure 3

Another interesting performance metric is the rate of updates per time step. We formally define the rate of updates per time step by:

Rate of updates (n) = Number of updates in the current time slot / Total number of pages.

Figure 4 illustrates the evolution of the rate of updates over the time.

The first obvious conclusion is that the 16 states automata detect a higher rate of updates than the 8 states automata.

On the other hand the optimal Rate of updates is equal to:

$$\text{Optimal Rate of updates} = \text{Prop}(\text{com}) * \text{Prob}(\text{com}) + \text{Prop}(\text{net}) * \text{Prob}(\text{net}) + \text{Prop}(\text{edu}) * \text{Prob}(\text{edu}) + \text{Prop}(\text{org}) * \text{Prob}(\text{org})$$

Where $\text{Prop}(\text{domain}) = \text{Total Number of pages of domain} / \text{Total number of monitored pages}$

And $\text{Prob}(\text{domain}) = \text{the update probability of the domain}$

$$\text{Hence the Optimal Rate of Updates} = \frac{2379}{5028} * 0.3 + \frac{1686}{5028} * 0.25 + \frac{808}{5028} * 0.1 + \frac{155}{5028} * 0.05$$

→ The Optimal Rate of Updates = 0.2433

With 16 states automata, our learning scheme attains a rate of updates around 0.13. Compared to the optimal value, this value is to some extent satisfactory. However, a better performance is expected as the number of automata states increases.

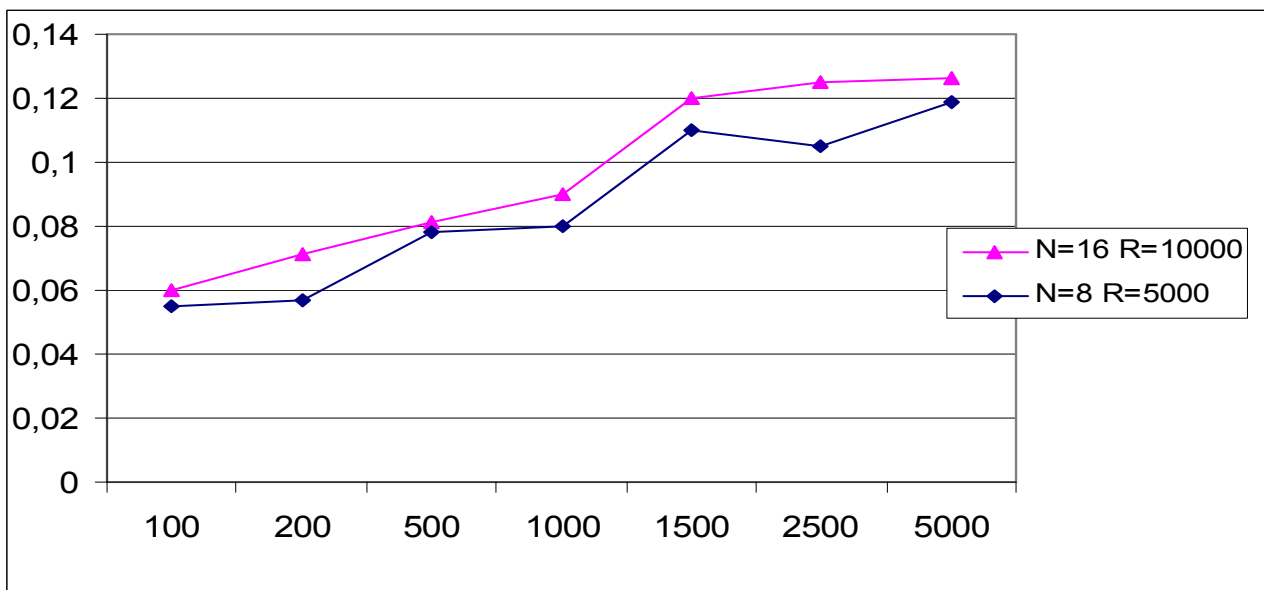
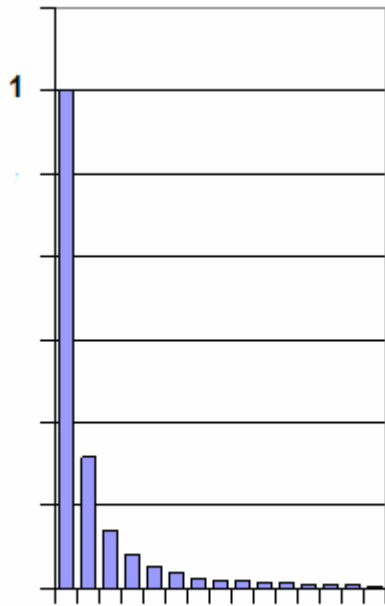


Figure 4

B-Fixed probability according to Zipf function:

In this section we make use of the Zipf distribution to model the update patterns of web pages. Such distribution seems to be more realistic than a fixed distribution according to top levels domains. The Zipf distribution was referred in [8, 9, 18, 19] as a model of the web pages changes.



