

Distributed Resource Allocation Algorithm

IKT 404
SPRING 2006



HØGSKOLEN I AGDER

Agder University College
Faculty of Engineering and
Science

Author(s):

Leiming Chen
Dongmei Wang

Supervisor(s):

Morten Goodwin Olsen

Version: 1

Pages: 29 (including this page)

Modified date: 2006-05-28

Status: Final

Keywords:

OMA

Web crawler

Objects

Optimal
situation

Resource
allocation

Abstract:

This report describes the using of OMA (a kind of learning automata) in the web accessing can perform better than the uniform web accessing algorithm (round robin). The web crawlers are all simulated in the local machine with the same number of states and objects, so it can only shows the ideal situation. However the results that we have got have a strong evidence for proving that the OMA algorithm works really efficient in the web accessing.

Executive Summary

In this report we present a method for accessing web sites shown to be more optimal than the much used uniform distribution. Using this method, we can easily optimize the paths for accessing for many computers ideally.

In order to do this we tried to simulate the web crawlers as arms for computers to access different websites. Arms means web crawlers that can be used for client computers to connect to the destination source object. Of course, here we can easily change the number of arms and the number of states in different arms.

In our program, in order to make it easier to implement and debug, we made all web crawlers have the same states and objects connection. That means we must force the program to choose one perfect arm for us to make the OMA algorithm work. The perfect arm means by this way we can access the wanted objects by the shortest time. Later in our progress, we also made the perfect arm change as time changes in order to simulate the really situation in which the working situation is not stable at all according to many factors.

In the program, we divide the situation into three different conditions. One in old way of objects accessing, we called it uniform, one in the OMA static way and the last is in the OMA dynamic way. We have done the tests for getting the results and the expected result.

However, because we put all the crawlers with same number of states and objects and all the objects are locally simulated which is not the realistic condition, these test results can only show the ideal situation of different conditions. That means all the crawlers can get contact with all the websites in the world and have the same situation no matter where you access the crawler.

Now, this method and the results show that the OMA algorithm works quite well in the remote objects accessing. That indicates that the OMA algorithm has a prosperous future in web crawler use.

Index

EXECUTIVE SUMMARY.....	1
INDEX.....	2
2 INTRODUCTION.....	3
2.1 PROJECT OUTLINE.....	3
2.2 THE PROJECTS CONTRIBUTION TO THE WEB ACCESSIBILITY	4
3 BACKGROUND (REVIEW OF LITERATURE).....	5
3.1 WEB CRAWLER.....	5
3.2 LEARNING PROCESS OF AN AUTOMATON.....	6
3.3 OBJECT MIGRATION AUTOMATON (OMA).....	7
4 PROBLEM DESCRIPTION.....	10
5 SOLUTION.....	12
5.1 REQUIREMENTS FOR THE ALGORITHM.....	12
5.2 IMPLEMENTATION	13
5.3 TESTING AND VALIDATION.....	15
6 DISCUSSION.....	24
6.1 THE PROJECT OUTCOME.....	24
6.2 EVALUATION OF THE TEST RESULT.....	24
7 FUTURE WORKS FOR THIS PROJECT.....	25
8 CONCLUSION.....	26
APPENDIX.....	27
A1 GLOSSARY & ABBREVIATIONS.....	27
A2 REFERENCES.....	28

2 Introduction

In today's modern society a lot of time is spent searching information on the web. Web can be viewed as a huge distributed system consisting of billions of web pages located at various sites. The main part of any mechanism used for large scale web assessment tool, e.g. as a web accessibility measurement tool or a search engine (as Google), will be a web crawler, possibly distributed. A web crawler (also known as a web spider or web robot) is a program which browses the Web in a methodical, automated manner. It is highly desirable that the crawler can download as many updated web pages as possible compared to the locally stored copies. This is a solution to the resource allocation problem where the resources have an unknown value by using a competitive game of learning automata. It seems like a viable approach if the distributed crawler can access the web pages needed as soon as possible. If the web pages to download are partitioned on the same group then the web crawler can access them as soon as possible. This is a starting point to improve the throughput of the web crawler. We have in this project solved this problem by using an object partitioning algorithm, namely by using Object Migration Automaton (OMA). We have further extended the OMA to Time Weighted Object Migration Automaton (TWOMA) to work as a distributed resource allocation algorithm.

2.1 Project Outline

2.1.1 Goal of the project

The goal of the project is to give a solution to the distributed resource allocation and scheduling by using Object Migration Automaton (OMA). In fact we extended the OMA algorithm to a new algorithm TWOMA.

2.1.2 What was done and how

In order to solve the problem, first we simulated some distributed crawlers and define several different states with different return time to simulate the situation from the optimal to the worst. Second, we simulated some objects lied in the different crawlers. These objects were the access point for the crawlers. According to the states and the location of the objects, we

simulated the ideographic migration of every pair of the objects. Finally we took a lot of test for the algorithm to show its functionality.

2.2 The projects contribution to the web accessibility

In [5], the contribution of this project to the web accessibility is mentioned. A mechanisms used for large scale accessibility measuring may involve a distributed web crawler. Furthermore, it makes sense to spread the web sites involved to different access points of the distributed crawler. The algorithm we developed in this project, namely TWOMA, utilizes the available resources to a much greater extent than the traditional uniform distribution of web sites. The heart of the algorithm involves continuously accessing web sites in pairs while measuring the duration of each access. The algorithm is repeated as long as the crawling / measurement is ongoing. This ensures that the schema works in a dynamic environment (as the real web).

2.3 Report Outline

This report has 8 chapters, including the summary, introduction and conclusion.

First, some related knowledge and background for this project will be introduced, including a briefly description of web crawler, learning process of automata and the algorithm OMA, in chapter 3. Then a problem description is provided in chapter 4. Chapter 5 is the solution for this project. This is the main part of the report including the requirement of this project, the implementation of this project and the test/validation of this project. Some discussion of the finding for this project is given in chapter 6. Chapter 7 discusses the possibility for the further work on the project. The conclusion for this project follows in chapter 8. In the appendix, a glossary and references list can be found.

3 Background (Review of literature)

In this section we will look at the relevant ideas behind this project. First, we will give a brief introduction of web crawler since two web crawling algorithms are simulated. Second, we will introduce some fundamental knowledge of learning automata. Last, Object Migration Automaton (OMA), which is one theoretical stochastic learning Automaton solutions for the Object Partitioning Problem (OPP), will be discussed. It demonstrated an excellent partitioning capability, and experimentally, this solution converges an order of magnitude faster than the best known algorithm in the literature [2]. In this project we tried to partition the accessed objects in an optimal way and OMA is a viable approach to solve the problem.

3.1 Web Crawler

Due to web pages' explosion in size, web search engines are becoming increasingly important as the primary means of locating relevant information. Such search engines rely on massive collections of web pages that are acquired with the help of *web crawlers*, which traverse the web by following hyperlinks and storing downloaded pages in a large database that is later indexed for efficient execution of user queries [9]. Thus, highly efficient crawling systems are needed in order to download the hundreds of millions of web pages indexed by the major search engines.

Two issues are addressed for a good crawler for a large search engine in [9]. First, it has to have a good crawling strategy, i.e., a good strategy for deciding which pages to download next. Second, it needs to have a highly optimized system architecture that can download a large number of pages per second while being robust against crashes, manageable, and considerate of resources and web servers.

