

# Distributed Resource Allocation Algorithm

IKT-407  
Autumn 2006



<b>Author :</b> Trabelsi Walid	<b>Supervisors :</b> Leiming Chen Morten Goodwin Olsen
<b>Version : 1.0</b> <b>Status : FINAL</b>	<b>Pages:</b> (including this page) <b>Modified date:</b> 2006-11-26
<b>Keywords:</b> Resource Allocation Problem, Object Migration Automaton, Time of download, true download, Simulated environment, Java.	
<b>Abstract :</b> <p>Due to the explosive growth of the Internet, web search engines are becoming increasingly important as the primary means of locating relevant information. This rapidly growing amount of data on the web leads to raising new challenges to allocated resources in distributed web crawler. In this report, we study the use of the Object Migration Automaton (OMA) for Distributed Resource Allocation Problem. The objective of our approach is to position web sites downloaded utilizing the available resources as best as possible and thus minimizing the total duration of download.</p> <p>Our claim here is that the Object Migration Automaton (OMA) can be used to minimize the time of download by measuring the duration of each access and in the same way to maximize the number of true download to improve the web crawler to get the highly efficient crawling systems. Then, we suggest allocating the resources basing on the Object Migration Automaton (OMA) approach. Our solution is tested through a Java implementation and an evaluation by developing an algorithm for a simulated environment.</p>	

## Table of Contents

1.	Introduction.....	3
2.	Problem Description .....	4
3.	Project's Goal.....	5
4.	Background.....	6
4.1	Distributed Web Crawling .....	6
4.2	Improving a web crawler by estimating frequency of change.....	6
4.3	Sampling strategies .....	7
4.4	A Game of Learning Automata.....	7
4.5	Object Migration Automaton (OMA).....	8
5.	Solution.....	10
6.	Requirements .....	11
7.	Design Specification .....	11
8.	Implementation .....	16
9.	Evaluation and testing.....	21
10.	Discussion and Further work .....	24
11.	Conclusion .....	25
	Appendix.....	26
A1	Glossary & abbreviations.....	26
A2	References.....	26

# 1. Introduction

Nowadays, Resource Allocation has a big importance in the whole novel technologies. This importance is for the simple reason the big area concerning resources. Thus, we can find economics resources (products, services, goods...), Natural resources (renewable/non renewable), Humans resources (social services, education, political economic), resources concerning also Telecom domains (frequencies allocation, channel allocation...). This project is concerning resources in the World Wide Web taking the problem of distributing web sites in a web crawler.

The main part of a mechanism used for large scale web assessment machinery, e.g. as a web accessibility measurement tool [5] or a search engine, will be a web crawler (also known as web spider or web robot), possibly distributed. The web crawler can download as many updated web sites as possible compared to the locally stored copies and that the time used on any evaluation of the complete set of web sites is as small as possible.

This can be seen as a Distributed Resource Allocation Problem, as the crawler should minimize the time spent downloading and evaluation of the web sites with limited resources available when the resources available vary between access points in the first way. In the other way, the web crawler should maximize the number of true download when many online web sites are always updated periodically.

In [2, 3], B.J. Oommen and D.C.Y Ma presented the Object Migration Automaton (OMA) as one of two theoretical stochastic learning Automaton solutions for the Object Partitioning Problem (OPP). It demonstrated an excellent partitioning capability that we can assume that resources should be partitioned on the best efficient.

The goal of this project is to present and evaluate a solution using the Object Migration Automaton (OMA) for Distributed Resource Allocation problem. The Object Migration Automaton (OMA) seems particularly promising for Distributed Resource Allocation Problem because it has demonstrated an excellent partitioning capability of objects that can be considered as resources should be allocated on the web.

Our solution is tested through a Java implementation and an evaluation by developing an algorithm for a simulated environment.

The rest of this report is organized as follows: In section 2 we describe the problem of our project and in section 3 we present the goal of the project. We present a review of relevant literature in section 4.

After that it comes in section 5, the description of our solution for Distributed Resource Allocation Problem. We detail the requirements for the testing and evaluation of this problem in section 6. The proposed design of a variant of the Object Migration Automaton (OMA) as a solution to the problem presented in section 2 is then presented in section 7. In section 8, the implementation process is described based on our proposed design in section 7. The Object Migration Automaton is then tested and evaluated in section 9. The discussion is found in section 10. We conclude the paper in Section 11 and offer prospects for further work.

## 2. Problem Description

In this project, we were to test if Object Migration Automaton could be used for large scale web assessment machinery as measurement tool or search engine to involve a distributed web crawler. It is highly desirable that both the crawler can download as many updated web sites as possible compared to the locally stored copies and the time of download, especially the true download, on any evaluation of the complete set of web sites is as small as possible.

This can be seen as a distributed resource allocation problem, as the crawler should minimize the time spent downloading of the web sites with limited resources available when resources available vary between access points. In the same way the crawler should maximize the number of real download which can be defined as one of the methods used to measure the crawler.

This is a concrete example to make the problem of resource allocation easier to understand: That is to say a group of  $N$  persons having to go to have diner in one restaurant. A set of  $M$  restaurants, each restaurant having a capacity of reception  $C_i$ , such that we have some constrains should be undertaken as the total capacity of the  $M$  restaurants is bigger than  $2 * N$ , but each restaurant capacity  $C_i$  is less than the number of persons  $N$ . The problem in this example is how to reach the satisfaction of all the persons to get a good place in one restaurant such that each restaurant can be saturated in normal time with a probability  $P_i$ .

The above example can be seen in the same way used for large scale measuring in a distributed web crawler such that the persons can be considered as the users navigating on the web and the restaurants can be similar to the web sites or the webs pages on the Internet. Furthermore, the capacity of each restaurant seems to be the bandwidth used by the crawler and as the capacity of each restaurant is limited also the bandwidth is always limited when it is used. Another point that the capacity vary between the restaurants, Resources

available vary between access points on the Internet. Finally, the time spent by one person to find a good place in one restaurant can be considered as the time spent by crawler to download and evaluate the web pages.

In general, resources can be allocated everywhere but the problem is how to get the most optimal positions for all of them and waiting to be used in best efficiency? How to handle the limited resources and how to optimize the allocation of these limited resources in web monitoring?

### 3. Project's Goal

The main objective of our project is to develop an algorithm solving a distributed resource allocation problem in web crawling. Then we should create a simulated environment to verify the algorithm. This environment is designed to represent the behavior of the real web as best as possible, both with regards of the behavior of webs sites and access point. The heart our scheme focused especially on two large points.

In a real web environment the duration of downloading the complete web sites which depends [4] highly on:

- The size of the web site
- The uploading capacity of the web site server.
- The physical location of the web site server.

The duration of evaluating web sites from each access point which depends [4] highly on:

- The resources available in this access point (CPU, Memory).
- The number of hours the access point is available. (Some access points may only be partly be available).
- The downloading capacity from the server.