Example of a Zipf distribution

Here again we propose to relate the update probability distribution to the top level domain.

Thus the skewed ness θ of the distribution will depend on the web pages domain.

We choose as skewed ness of the distribution the following values:

- $\theta=0.75$ for the domain .com. In this meaning, the least skewed distribution refers to the most dynamic domain.
- $\theta=1$ for the domain .net
- $\theta=1.5$ for the domain .edu
- $\theta=2$ for the domain .org. In fact, a highly skewed distribution refers to static data

For the simulations we chose as data 20 web sites from 4 different domains:

	Number of web sites	Number of pages
.com	8	131 , 91 , 91 , 87 , 91 ,46 ,61 ,71
.net	6	74 ,84 ,62 ,92,89 , 105
.edu	4	75,95,105 ,113
.org	2	82 ,93

We chose $N=16$ levels.

The bin capacity=80

After 200 runs, we obtain the following results:

	Bin 0	Bin 1
200 runs	79.1875	29.4375

Therefore, the optimal solution to our resource allocation problem is achieved by using 2 crawler nodes as described below:

	Bin 0	Bin 1
Optimal Weight	78.98175023108985	32.58512586169496

Here is a comparative brief overview of the results of the two conducted tests. The estimations of the web sites lead by the automata are accurate. After 200 runs the automata returned as total weight of the web sites equal to 108,625 .(108,625 =total size of the 2 bins). Such total weight is more or less equal to the optimal web sites weight: 111.56687609278475 (111.56687609278475=78.98175023108985+32.58512586169496).

	Optimal solution	N=16 , Runs=200
Total web sites weight	111.56687609278475	108,625

4.3.2 Dynamic environment:

In this section we test the adaptability of our scheme by confronting it to a dynamically changing environment. In order to simulate such environment we change the update probabilities every 2000 time slots by switching the probability of update of every web page with the succeeding web page in the order.

We choose as initial parameters of the experiment the same parameters chosen the precedent test (The same parameters as the test related to fixed probability according to Zipf function).

In other words, we choose the same web sites with the same distribution.

We choose as resolution of the automaton 16 states.

We run the experiments 1000 time slots and we check the results every 200 time slots.

The experiments show that the results are the same for $t=200$, $t=400$, $t=600$, $t=800$ and $t=1000$. In fact, in all these cases the solution was attained using two bins. The first bin has a total weight of 79.1875 and the second has a total weight of 29.4375.

Hence the sum of the bins weight is 108.625. This value is near optimal compared to the optimal value 111.56687609278475.

The most notable observation is that our solution has shown an adaptive behaviour. In fact, after each environment switch, the solution recovers after less than 200 time slots and converges online.

5 Discussion

The conducted results seem to be particularly promising. In fact, we tested our scheme in both dynamic and static environment. The proposed solution has shown an adaptive behaviour in both cases .The solution moves steadily towards the optimal number of bins. Moreover, the solution turns into stability after a finite number of time slots necessary to learn the environment.

In addition, the tests have shown an efficient use of the bins capacity. Most remarkably the items are spread to the bins near optimal way.

However we can not conclude about the accuracy of our automata when the update probabilities of the web pages are extremely low. Indeed, handling such values demands a high number of automaton states and simulations runs which was not done in our project due to limitations in computational resources.

In our project we used fixed structure automaton to estimate the polling frequencies of web pages. However, implementing the approach based on a multiple action variable structure automaton is also worth investigating. Besides, in order to decide the appropriate action to perform in the case of overload we designed a variable structure learning automaton. However, we believe that this automata design needs some additional improvements to provide more accurate decisions.

6 Further work

As future work, we propose to test the scheme in a real web environment and to prove analytically its convergence and not just empirically. We also intend to find application domains for our algorithm. Another possible axe of research is attempting to model the update patterns of web pages as a Zipf distribution depending on top-level domains. In fact, there is no previous work that studied the skewed ness of the Zipf distribution with regards to the top level domains. In addition, it would be beneficial to implement the presented approach using sampling.