3.2 Learning process of an automaton

In [1] B.J. Oommen and D.C.Y Ma gave us some idea about learning process of an automaton. The learning process of an automaton can be described as follows. The automaton is offered a set of action by the environment with which it interacts, and it is constrained to choose one of these actions. When an action is chosen, the automaton is either rewarded or penalized by the environment with a certain probability. A learning automaton is one which learns the optimal action. This means that the automaton learns the action which has the minimum penalty probability, and eventually chooses this action more frequently than other actions. In short learning process is defined as the ability to improve ones ability based on experience from previous behaviour.

In the literature the automaton is defined by a quintuple $(\alpha, \Phi, \beta, F, G)$ where

- a) $\alpha = \{ \alpha_1, \dots, \alpha_R \}$ denote the set of actions that can be produced by the automaton. This is the output set of the automaton, hence also being the input set to the environment.
- b) $\Phi = \{ \phi_1, \dots, \phi_N \}$ is the set of internal states of the automaton. The states determine the action to be produced by the automaton.
- c) $\beta = \{0,1\}$ is the input set to the automation and is the set of response from the environment. The input "1" represents a penalty.
- d) F is a map from $\Phi \times \beta$ to Φ . It is a function that maps the current state and the response from the environment into next state. It defines the transition of the internal state of the automaton on receiving an input. F may be stochastic.
- e) G is a map from Φ to α , and determines the action taken by the automaton if it is in a given state.

The automaton is showed in Figure

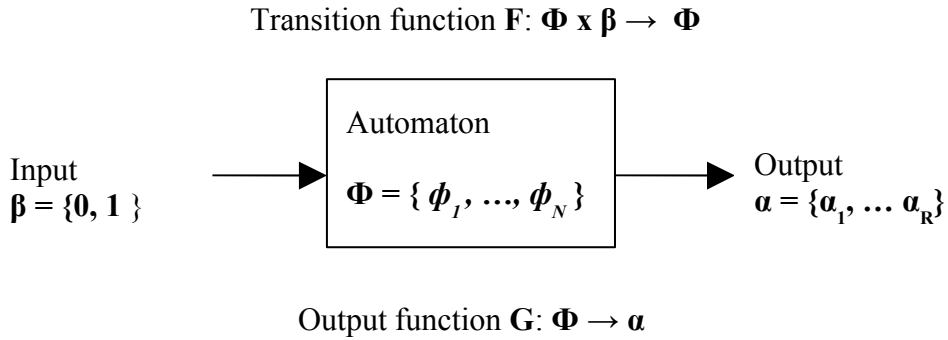


Figure 3.2-1 Learning Process of an Automaton

3.3 Object Migration Automaton (OMA)

In this project OMA is used to grouping the accessed objects and allocated them into an optimal way. By this way the web crawler can download the web pages as much as possible.

In [1], B.J.Oommen gave us a detailed description about OMA. We define the object migrating automaton (OMA) as a quintuple in which there are only R actions, each action representing a certain class. Furthermore, for each action, there are a fixed number of states N. If the abstract object O_i is in action α_k (we say the action is one arm of the automaton), this is to signify that this abstract object will be in the class numbered k . Thus, if O_i and O_j are in the same class and the query requests (A_i, A_j) , the OMA is rewarded. Otherwise, it is penalized. It means that if all the objects are in the same class, the OMA will obviously never be penalized, and so the automaton will tend to converge to such an arrangement. Clearly this is undesirable and so this scenario will have to be explicitly prohibited. We shall show below how this is done.

For each action α_k , there is a set of states $\{\phi_{k1}, \dots, \phi_{kN}\}$, where N is the depth of memory and $1 \leq k \leq R$, the number of classed (partitions) desired. With no loss of generality, we assume ϕ_{k1} to be the most internal state of that action and ϕ_{kN} to be the boundary state. When a pair of physical objects (A_i, A_j) are accessed, if the *abstract* objects O_i and O_j are in the same class α_k (they are in the same arm) both of them are rewarded and are moved toward the most internal state, ϕ_{k1} , of that action, one step at a time. See Figure 3.3-1 (a). If, however, the abstract objects lie in different classes (different arms), say α_k and α_m , respectively, (i.e., O_i is in state ξ_i , where $\xi_i \in \{\phi_{k1}, \dots, \phi_{kN}\}$, and O_j is in state ξ_j , where $\xi_j \in \{\phi_{m1}, \dots, \phi_{mN}\}$), they are moved away from ϕ_{k1} and ϕ_{m1} as follows.

a) If $\xi_i \neq \phi_{kN}$ and $\xi_j \neq \phi_{mN}$, then move O_i and O_j one state toward ϕ_{kN} and ϕ_{mN} respectively. See Figure 3.3-1. (b).

b) If exactly one is in the boundary state, (i.e., either $\xi_i = \phi_{kN}$ or $\xi_j = \phi_{mN}$ but not both), then move the object which is *not* in the boundary state towards its boundary state. See Figure 3.3-1. (c)

c) If both are in the boundary state, (i.e., $\xi_i = \phi_{kN}$ and $\xi_j = \phi_{mN}$), then one of them, say O_i will be moved to the boundary state of α_m . In this case, since this will result in an excess of objects in α_m , one of the objects in α_m which is *not* accessed is moved to the boundary state of α_k . We choose to move the one closest to ξ_j . See Figure 3.3-1. (d)