This can be seen as the first goal to minimize the time spent downloading and evaluation web pages and perform further calculations by the crawler. The second goal is concerning the number of true downloads to improve a web crawler downloading web pages when many online data sources are always updated. Then a typical crawler usually revisits the entire set of pages periodically and updates them all. However, if the crawler can estimate how often an individual page changes, it may revisit only the pages that have changed (with high probability), and improve the “number of true downloads” of the local snapshot without consuming as much bandwidth.

## **4. Background**

To accomplish the task at hand, we have been reading about previous work done in the field. In this section we look at different technologies relevant to resource allocation problem.

### ***4.1 Distributed Web Crawling***

With the explosive growth of the Internet, web search engines are becoming increasingly important as the primary means of locating relevant information.

Thus, highly efficient crawling systems are needed in order to download the hundreds of millions of web pages indexed by the major search engines. In fact, search engines compete against each other primarily based on the size and currency of their underlying database, in addition to the quality and response time of their ranking function. Even the largest search engines, such as Google or AltaVista, currently cover only limited parts of the web, and much of their data is several months out of date. (We note, however, that crawling speed is not the only obstacle to increased search engine size, and that the scaling of query throughput and response time to larger collections is also a major issue [6].

Two issues are addressed for a good crawler for large scale web assessment in [6]. First, the crawler needs to have a good crawling strategy, i.e., deciding the order of web sites to download. Secondly, a crawler needs to have a highly optimized system architecture that can download a large number of sites per second while being robust against crashes, manageable, and considerate of resources and web servers.

### ***4.2 Improving a web crawler by estimating frequency of change***

Due to the explosive size of the Internet, many data sources are available online. Most of the data sources are autonomous and are updated independently of the clients that access the sources. For instance, popular news web sites, such as CNN and NY Times, update their contents periodically, whenever there are new developments. Also, many online stores update the price/availability of their products, depending on their inventory and on market conditions [1, 11].

A web crawler is a program that automatically visits web pages and builds a local snapshot and/or index of web pages. In order to maintain the snapshot/index up-to-date, the crawler periodically revisits the pages and

updates the pages with fresh images. A typical crawler usually revisits the entire set of pages periodically and updates them all. However, if the crawler can estimate how often an individual page changes, it may revisit only the pages that have changed (with high probability), and improve the “number of true download” of the local snapshot without consuming as much bandwidth. According to [1,11], the crawler may improve the “number of true download” by orders of magnitude in certain cases, if it can adjust the “revisit frequency” based on the change frequency.

### ***4.3 Sampling strategies***

One of the solution to resource allocation problem use sampling strategies, applying parts of the available resources for sampling and further determine if the sampled web site should be priorities in the remaining resources available [8,10].

Due to limited network and computational resources, however, it is often difficult to monitor the sources constantly to check for changes and to download changed data items to the copies [8, 10]. The idea is to detect as many changes as it can use the fixed download resources that we have. The main idea is to use sampling. That is, we first download a small number of data items from each data source as samples, and use the samples to decide which sources we download more data items from. While the idea is simple, the analysis and experiments show that sampling based policies have great potential and lead to significant improvement.

### ***4.4 A Game of Learning Automata***

This approach [1, 7, 9] involves a Team of deterministic Learning Automata (LA) such that resource allocation problem can be seen as Fractional Knapsack Problem. The resource allocation problem is generally modeled as a knapsack problem with known deterministic properties. However, for practical purposes the web must often be treated as stochastic and unknown.

The Fractional Knapsack Problem concerns  $n$  materials each characterized by its value per unit volume. The problem involves filling a knapsack of fixed volume with a mixture of the materials so as to attain a maximal value

The Nonlinear Fractional Knapsack Problem demonstrates how its solution can be effectively applied to two resource allocation problems dealing with the World Wide Web. The first real-life problem relates to resource allocation in web monitoring so as to optimize information discovery when the polling capacity is constrained [1, 7, 9]. The disadvantages of the currently-reported solutions are explained in the paper. The second problem concerns allocating limited sampling

resources in a “real-time” manner with the purpose of estimating multiple binomial proportions.

#### **4.5 Object Migration Automaton (OMA)**

The standard for evaluating whether the resource object is the most optimal situation can be different, such like time, distance and so on. Using the Object Migration Automaton (OMA) towards partitioning the resource objects to receive the most viable solution seems like a viable approach [2, 3].

In [2, 3], B.J. Oommen and D.C.Y Ma presented the Object Migration Automaton (OMA) as one of two theoretical stochastic learning Automaton solutions for the Object Partitioning Problem (OPP). The automaton is offered a set of actions by which it interacts, and is constrained to choose one of these actions. When an action is chosen, the automaton is either rewarded or penalized. The algorithm works such that the automaton learns the action which has the minimum penalty probability and eventually chooses this action more frequently than the other actions.

The can be seen as a set of  $R$  actions, each action representing a certain class named as an arm. Further for each action, there is a fixed number of a states  $N$ . Additionally, a set of abstract objects  $\Omega = \{O_1, \dots, O_w\}$  move around in the automaton. The state is an integer representing the place of the object  $O$  in its current arm.

Figure 1 shows an example of the Object Migration Automaton (OMA) with three arms, nine objects and four states in each arm.