7 Conclusion

In this paper, we have proposed a novel solution to the problem of resources allocation of a distributed crawler when the polling capacity is restricted. We have mapped this problem to variant of the Bin packing problem where the characteristics of items are unknown.

Comprehensive experimental results have shown the ability of our scheme to cope with both dynamic and static environments .In fact, the solution was able to adapt to the environment. In addition, we empirically demonstrated that the solution converges in a finite number of time slots necessary to learn the environment.

References

- [1] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In Proc. WWW conf., April 1998.
- [2] A. Heydon and M. Najork. Mercator: A scalable, extensible web crawler. In Proc. WWW, 2(4):219-229, December 1999.
- [3] S. Lawrence and C. L. Giles. Searching the World Wide Web. *Science*, 280(5360):98-100, April 1998.
- [4] J. Cho, H. Garcia-Molina. The evolution of the web and implications for an incremental crawler. In Proc. Of 26th Int. Conf. On VLDB, September 2000.
- [5] Cho, J., Garcia-Molina, H. Estimating frequency of change. *ACM Transactions on Internet Technology*, 3(3): 256-290, 2003.
- [6] B. E. Brewington and G. Cybenko, "Keeping up with the changing Web," *Computer*, vol. 33, no. 5, pp. 52-58, 2000.
- [7] J. Cho and A. Ntoulas. Effective change detection using sampling. In Proc. 28th VLDB Conf., Hong Kong, China, 2002.
- [8] Ole-Christoffer Granmo, B. John Oommen, Svein-Arild Myrer and Morten G. Olsen. Learning Automata-based Solutions to the Nonlinear Fractional Knapsack Problem with Applications to Optimal Resource Allocation. *IEEE Transactions on Systems, Man and Cybernetics*.
- [9] Ole-Christoffer Granmo, B. John Oommen, Svein-Arild Myrer and Morten G. Olsen. Determining Optimal Polling Frequency Using a Learning Automata-based Solution to the Fractional Knapsack Problem. *IEEE International Conferences on Cybernetics & Intelligent Systems (CIS) and Robotics, Automation & Mechatronics (RAM)*, Bangkok, Thailand, June 2006.
- [10] Cho, J. and Garcia-Molina, H. (2002). Parallel crawlers. In Proceedings of the eleventh international conference on World Wide Web, pages 124-135, Honolulu, Hawaii, USA. ACM Press.
- [11] Boldi, P., Codenotti, B., Santini, M., and Vigna, S. (2004a). UbiCrawler: a scalable fully distributed Web crawler. *Software, Practice and Experience*, 34(8):711-726.
- [12] D. Zeinalipour-Yazti, M. Dikaiakos, "Design and Implementation of a Distributed Crawler and Filtering Processor," *The Fifth Workshop on Next Generation Information Technologies and Systems (NGITS'2002)*, Caesarea, Israel, June 25-27, 2002
- [13] K. S. Narendra and M. A. L. Thathachar, "Learning automata: A survey," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-14, pp. 323-334, 1974.
- [14] Martello S and Toth P (1990). *Knapsack Problems*. Wiley: New York.
- [15] 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.
- [16] Morten Goodwin Olsen "Adapting to feasible polling rate in an incremental web crawler using learning automata" *Proceedings from the Workshop on Web Accessibility and Met modelling* April 14-16, 2005
- [17] D. Fetterly, M. Manasse, M. Najork, and J. L. Wiener. A large-scale study of the evolution of web pages. In *Proceedings of the Twelfth WWW Conference*, Budapest, Hungary, 2003.
- [18] 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.
- [19] 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.
- [20] http://en.wikipedia.org/wiki/Bin_packing_problem
- [21] Coffman Jr., E.G., Garey, M.R., and Johnson, D.S., "Approximation Algorithms for Bin-Packing -- An Updated Survey," *Algorithm Design for Computer System Design*, G. AusieUo, M. Lucertini, and P. Serafini Ed., Springer-Verlag, 1984, pp. 49-106.
- [22] Stadje, Wolfgang Bin-packing problems for a renewal process. *Annales de l'institut Henri Poincaré (B) Probabilités et Statistiques*, 26 no. 1 (1990), p. 207-217
- [23] Hoffman U. A class of simple stochastic online bin packing algorithms. *Computing* 1982;29:227-39

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