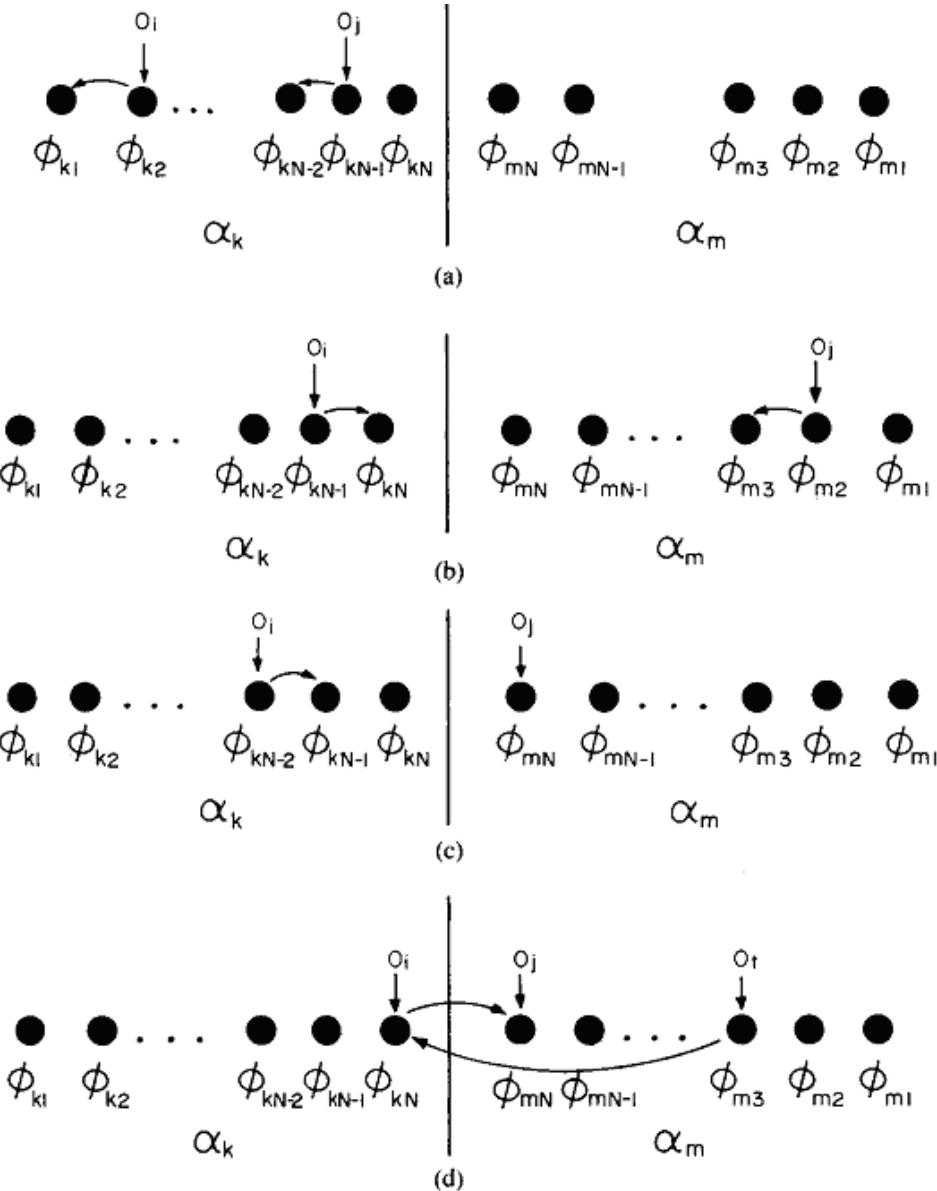


Figure 3.3-1 from [1] The object moving automaton:

On reward move the abstract objects to the most internal state for that action as in (a).
On penalty, move the abstract objects either toward the boundary state as in (b) or (c),
or toward the boundary state of another action (d).

4 Problem description

A solution to the resource allocation problem where the resources have an unknown value can be solved using a competitive game of learning automata. Many distributed crawlers uniformly distribute the web sites between its access points. This means that each access point (or crawler arm) get an equal amount of web sites. Some arms may get only web sites which take a long time to download, while others may get web sites which only take a short time to download. From discussion in [4], the resource with an unknown value can be seen as that it is no way of knowing which web sites are quick, which arms has a lot of room and how the web sites behave from different access points before they actually started to download there sites. This problem can be seen as a web crawler attempting to detect and download as many updated web pages as possible compared to the locally stored copies. This project aims on solving the same problem when the resources are distributed, and the resources may have different value depending on its position.

This problem can be seen as a distributed crawler working towards a solution on how to partition web sites to download. Using the Object Migration Automaton (OMA) towards partitioning the web sites to receive the most viable solution seems like a viable approach. This project focuses on not only implementing TWOMA in a test environment, but also providing experimental test to show that TWOMA algorithm utilizes the available resources to a much greater extent than the traditional uniform distribution of web sites.

5 Solution

5.1 Requirements for the algorithm

In order to implement the algorithm, we must predefine a few requirements for its functionality.

- **Web sites:** We need to simulate a lot of web sites. This is the objects to be accessed in the algorithm. They map to abstract objects in the OMA algorithm. They play a ‘passive object’ role in the algorithm. These simulated web sites must initially be randomly distributed on the different location. And the numbers of the web sites should be flexible in order to be more close the real World Wide Web.
- **Web Crawler:** Web crawler is needed to simulate too. Web crawler plays a ‘user’ role in the algorithm. It takes charge of accessing the web sites and it will get the download time. The download time is the most important factor to show if the algorithm is valuable. The implementation of the simulated web crawler should be responsible for giving feedback to the implemented algorithm, which includes the download time.
- **Implementation of OMA:** After simulating the web crawler and web sites, we should implement the algorithm OMA. The OMA is a fundamental algorithm to TWOMA. The OMA plays a ‘control mechanism’ role in the algorithm. By using OMA algorithm, the web sites (i.e. objects) can be moved to the more optimal location. Hence the download time of these web sites can decrease dramatically.
- **Flexibility of arms:** As we has discussed before, arms are the visual name for the web crawler’s accessing points. Here we assume that there are several accessing points in the real World Wide Web depending on location, for example, USA and Europe are two accessing points. All of web sites lie on one accessing point (i.e. one arm). Where is the web site lying is one of conditions for the OMA to take its action moving the object. In order to simulate a dynamic environment, the arms of the crawler must be flexible.

5.2 Implementation

In order to simulate the OMA resource allocation in java programs, we did as follows:
First, we simulated the crawlers (means the arms in the OMA algorithm).

- Build a class to simulate the total crawler which we can access and get return time.
- Define several states to simulate the states in the real crawler.
- Set the states with different return time that can differ from the optimal to the worst.

Second, we simulated the objects allocated in the crawlers.

- Build a class to simulate the objects lied in the different crawlers.
- Define the objects first lie in the worst states of the crawlers.
- Define the showing state of the objects' current states.
- Define which crawler the objects belong to.

Third, we simulated the ideographic migration of every pair of the objects.

- Define several crawlers with a certain number of states; here we can change the number of states and the number of crawlers.
- Define the number of the objects and the crawlers they belong to.
- Define the situation which the two objects belong to the same crawler, and move the objects to the most optimal states.
- Define the situation which the two objects belong to different crawlers, and move the objects according to the OMA algorithm until they both move to the optimal state. Generally speaking, they lie in the same crawler.

Finally, we simulate the real situation that many objects lie in the different crawlers and all in the worst states.

- Every time, access the objects and get the return time.
- Eventually, get to the optimal states of the objects and quit.

Figure 5.2-1 shows how the algorithm works:

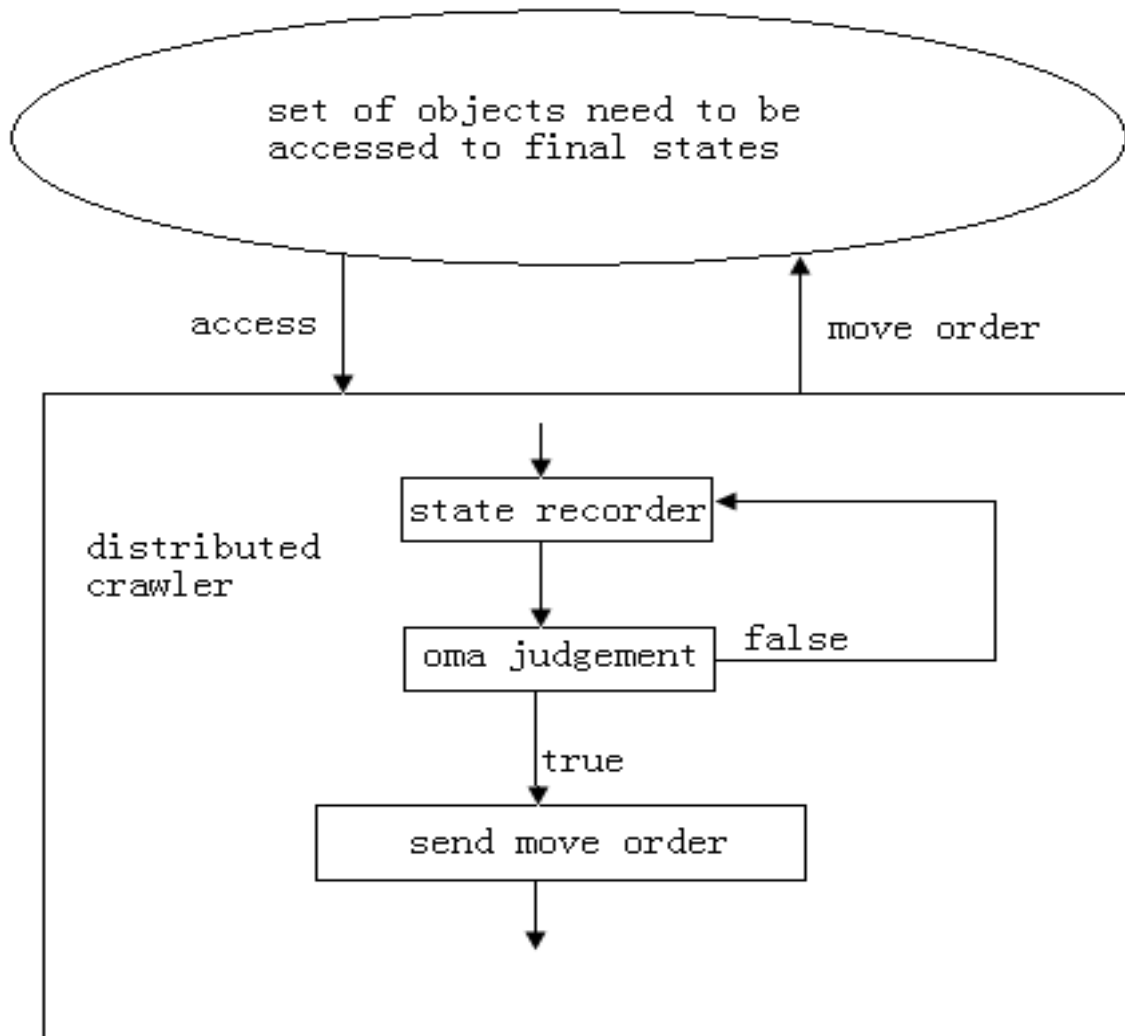


Figure 5.2-1 The diagram to show how the algorithm works

5.3 Testing and validation

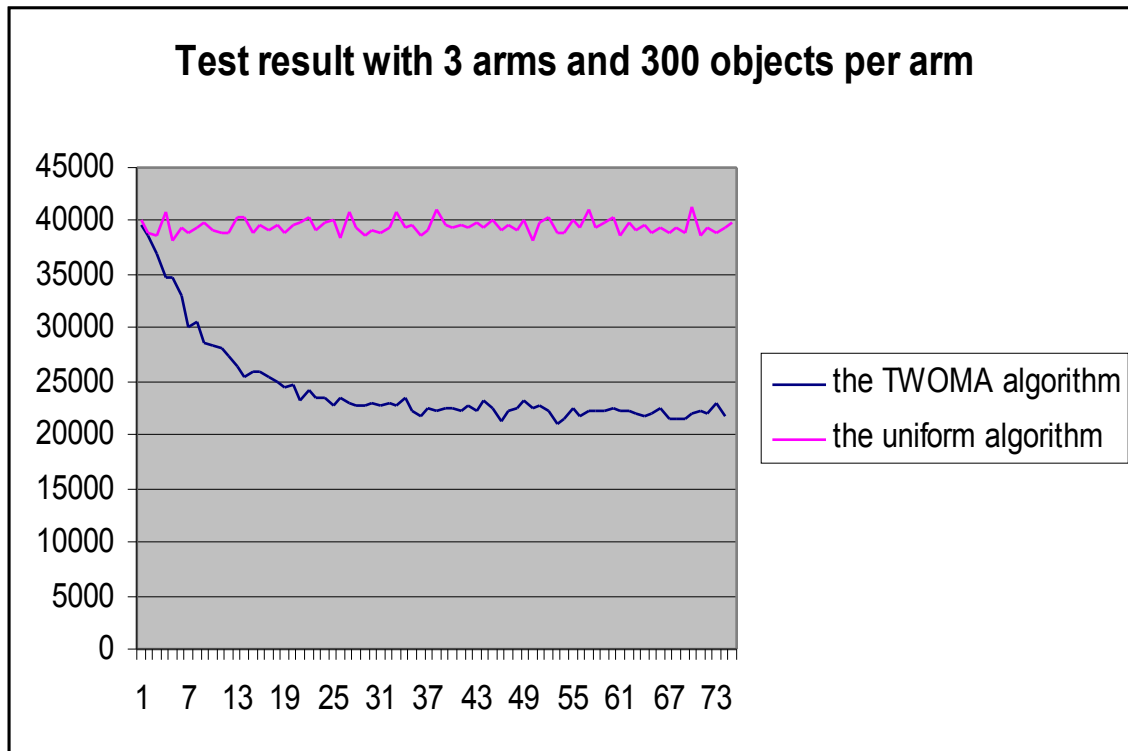
We did a lot of experiments to test how the algorithm works and if it could satisfy our requirement. Most of these experiments showed that the algorithm worked very well comparing with the uniform algorithm. Further testing also showed that the algorithm worked well in the dynamic environment. But one of the experiments had a little trouble. When we increased the number of the arms to 30 with 100000 objects per arms, our personal laptop seemed having no ability to execute the algorithm with this large amount of objects, which we believe is due to processing limitation of our laptop.

The method we took for the experiments were to define a set of variable numbers of arms and objects. With the same condition of the arms and objects, we execute the algorithm in two different ways respectively, namely with or without the algorithm. We got total different download time by these two ways.

After the testing in a static environment, we tried to do some test for the algorithm in a dynamic environment. The results showed the algorithm worked well in the dynamic environment also.

The following figures show some test result:

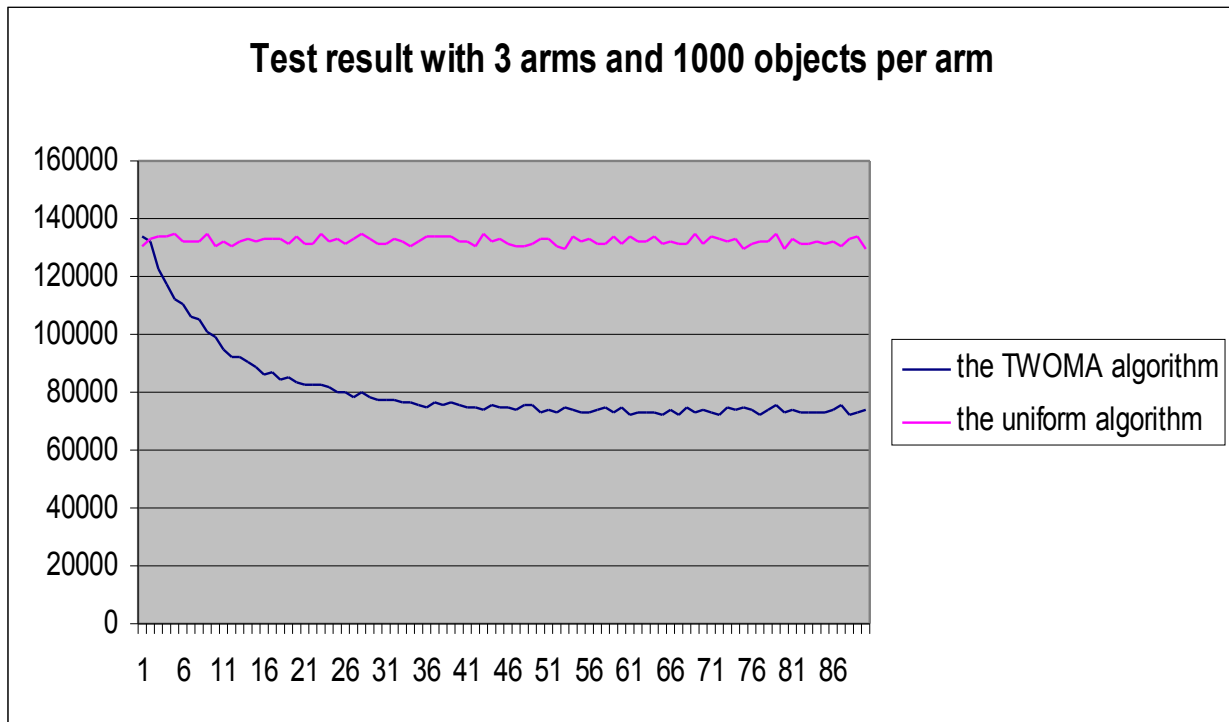
Figure 5.3-1 Test Result with 3 Arms and 300 Objects per Arm