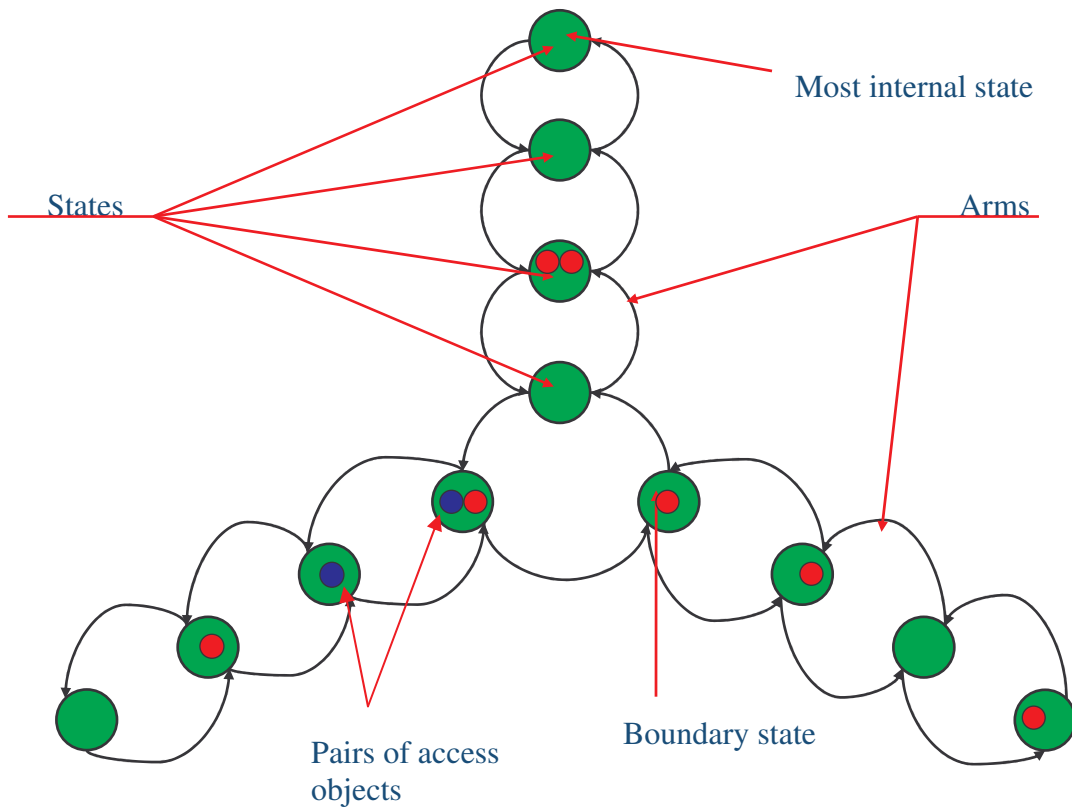


Figure 1: A visual representation of an OMA that have 3 arms with 4 states each

The Automaton as described below:

- Each object  $O_i$  is all the time located in one of the predefined  $N$  states and respectively connected to one of the predefined  $R$  actions.
- Each object  $O_i$  has an action  $\alpha_i$ . This action is dependant on which of the  $R$  arms the object is located in.
- All objects are accessed in pairs ( $O_i$  and  $O_j$ ) and whenever two  $O_i$  and  $O_j$  are accessed they will always be penalized or rewarded.
- Whenever an object is rewarded, it moves one state closer to the most internal state in its corresponding arm. If the object is already located in its most internal state, it does not move.
- Whenever an object is penalized, it moves one state closer to the boundary state in its corresponding arm. If the object is already located in its boundary state, it is candidate to move to one of the other  $R$  arms.
- The objects are rewarded if  $O_i$  and  $O_j$  are accessed together and are located in the same arm.

- The objects are penalized if  $O_i$  and  $O_j$  are accessed together and are located in different arms.
- If  $O_i$  and  $O_j$  penalized and at least of of them are located boundary states of different arms, they switch arms and thus change their action.

## 5. Solution

In order to solve the distributed resource allocation problem we have used Object Migration Automaton (OMA) with some extensions that will be described in the next section.

In [2, 3, 4], B.J. Oommen and D.C.Y Ma presented the Object Migration Automaton (OMA) as one of two theoretical stochastic learning Automaton solutions for the Object Partitioning Problem (OPP). It demonstrated an excellent partitioning capability, and experimentally, this solution converges an order of magnitude faster than the best known algorithm in the literature.

The objective of this project is to present and evaluate a solution using the Object Migration Automaton for Resource Allocation Problem. The OMA seems particularly promising for Resource Allocation Problem because it learns incrementally/on-line, has low computational complexity and has an excellent partitioning capability of resources that why we choose to use Object Migration Automaton (OMA) to solve Resource Allocation Problem.

We have extended the Object Migration Automaton (OMA) and we have used the duration of accessing each object as basic for partitioning in addition to using the order of which the objects are accessed [4]. Our scheme will be used in order to minimize the time spent downloading and evaluating of the web sites by the crawler with limited resources available. Note that the time of download involves the state of the web sites moving around the Automaton and the number of accesses from users. Each time the number of accesses increases from users to the web sites, the time of download decreases.

Furthermore, in the second step and in order to improve the web crawler to use in the best efficient the limited bandwidth by maximizing the number of real downloads; we will estimate the frequency of change with the Object Migration Automaton (OMA) approach. Our scheme will be used to verify that distributing web sites with Object Migration Automaton (OMA) approach can be suitable to detect the change of the web sites.

## 6. Requirements

The solution under development needs some requirements to work satisfactory, we have predefined a few requirements for the Automaton's functionality.

- We need to create a simulated environment to test and evaluate our solution basing on the Object Migration Automaton (OMA). This environment is designed to represent the behavior of the real web as best as possible, both with regards of the behavior of web sites and access points.
- In order to simulate the time of download of the web sites, we need to fix some parameters to be used. In this way, we choose to merge the state of the web site in the Automaton, the number of users accessed to the web site and finally we predefine for each web site, which is seen as an object in our environment, a perfect Arm in the Automaton.
- The Automaton should be able to organize the objects basing on its actions: both objects are in the same Arm and concerning the time of download.
- In order to maximize the number of true download, the Automaton should be able to detect if the web site change in each access.
- If we finish the development of the algorithm as soon as possible, we develop also a graphical user interface GUI of the automation in action, to observe the Automaton's decisions. This GUI should be able to give us an overview of the actions taken around the Automaton.

## 7. Design Specification

Our novel scheme is based on Object Migration Automaton (OMA) and we have extended the OMA described in section 2 in order to solve the distributed Resource Allocation Problem. We have used the time of download as basis for partitioning when an object was accessed.

Before we describe our assumption of the choice of actions, we present the mapping between the developed algorithm and a real web crawler in the table 1.

Symbol	Simulated Environment	Web Crawler
$\Omega = \{O_1, \dots, O_w\}$	Objects	Web sites
$R = \{A_1, \dots, A_n\}$	Arms	Access point
N	Internal states of each Arm	Evaluation duration
T	Time	Time of download

Table 1: Mapping between a simulated environment and a real web crawler

First of all, the objects are randomly placed in the innermost states of the Automaton. The Automaton has a predefined set of starting arms, each with a predefined length. After the corpus has been inserted in the different states, the Automaton can start its process.

The Automaton runs in one specific loop until the Automaton reach the most optimal position of all objects, with an iteration defined as one test and the following action taken based upon this test. A general iteration is outlined below, in figure 2.

Unlike the basic rules of Object Migration Automaton (OMA) that each Object  $O_i$  in the corresponding Arm  $A_j$  and is located in one of the N states, an object may only move from  $N_i$  to either  $N_{i-1}$  or  $N_{i+1}$ . Our scheme involves accessing web sites while measuring the time of download of each access.

It is noticeable that in our solution, an object will always have a perfect Arm. If the object is located in the perfect Arm the time of download will in average be less than if the object was located in any other Arm than the perfect Arm. The perfect Arm for all the objects is predefined in the learning process and it is assumed as a distance that can take the web sites between all the different access points.

In our case, in order to handle the time of download, we had to change the actions which the OMA originally use for the fast object partitioning problem, presented in [2, 3].

Still, we base our actions on the two simple tests:

- 1- Are the pairs of objects accessed in the same arm?
- 2- Compare the time of download of the access objects with the average time of all objects in the corresponding Arm.