We begin our test from setting 3 arms with 300 objects per arm to the algorithm. Figure 5.3-1 shows that in order to access these $300 \times 3 = 900$ objects, the uniform algorithm used about 40000 time units. The time the uniform algorithm used does not change very much. For the algorithm TWOMA, it used almost same time units as the uniform algorithm at the beginning. But with more and more the objects migration, the time used decreased and the TWOMA reached the optimal situation. After migrating about 25 times the algorithm TWOMA is convergent at 22000 time units. It means that the efficiency of the algorithm TWOMA is about two times of the uniform algorithm.

The algorithm TWOMA takes 74 crawler duration migration to reach the optimal situation in this situation. A crawler duration includes accessing and downloading each website within each arm.

Figure 5.3-2 Test Result with 3 Arms and 1000 Objects per Arm



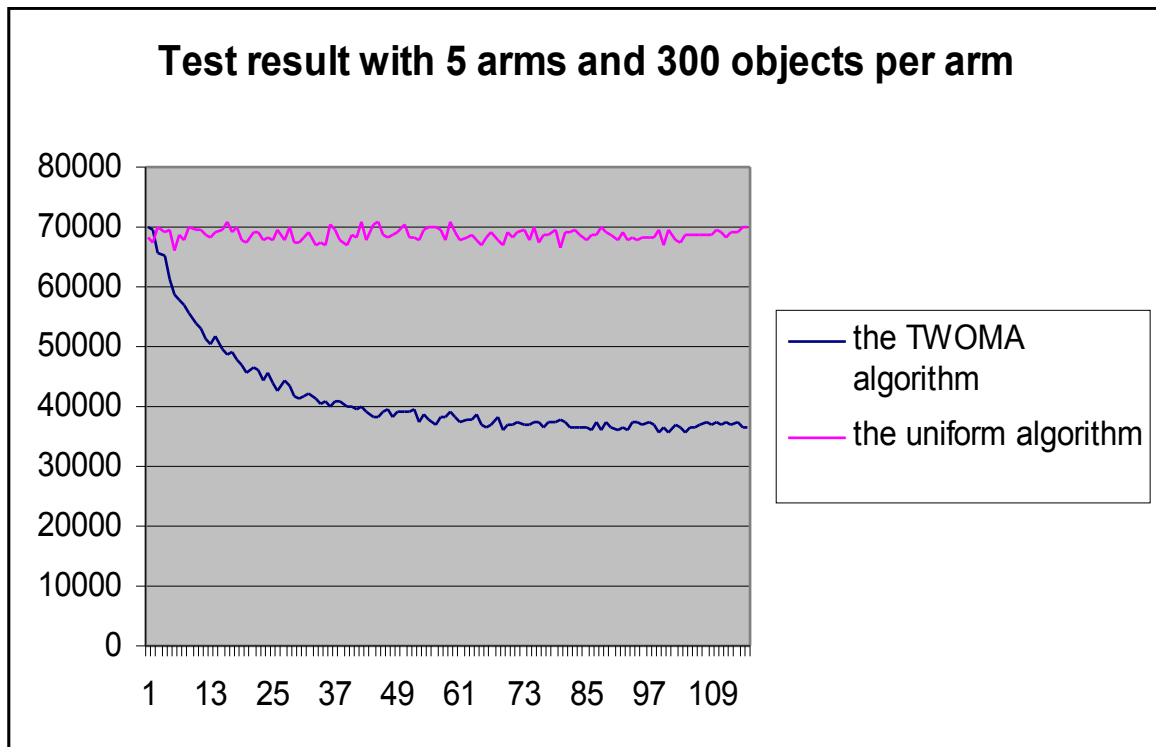
The second experiment was taken in the situation with 3 arms and 1000 objects per arm to the algorithm. In this situation we increased the numbers of the objects while the numbers of the arms was same as the Figure 5.3-1. The total objects increased to $1000 \times 3 = 3000$. Figure 5.3-2 shows that in order to access these 3000 objects, the uniform algorithm used more than 130000 time units and converge at this point.

As same as Figure 5.3-1, the algorithm TWOMA used almost same time units as the uniform algorithm at the beginning. But with increasing the objects migration to about 40 times, the TWOMA converged at about 75000 time units.

Comparing with the test result from Figure 5.3-1, Figure 5.3-2 shows that with the same numbers of the arms, the download time is proportional to the numbers of the objects.

The algorithm TWOMA takes 90 crawler duration migration to reach the optimal situation in this situation, 3 arms with 1000 objects per arm.

Figure 5.3-3 Test Result with 5 Arms and 300 Objects per Arm

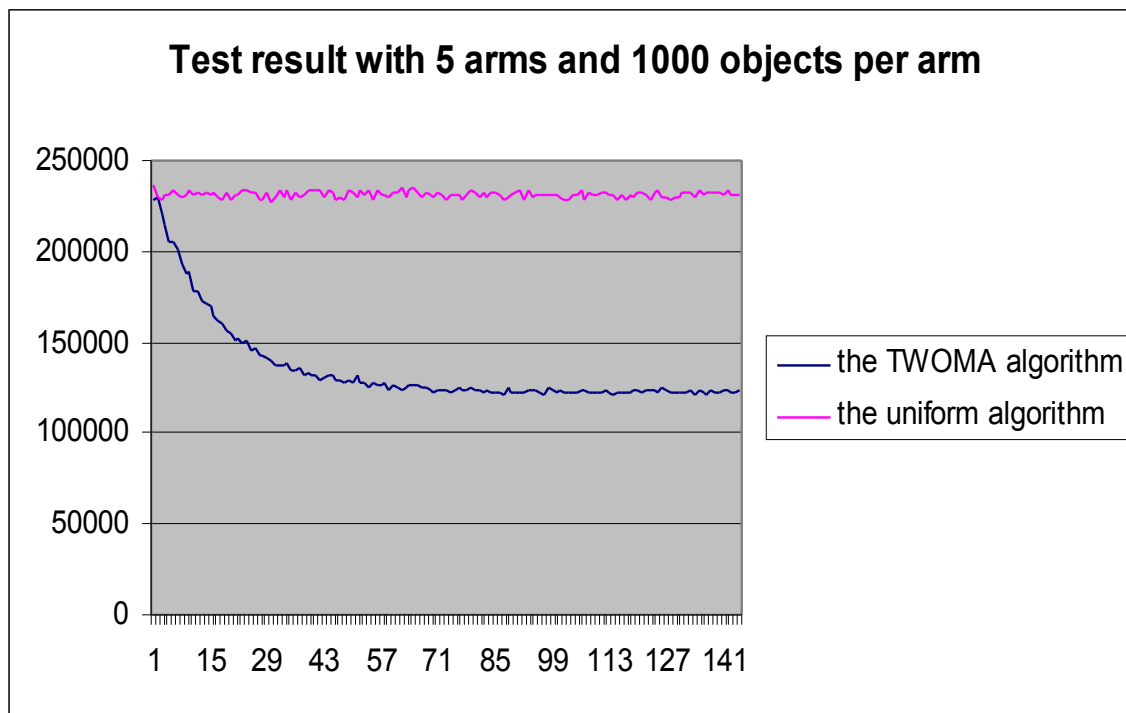


This experiment was taken in the situation with 5 arms and 300 objects per arm to the algorithm. In this situation we increased the numbers of the arms to 5 while the numbers of the objects was decreased to 300. The total objects increased to $300 \times 5 = 1500$. We will compare the result of this experiment with the other two previous results.

Figure 5.3-3 shows that for the uniform algorithm the increasing download time is proportional to the increasing of numbers of the total objects.

For the algorithm TWOMA the increasing download time is proportional to the increasing of numbers of the total objects also. There is another factor is of interest to us, namely the crawler duration to the optimal situation. In this situation the algorithm TWOMA takes 116 times migration to reach the optimal situation. In the result of Figure 5.3-2 (3 arms with 1000 objects per arm, total objects is 3000) showed, the crawler duration migration is 90. In the result Figure 5.3-3 shows the crawler duration migration is 116 although the numbers of total objects is less than those in Figure 5.3-2. The main reason for this phenomenon is the increasing numbers of the arms.

Figure 5.3-4 Test Result with 5 Arms and 1000 Objects per Arm

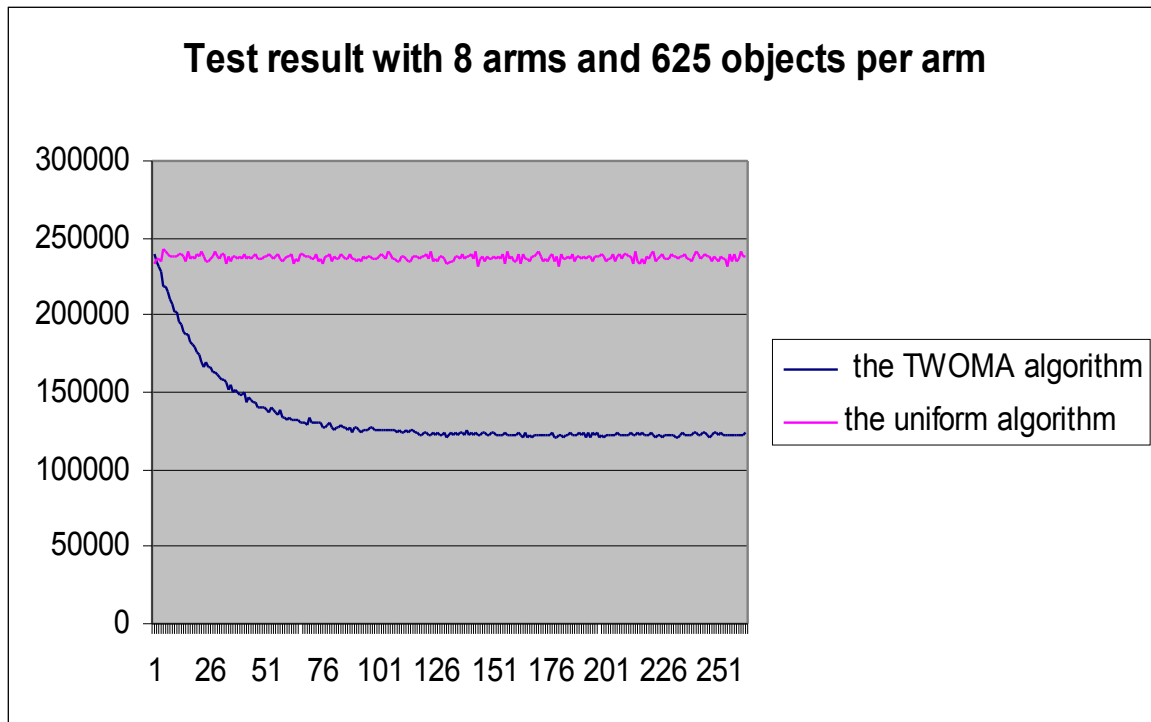


This experiment was taken in the situation with 5 arms and 1000 objects per arm to the algorithm. In this situation we increased the numbers of the objects to 1000 while the numbers of the arms keeps same with the test in Figure 5.3-3. The total objects here is $1000 * 5 = 5000$.

As all the previous experiment, Figure 5.3-4 shows that the algorithm TWOMA has higher efficiency than the uniform algorithm. For both algorithms, the increasing download time is proportional to the increasing of numbers of the total objects.

In this situation the algorithm TWOMA takes crawler duration migration to reach the optimal situation.

Figure 5.3-5 Test Result with 8 Arms and 625 Objects per Arm

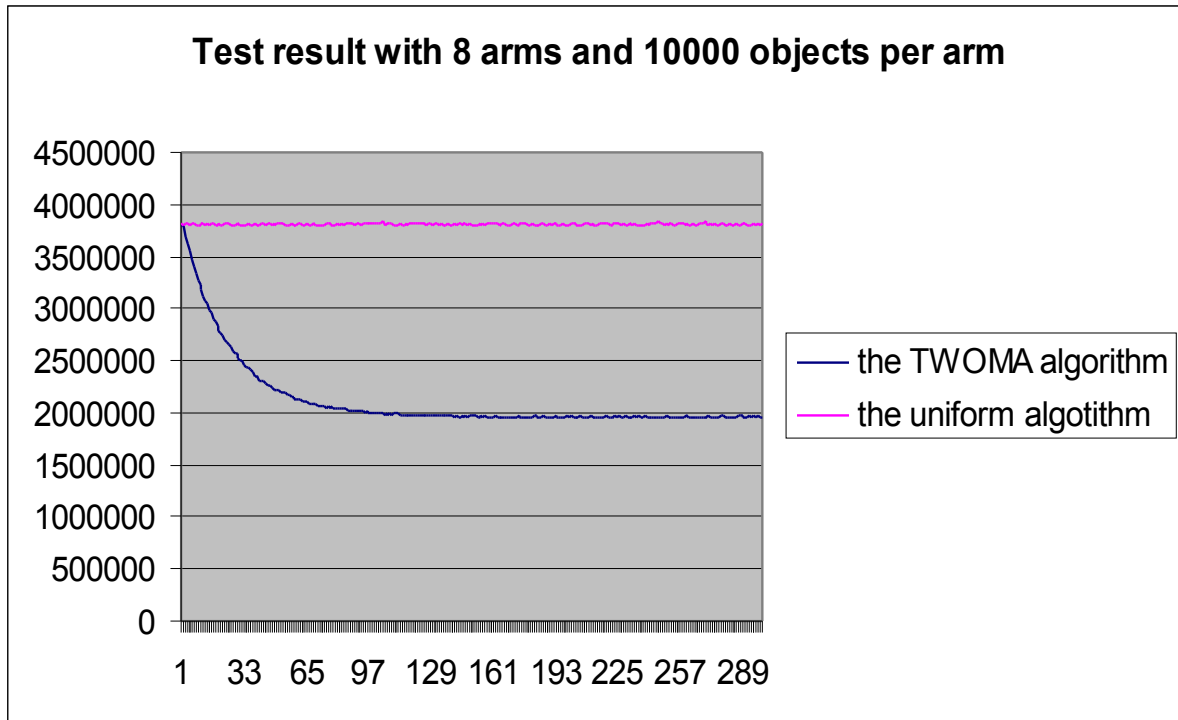


This experiment was taken in the situation with 8 arms and 625 objects per arm to the algorithm. The total objects here is $625 \times 8 = 5000$, which was same as those in test Figure 5.3-4 showed. However, the numbers of the arms increased from 5 to 8. We did this experiment in such situation in order to see which role the numbers of arms played in the algorithm.