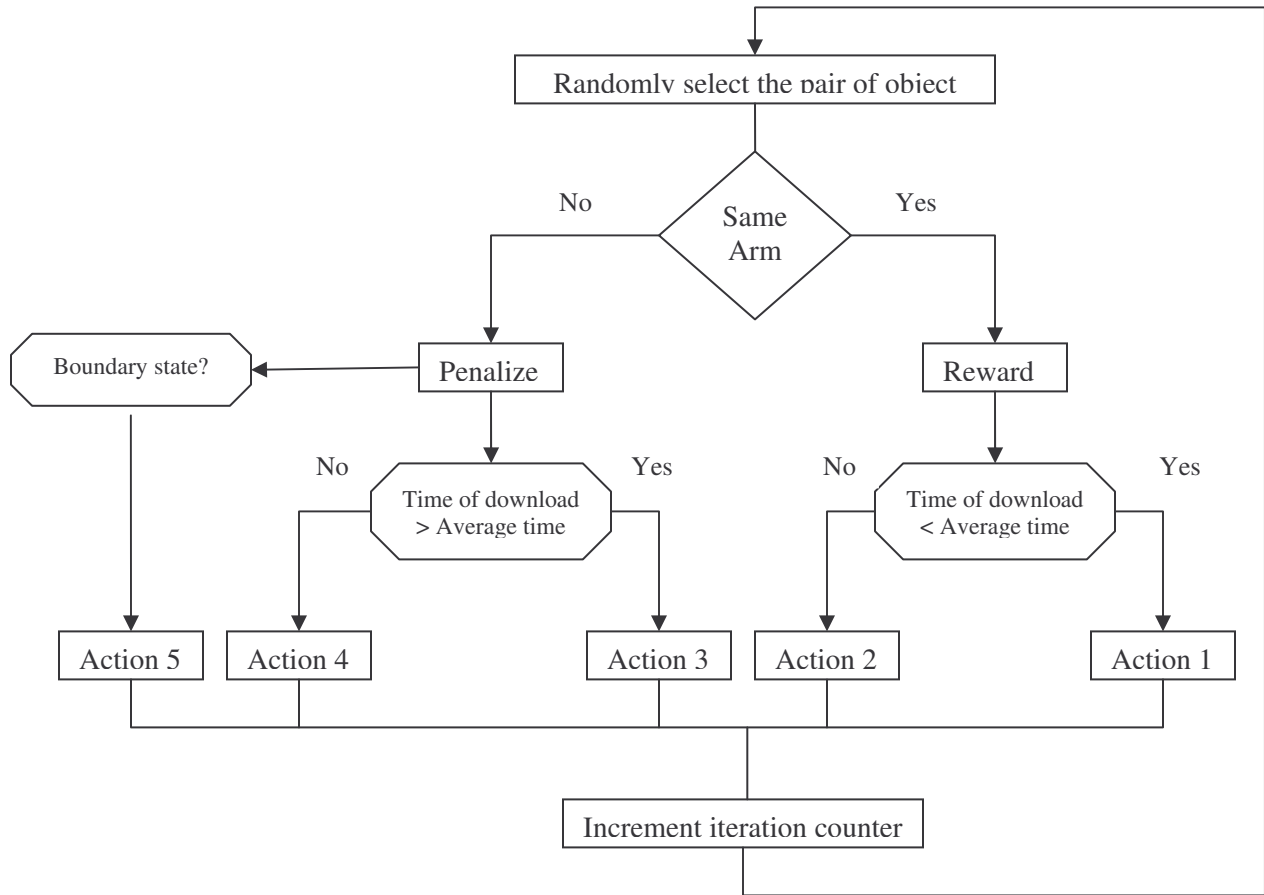


Figure 2: The Automaton Process based on the time of download

The general assumptions of the choice of actions take firstly the localization of the accessed pairs of objects as a basic in the Object Migration Automaton (OMA) approach. In fact, if the objects accessed in pair are in the same Arm then the action should be rewarded, in the other case that the objects accessed in pair are in different Arm then the action should be penalized. To extend the OMA approach, we involve accessing web sites measuring the time of download of each object access and we compare this with the average time of all objects in the corresponding Arm. Then in each time  $t$  each object  $O_i$  is accessed and the duration of download is measured.

the main rules of actions for each crawler run  $t$  are outlined below:

- If the pairs of objects are in the same Arm then the objects are rewarded and in the same way we calculate the time of download of each one of them:

**Action 1:** If the time of download is less than the average duration of all objects in the corresponding Arm, the object jumps two states closer to the most internal state.

**Action 2:** If the time of download is more than the average duration of all objects in the corresponding Arm, the object only moves one state closer to the most internal state.

- If the pairs of objects are in different Arm then the objects are penalized and in the same way we calculate the time of download of each one of them.

**Action 3:** If the time of download is more than the average duration of all objects in the corresponding Arm, the object jumps two states closer to boundary state.

**Action 4:** If the time of download is less than the average duration of all objects in the corresponding Arm, the object moves only one state closer to the boundary state.

**Action 5:** If the pairs of objects are penalized and at least one of them is located in the boundary state, they switch arms and thus change their action.

- Any object that is already located in the most internal state and rewarded then it doesn't move.

Note we are working with an unknown environment. This means that the algorithm can only know the duration of download by actually performing a download. In other word, in order to represent the environment presented above, each web site in our environment is seen as an object with a random stochastic variable. This means that the time of download is represented as a random variable that can only be retrieved after the object has been accessed and the crawler is running.

The heart of our scheme involves also the accessing web sites while verifying the web site change in each access. Then we take advantage of the excellence partitioning capability in [2, 3, 4] to estimate the frequency of change of the web sites. When the web sites, assumed as the objects in the Automaton, are partitioned following the Object Migration Automaton (OMA) approach we can profit from this by developing a function to detect in each access if the web site has changed.

The approach to detect the change of the web sites is described as follow:

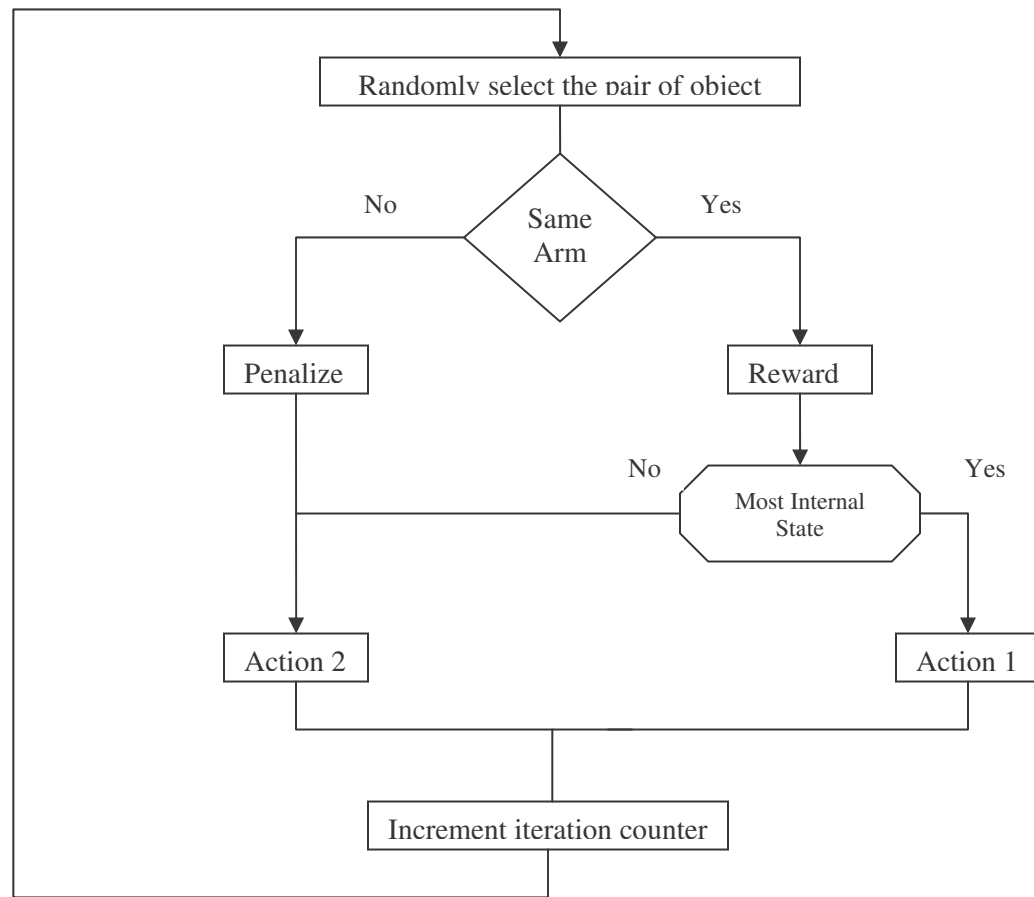


Figure 3: The Automaton Process to detect the change of the web site

The general assumptions of the choice of actions depends if the automaton is either rewarded or penalized.

**Action 1:** if the object is rewarded and it is already located in the most internal state, the object doesn't change because its state doesn't move.

**Action 2:** if the object is penalized or rewarded and don't located in the most internal state, the object changes because its state moves around the Automaton.

We notice that with the Object Migration Automaton (OMA) approach of objects partitioning we can detect the change of the object and the crawler don't need to download all the web sites. But, it only goes for the accessed web sites in the Automaton that detect its changing. It remains then only to update these web sites (objects).

## 8. Implementation

First of all, we present the parameters and the methods used as an implementation of our design described in section 7. The system starts by reading the input stream to be used into the Automaton. This can be done by having some predefined parameters such that the number of starting Arms, the number of pages in each Arm inducing to get the number of objects to be moved around within the Automaton. Many others parameters will be used such that the maximum of states, the perfect Arm array of all different objects, the Boolean “hasChanged” parameter to test if the object has changed. All these parameters are described in the figure 4.

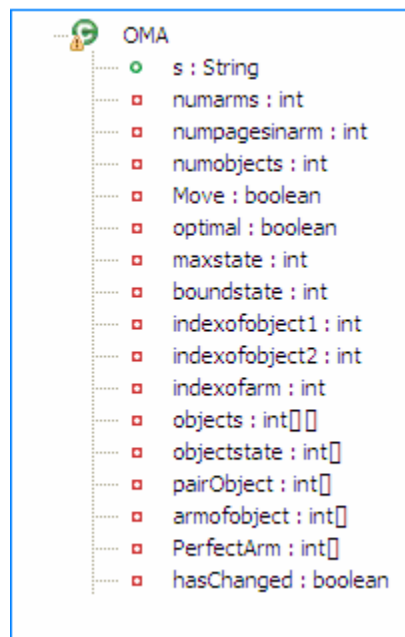


Figure 4: Different parameters used to implement our solution

After predefining the parameters, we can develop the necessary methods that can be used as an implementation of our design. The list of methods is described in the figure 5.

The 3 methods “getnumarms”, “getnumpagesinarm” and “getnumstatesinarm” are specified respectively for the 3 predefined parameters “numarms”, “numpagesinarm” and “numstatesinarm” as input stream.

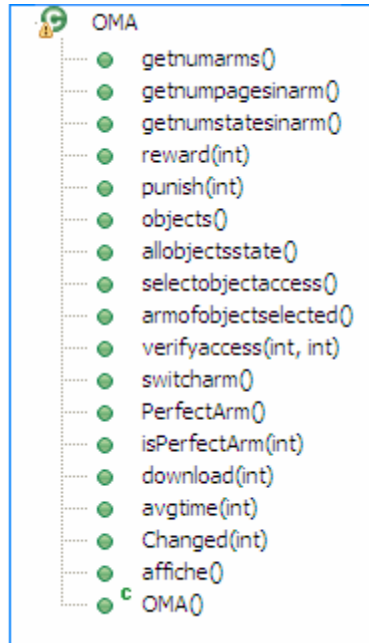


Figure 5: Different methods used to implement our solution

After that, the objects accessed randomly between the different arms when each iteration consist that only one pair of object is accessed and the automaton offered a set of actions concerning the 2 methods : reward and punish.

The next step is to calculate the time of download when each the object is accessed. The time calculated is designed to represent the behavior of the real web as best as possible, this is done with regards if the object is places in his predefined perfect arm in the first step. Because if the object is located in the perfect arm, the duration of access will in average be less than if it was in any other arm. Secondly, we take on consideration the state of the object in the Automaton and finally the time of download depends on the number of users accessing to the web site, seems as an object in the Automaton.

Our assumption to calculate the time of download is outlined in the pseudo code in figure 6.

```

public float download(object){

    numofaccess=rand.nextInt(maxofaccess+1);

    if (isPerfectArm(objec)){

        if (objectstate[objec-1]>=Predefined_state){
            if (numofaccess<Pre_value)
                ..... Time =.....
        }
        else {
            if (numofaccess< Pre_value)
                ..... Time =.....
        }
    }
    else {
        if (objectstate[objec-1]>=Predefined_state){
            if (numofaccess<Pre_value1)
                ..... Time =.....
        }
        else {
            if (numofaccess< Pre_value1)
                ..... Time =.....
        }
    }
    return time;
}

```

Figure 6: Pseudo code of calculating the time of download

Based on both, the location of the objects if they accessed in the same arm as a base of the Object Migration Automaton (OMA) approach and the time of download calculated in the first step, an action is chosen by the Automaton. Our original presumptions of action are outlined in the pseudo code in figure 7.

```

if(same_arm){
    reward();
    if (download (object)<avgtime(armofobject)){
        Move(2 states); //closer to the most internal state
    }else {
        Move(1 state); //closer to the most internal state
    }
} else
{
    punish();
    if (download (object)<avgtime(armofobject)){
        Move(1 state); //closer to the boundary state
    } else{
        Move(2 states); //closer to the boundary state
    }
}

```

Figure 7: Pseudo code of Automaton's actions

Note that in the Automaton's actions, it moves sometimes 1 state as the base of Object Migrations Automaton (OMA) in [2, 3] either rewarded or penalized. Sometimes, it moves 2 states as our extension to OMA by comparing the time of download with the average duration of all objects in the corresponding arm. Choosing to move 2 states and not 3 or 4 or more number of states will be explained in the experiments chapter.