In this situation the algorithm TWOMA takes 263 crawler duration migration to reach the optimal situation.

Figure 5.3-5 shows that the algorithm TWOMA has higher efficiency than the uniform algorithm. The download time is same as those in the test which has the same total objects 5000. But the crawler duration migration to the optimal situation increases dramatically, from 145 to 263, mapping the increasing arms from 5 to 8. This test shows again that the numbers of arms play a very important role in the algorithm.

Figure 5.3-6 Test Result with 8 Arms and 10000 Objects per Arm



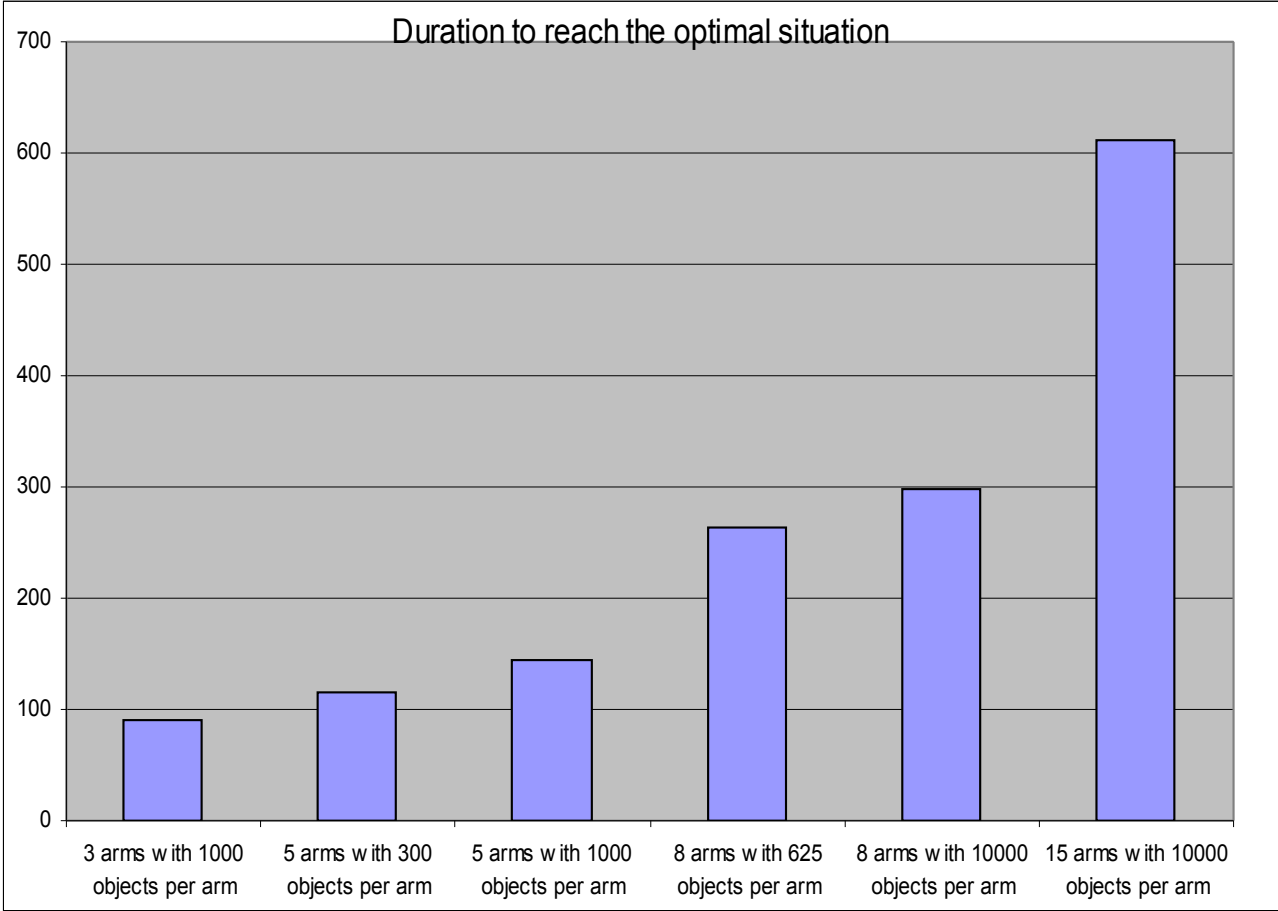
This experiment was taken in the situation with 8 arms and 10000 objects per arm to the algorithm. The total objects here is $10000 \times 8 = 80000$. We did this experiment in such situation in order to how the algorithm worked with many objects.

In this situation the algorithm TWOMA takes 299 crawler duration migration to reach the optimal situation.

Figure 5.3-5 shows that the algorithm TWOMA also has higher efficiency than the uniform algorithm in the situation with a lot of objects. This test has the practical meaning because the real World Wide Web contains billions of web pages. It means that the algorithm can be practiced in the real world.

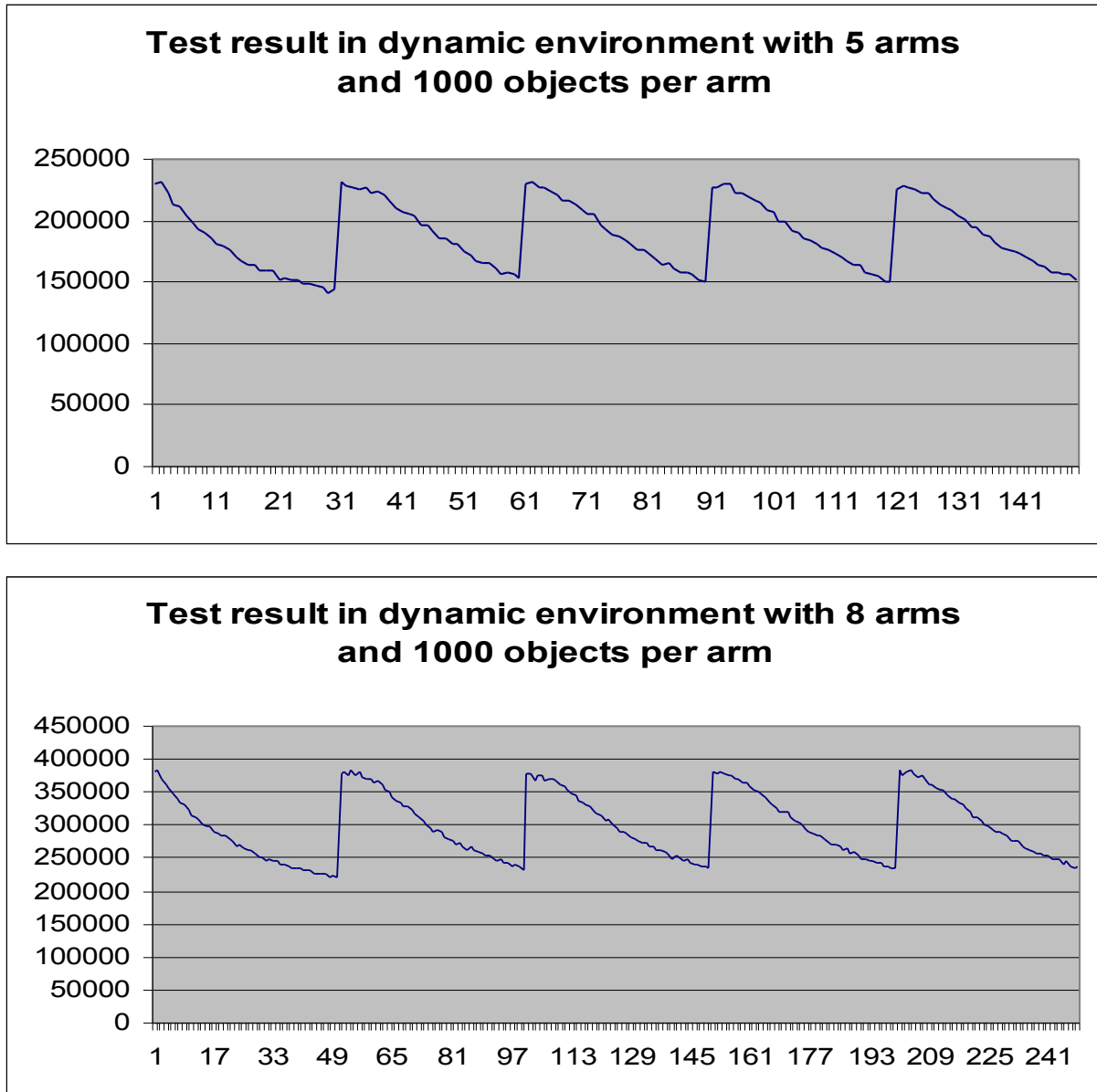
We have done a test with 15 arms and 10000 objects per arm. The result was well also.