The final method implemented is concerning the change of the object. This method should be able to detect if the object has changed when it accessed. Then we take advantage of the excellence partitioning capability in [2] to estimate the change of the web site in each access. Our assumption to estimate the change of the object is outlined in the pseudo code in figure 8.

We also implemented a graphical user interface for the simulated environment to show the process of the Automaton, and to easily get the information attached to each object in each arm. Figure 9 shows a screenshot of this user interface.

```

public boolean Changed(object){

    for(int i=1; i<=numarms; i++){

        for (int j=1; j<=numpagesinarm ; j++){

            if (objects[i-1][j-1]==object){
                if (stateofObject(object) == maxstate)
                    hasChanged = false;

                else hasChanged=true;
            }
        }
    }
    return hasChanged;
}

```

Figure 8: Pseudo Code of change detection

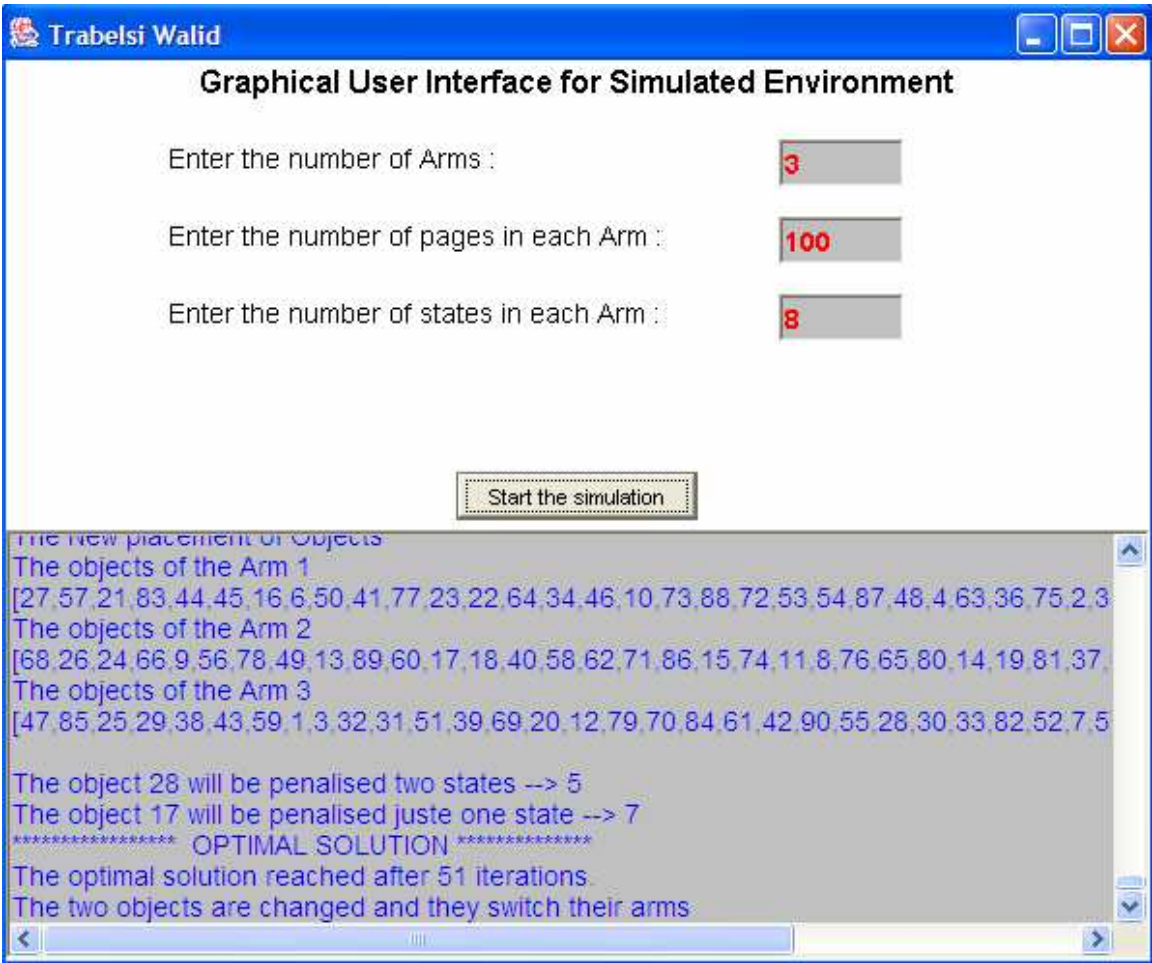


Figure 9: Graphical User Interface for the simulated environment

## 9. Evaluation and testing

In this section, we evaluate our solution and the presented algorithm. In all the test results, the iterations to reach the optimal solution are the mean value of 8 test runs per test. Then all results presented are the outcome of averaging 8 runs.

Note that according to the goal's algorithm to minimize the time of download, the optimal solution is when the time of download of each object is less a fixed time. This fixed time is taken as an average time considering that it can reach the sufficient of the users.

In the first test we used ninety objects per arm, three arms and an arm length of 8 states. We present in the figure 10 the performance of the algorithm to minimize the time of download compared to the initial measurement of time of download.

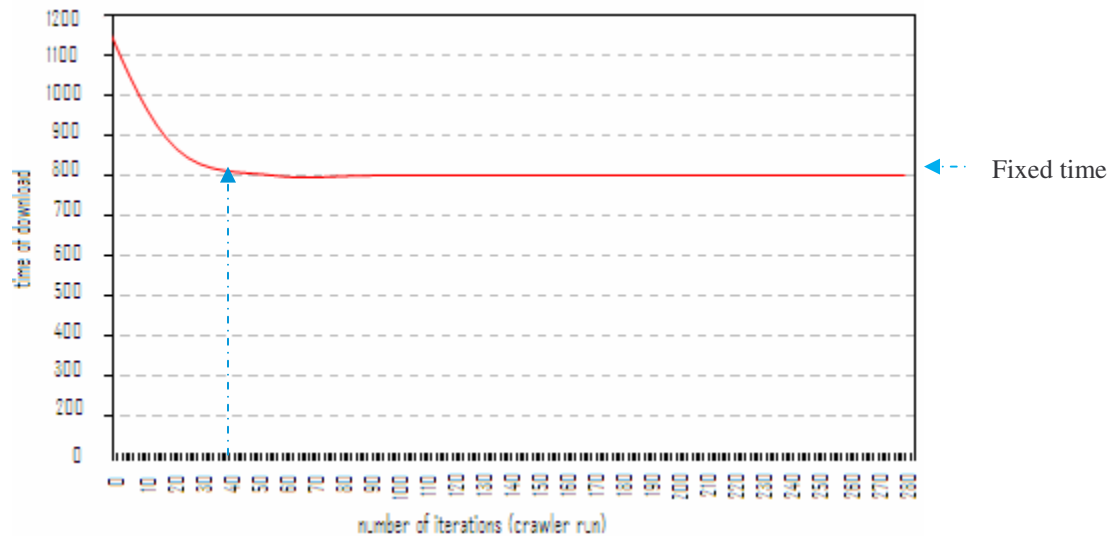


Figure 10

In the figure 10 the behavior of the algorithm in a static environment with 3 arms and 100 objects is shown. Note that the x-axis shows the number of iterations as the total number crawler runs, while the y-axis shows the time of download of accessing all objects. This experiment shows that the behavior of the algorithm starts out having an initial time of download of the objects, but quickly reduces the time of download of each crawl as the number of iterations increases. It reaches the optimal solution after about 40 iterations.

Note that in this experiment we assume that the time of download is static as we work with an unknown environment.

The next test is done to explain and justify why in the Automaton's actions move sometimes 1 state and sometimes 2 states. First of all, all the abstract objects moves around the automaton at least one state, the automaton is either rewarded or penalized. This action is dependant on which of the pair of objects are in the same arm as the base of the Object Migration Automaton (OMA) approach. To extend this approach, the Automaton's actions depend also on the time of download of each object. That is why it should move another state in our solution. However, the question that we can put it is why we choose to move 2 states?

To justify our proposition, we evaluate the same test used in the figure 11 when the Automaton's actions move 3 or 4 states instead of moving 2 states. We combine the result in the same figure 11 to show the different behavior of each one of them.

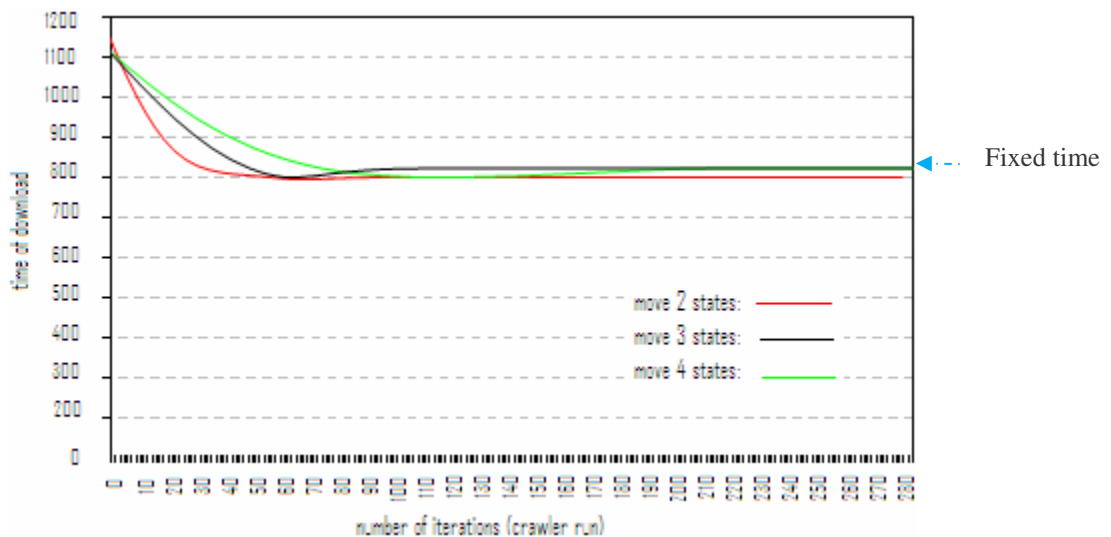


Figure11

The figure 11 shows that our proposition to move only 2 states has a behavior better than moving 3 or 4 states because it reaches the optimal solution in less number of iterations. This can be explained by the limited number of states used by the Automaton, 8 or 10 in each arm.

The figure 12 shows the number of webs sites (the objects) plays an important role to reach the optimal solution. Thus, the Automaton needs more iteration to attend the optimal solution when the numbers of sites (the objects) increases. That's why we think to increase also the number of access points (arms).

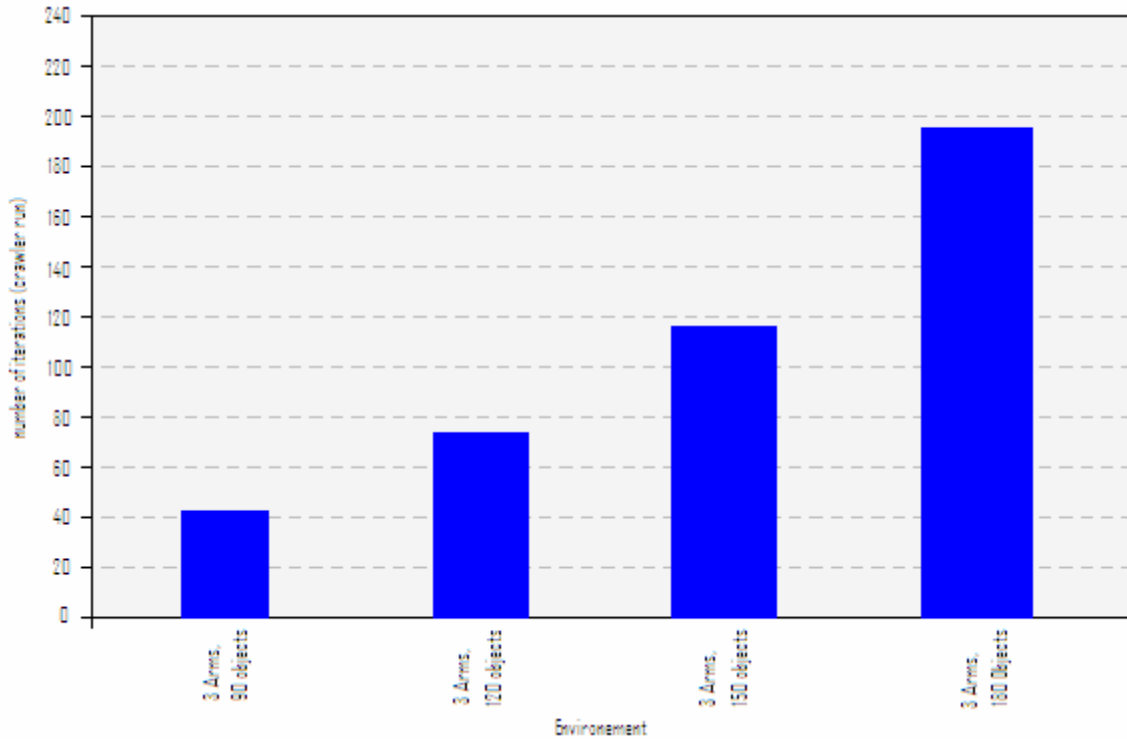


Figure 12

Another interesting result shown in figure 13 is that the numbers of access points (arms) plays a significant role to reach the optimal solution compared to the numbers of webs sites (the objects). The figure 13 shows the different behavior and the relationship between the objects and the corresponding arms.