Figure 5.3-7 Duration to Reach the Optimal Situation



After doing a lot of experiments we selected some test results to see how about the duration to reach the optimal situation was. Figure 5.3-7 shows the different duration to reach the optimal situation in some test situation. The figure shows that the numbers of the arms plays a more important role than the numbers of the objects if it is concerning to the duration to reach the optimal situation in the test. From the third bar (5 arms-1000 objects/arm) and the fourth bar (8 arms-625 objects/arm) in Figure 5.3-7, we can find that the duration to reach the optimal situation increase dramatically if the numbers of the arms increase. We have noticed that in these two cases the numbers of total objects are same. Comparing the two bars (8 arms-10000 objects/arm, 15 arms-10000 objects/arm) on the right side of Figure 5.3-7, we can find the same conclusion.

Figure 5.3-8 Test Result in Dynamic Environment



In order to increase the practical meaning of the algorithm TWOMA, we did some experiments for the algorithm in the dynamic environment. The test result in Figure 5.3-8 shows also that the algorithm can work well in the dynamic environment. Dynamic environment means that when the objects are trying to reaching to their optimal situation, the perfect arm state can be changed. In other words, the optimal location for two accessed web sites can be dynamic in the real world. The above figures show the optimal state changing at each 30th and 50th crawler duration. Note that the implementation is still able to reach close to optimal states within the limited time period.

6 Discussion

In this section we will discuss some findings for this project.

6.1 *The project outcome*

The goal of this project is to find an algorithm for the distributed resource allocation. The outcome of the project satisfies this initial purpose as the test result in chapter 5 has showed. The outcome of this project also shows that the algorithm OMA is a viable approach to solve such problem. The algorithm TWOMA differs from OMA that it contains the download time to access the pair's objects. And we have showed that TWOMA can work both static and dynamic well.

As we have mentioned in the chapter 2, the algorithm TWOMA has a practical meaning for the web accessibility. The contribution to the web accessibility gives the algorithm TWOMA more interest to be researched in the future.

6.2 *Evaluation of the test result*

Chapter 5 has showed some test result. The overall result shows that the algorithm works very well in relation to the uniform algorithm in both static and dynamic environment.

The trouble we have met during testing was when we set the numbers of arms to 30 with 100000 objects per arm. Our laptop seemed to have no ability to execute the algorithm. As we had taken the experiments successfully many times, we assumed the failure with 30 arms/100000 objects per arm was because of the computing ability of our laptop.

The interesting factor in the test result was the numbers of the arms. As we have showed in Figure 5.3-4 and Figure 5.3-5, although the numbers of the total objects are same in these two tests, the duration to the optimal situation increased dramatically when the numbers of the arms increased 3. A question could arise here: is there any relation between the numbers of arms and the duration to the optimal situation? Unfortunately, the test result of this project was too limited to get a conclusion.

7 Future works for this project

Here some possibilities and necessities are discussed if someone will continue further work for this project.

Web Crawler: One of the important further works is concerning improving the web crawlers. In this project, we simulated the web crawlers in a very simple way. Of course it was practical because of the limited work time. If we could use more powerful and robust crawler to execute the objects migration we could find more meaningful result from the test. And we can be more closely with the real world instead of simple simulation.

Web Pages: With the same reason as web crawler mentioned above, we have not accessed the real web pages. The uncertainty of the web crawlers and the web pages could have some influence on the algorithm.

Dynamic environment: The dynamic property in the real world is more complex than we assumed in the algorithm. How to make the simulation of the dynamic environment better is also a critical problem.

8 Conclusion

In this report we have presented an algorithm, namely TWOMA, to solve the problem of the distributed resource allocation. In order to implement the algorithm, the web crawlers and the web pages were simulated. The test result has demonstrated that the algorithm can work very well compared to the uniform algorithm. And it works well in the dynamic environment. Comparing the uniform algorithm, the access time for the web pages by using the algorithm TWOMA can be decreased dramatically by means of migration. This contribution can strengthen the interest of further research to this project.

From this project we find that the OMA is very suitable for solving the partition problem. In fact there are some reference have discussed it. But we actually understood and practiced it in this project. As an extension of the algorithm OMA, the algorithm TWOMA keeps the properties of the OMA along with containing the download time.

Concerning this project and its further work, it is important to make the crawlers and the web pages (objects) more powerful and more realistic. This can make the algorithm more robust and practical.

Appendix

A1 Glossary & abbreviations

A1.1 OMA

Object Migration Automaton

A1.2 OPP

Object Partitioning Problem

A1.3 TMOMA

Time Weighted Object Migration Automaton

A2 References

[1] B. J. Oommen, D. C. Y. Ma

Deterministic Learning Automata Solutions to the Equipartitioning Problem

[2] Yiming Yang, Jaime Carbonell, Ralf Brown, Tom Pierce, Brian T. Archibald, Xin Liu.

Learning approaches for Detecting and Tracking News Events.

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

[4] Ole-Christoffer Granmo, B.J.Oommen, Svein A.Myrer and Morten G. Olsen

Determining Optimal Polling Frequency Using a Learning Automata-based Solution to the Fractional Knapsack Problem

[5] Leiming Chen, Dongmei Wang and Morten Goodwin Olsen

Towards distribution of web sites in a crawler used for large scale web accessibility assessment

Workshop on Web Accessibility and Metamodelling (WWAM2006)

[6] EIAO – European Internet Accessibility Observatory:

<http://www.eiao.net/>

[7] Wikipedia.org:

<http://en.wikipedia.org/wiki/>

[8] <http://www.google.com/>

[9] Vladislav Shkapenyuk, Torsten Suel

Design and Implementation of a High-Performance Distributed Web Crawler

[10] Trond Abusdal, Karl Syvert Løland, Ole-Alexander Moy and Aleksander M. Stensby

Topic Detection and Tracking using the Object Migration Automaton