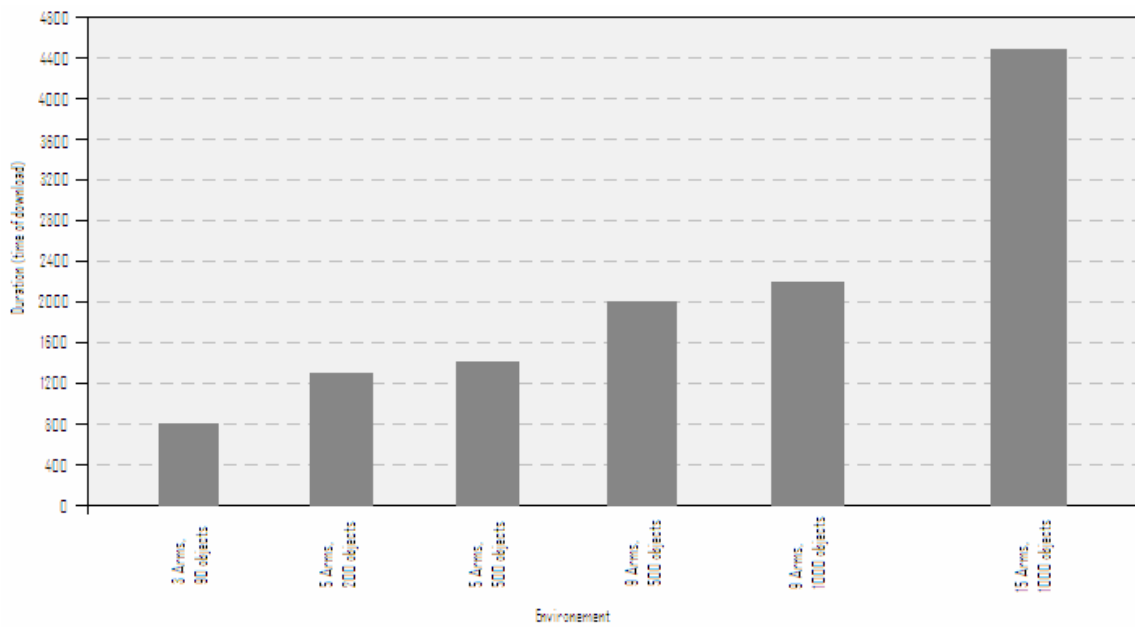


Figure 13

## 10. Discussion and Further work

Our initial task was to see whether the OMA was a good alternative to use in Distributed Resource Allocation Problem. We have found that it works very well to minimize the time of download with small number of web sites (objects), but not so good with a large number of web sites (objects). It needs a big number of iterations and more access points (arm) to reach the optimal solution.

We believe this can be explained by the explosive size of the Internet and in the same way the growth of users. That is why we think that we can take profit of our solution working very well with small number of web sites (objects). The new idea is to combine our solution with clustering approach. This makes it a lot easier to collect the large number of web sites (objects) in small set of cluster. After that we apply Object Migration Automaton (OMA) approach to the small number of web sites (objects). This is can be the first point of our further work.

Because the real web is dynamic, we will evaluate our algorithm in a dynamic environment. We test the behavior of our algorithm where most optimal arm for each object changed. This is the second point of our further work.

We have found also that Object Migration Automaton (OMA) is a good solution to improve a web crawler to detect the change of the web sites. Thus, if the web sites (objects) are distributed by the Object Migration Automaton (OMA) scheme, the crawler doesn't need to download all the web sites (object). However, it only goes for the accessed web sites in the Automaton that detect its changing. It remains then only to update these web sites (objects).

The big area of resource allocation problem leads us to think if we can use our solution in Telecom domain. It is interesting to handle frequencies allocation or channels allocation especially in GPRS and UMTS. This is the last point of our further work.

## 11. Conclusion

In this report, we have presented a working solution for Distributed Resource Allocation problem using the Object Migration Automaton (OMA). The Automaton is able to handle the web sites (objects), storing them in the system and making them moving around the automaton.

We have extended the Object Migration Automaton (OMA) and developed a new algorithm. Evaluating and experimental results have demonstrated that our solution can be used for handling the Resource Allocation Problem to minimize the time of download of the web sites (objects) and improving the web crawler to maximize the number of true download.

We also learned in the development process that for this type of solution depends of the number of web sites (objects) and also depends of the numbers of access points (arms). They play a significant role to reach the optimal solution compared to the numbers of webs sites (the objects).

The solution described has shown to handle a small number of web sites very well, but it has some problems when it comes to large number of web sites. Clustering is a proposed solution described in section 10 as a further work.

Our conclusion is that the Object Migration Automaton (OMA) is very suitable for Resource Allocation Problem. As it described in the problem description, Object Migration Automaton (OMA) is a good solution to handle the limited resources. It optimizes their allocation in web monitoring to be used in the best efficiency. Our successful experiments have shown that it was able to minimize the time of download and to detect the change of web sites.

We believe that OMA may turn out to be a valuable tool when it comes to combining our solution with clustering approach to handle a large number of web sites.

The last question that we can put it is: can we use our solution in Telecom domain to handle frequencies allocation and Channel allocation in GPRS and UMTS?

## Appendix

### **A1**            *Glossary & abbreviations*

Short definitions of terms used and references to further relevant glossary sources

### **A2**            *References*

[1] Morten Goodwin Olsen. Introduction to: Web Crawling Techniques and Resource Allocation

[2] B. J. Oommen, D. C. Y. Ma. Fast Object Partitioning Using Stochastic Learning Automaton.

[3] B. J. Oommen D. C. Y. Ma Deterministic Learning Automata Solutions to the Equipartitioning Problem.

[4] Leiming Chen, Dongmei Wang, Morten Goodwin Olsen. Towards distribution of web sites in a crawler used for large scale web accessibility assessment.

[5] M. Snaprud, M. G. Olsen, and F. Aslaksen. Automatic benchmarking and presentation of the first results from the european Internet accessibility observatory," eChallenges ETSI, 2006.

[6] Vladislav Shkapenyuk, Torsten Sue. Design and Implementation of a High Performance Distributed Web Crawler.

[7] Ole-Christoffer Granmo, B. John Oommen, Svein A. Myrer, Morten G. Olsen. Determining Optimal Polling Frequency Using a Learning Automata-based Solution to the Fractional Knapsack Problem.

[8] J.L. Wolf, M.S. Squillante, P.S. Yu, J. Sethuraman, L. Ozsen. Optimal Crawling Strategies for Web Search Engines

[9] Ole-Christoffer Granmoy, B. John Oommen, Svein Arild Myrer, Morten Goodwin Olsen Learning Automata-based Solutions to the Nonlinear Fractional Knapsack Problem with Applications to Optimal Resource Allocation

[10] Junghoo Cho, Alexandros Ntoulas Effective Change Detection Using Sampling.

[11] Junghoo Cho, Hector Garcia-Molina Estimating Frequency of Change.