ISSN 2278-3091
Volume 10, No.1, January - February 2021
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse111012021.pdf
https://doi.org/10.30534/ijatcse/2021/111012021

Development of Optimized Itinerary Agent Design Pattern Using Ant-Colony Algorithm

Fawwaz Al-Refa'e [1], Faiz Al-Shrouf [2], Shadi R.-Masadeh [3]
[1] Talal Abu Gazaleh, Java Developer, Amman-Jordan,
fawazalrefai@hotmail.com
[2]Isra University, Faculty of Information Technology, Department of Computer Science, Amman-Jordan,
Fayez.shrouf@iu.edu.jo
[3]Isra University, Faculty of Information Technology, Department of Computer Information Systems and Cyber Security, Amman-Jordan
Corresponding Author: shadi.masadeh@iu.edu.jo

ABSTRACT

The PC framework has been advanced from a solid PC gadget to a significantly more perplexing customer worker condition in earlier years. One of those recently evolved advancements is Mobile Agent. A Mobile Agent is a creation of program and information that can (move) starting with one PC then migrates onto the next node and proceeds with its execution on the goal PC. As a general rule, the portable operator is the code/object moving which goes in its itinerary inside the system of associated hubs. In this work, the best way in insignificant time is found by relocating the Mobile Agent from the source hub to the goal hub utilizing the numerical procedure and streamlining strategy. This work centers around how to locate the best way to utilize the Ant Colony Optimization calculation (ACO), then figuring out the best path, the path will be compared with those of similar works that used the master-slave design pattern with the Genetic algorithm and Node Compression Algorithm

Key words: Mobile Agent, Ant Colony Optimization Algorithm (ACO), Itinerary Design Pattern

1. INTRODUCTION

Mobile Agent technology is a way to build an intelligent generation of highly distributed systems because of its flexibility, capability, adaptability, and independency. However, there are issues that are not yet fully resolved by developers, such as reliability and safety.
Mobile agents work on a clear scenario, so users have only laptops with a network connection.
A mobile operator can work and move without staying on a mobile device, which means he can go to the Web sites to get information from the same vendor, the results obtained are then sent to the mobile device.
Mobile agents support flexible and adaptive load transfer from host to host, depending on bandwidth and other available resources. So the mobile agent technology is good for wired and wireless networks.
A migration strategy is a push-all data strategy. It is a strategy through which the code is fully transferred to all the nodes that the agent visits, not to the next node only.
Design patterns are very good in the process of improving the object-oriented system, the use of design patterns are intended to improve application development and identifying elements of reusable good designs for mobile agent applications.
There are different types of design patterns for agent moving, one of the design patterns used is the so-called Itinerary pattern, which objectifies agents' itineraries and their navigation among multiple destinations.
When agent tracking is complete, the results will be returned to the master, through this migration, pathways are formed to calculate the best path using one of the artificial intelligence algorithms; Ant Colony Optimization algorithm (ACO).

The ACO differs from the classical ant system in the sense that here the pheromone trails are updated in two ways. First, when ants construct a tour they locally change the amount of pheromone on the visited edges by a local updating role, after that, all the ants have built their individual tours, a global updating rule is applied to modify the pheromone level on the edges that belong to the best tour found so far.

2. RELATED WORK

Many researchers introduce mobile agents and design patterns with set of algorithms to optimize the time route amongst several paths,[2], proposed master-slave design and genetic algorithm (GA) patterns to find the best path in minimal time is found by migrating the mobile agent from the source node to the destination node using the mathematical model and optimization technique.[1], proposed a set of performance improvement patterns for use in mobile agent systems. These patterns are utilized to study agent behavior

and characteristics, in addition to the interaction between factors supported by the mathematical approach and models which support improved mobility performance. A set of master-slave samples were used to work with a group of mobile agents using the Aglet platform, the Tahiti server, and the Java Execution Environment (JEE). By sending a group of messages from master to slave and recording message response times in milliseconds, improved time is achieved with two types of design patterns (V-agent and P-agent, [5] introduced a comparative analysis of the most successful methods of optimization techniques inspired by Swarm Intelligence (SI): Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO), a comparative analysis is carried out to endow these algorithms with fitness sharing, aiming to investigate whether this improves performance which can be implemented in the evolutionary algorithms,[7], proposed Ant colony optimization and artificial potential field were used respectively as global path planning and local path planning methods. Some modifications were made to accommodate ant colony optimization to path planning. Pheromone generated by ant colony optimization was also utilized to prevent the artificial potential field from getting the local minimum. Simulation results showed that the hybrid algorithm could satisfy real-time demand. The comparison between ant colony optimization and the genetic algorithm was also made.

3. MOBILE AGENTS

Mobile agents can be defined as mobile codes that sent to a destination from a source (PCs) for a specific task [4]. Mobile agents are autonomous programs that can be transferred in a network, at a predefined time, starting with one PC then onto the next, places of their choice. By transporting to the destination, the running program is preserved. The program will be resumed and processed in the preserved state at the destination. For a few reasons, Mobile agents can convey an appropriate, vigorous, and productive framework for applying brilliant conditions and circulated tasks. These incorporate enhancements to the inertness and data transfer capacity of client-server applications, just as a diminishing weakness to interferences and disconnections in the network. "In fact, in both smart and distributed environments, mobile agents have numerous benefits in developing different services" [3]. A mobile agent is a software entity that migrates between host and host, performing data collection and configuration tasks on a user's behalf. "It migrates between agent runtime environments (AREs) and computers".

4. ITINERARY DESIGN PATTERN

Itinerary design pattern as shown by Figure 1, that shows overall scheme [4] (i.e. the structural relationship of participants itinerary design pattern). This scheme is arranged into the following categories:

- **Intent**

The Itinerary design pattern externalizes agents' routes and their routes among various destinations.

- **Motivation**

Being a self-sufficient mobile element, an agent is equipped for navigating itself autonomously to numerous hosts. In particular, it ought to have the option to deal with special cases, for example, obscure hosts while attempting to dispatch itself to new goals or make a composite visit (e.g., come back to destinations it has just visited). It may even need to change its itinerary dynamically.

Therefore, it is likely desirable over independent the treatment of route from the agent's behavior and message dealing with, in this way advancing measured quality of each part. The Itinerary design pattern allows you to do as such. The key thought is to move the duty regarding route from the agent article to an Itinerary object. The itinerary class will give an interface to keep up change the agent's itinerary and to dispatch it to new destinations. An agent object and an Itinerary object will be associated as follows: The agent will make the Itinerary object and instate it with, the first step is a roster of destinations to be visited consecutively and the second step a reference to the agent. At that point, the specialist will utilize the go method to dispatch itself to the following accessible destination in its schedule or back to its source, respectively. To support the abovementioned, it is fundamental that the Itinerary object is moved along with the agent, and that their references to one another be kept up at each destination.

- **Applicability**

Utilize this pattern when you desire to:

1. Conceal the specifics of interest of an agent's visit from its behavior to advance modularity of the two sections.
2. Give a uniform interface to consecutive heading out of agents to different hosts.
3. Define tours that can be taken by agents.

- **Participants**

Itinerary. Defines an interface for navigating with an agent.

1. ConcreteItinerary. Implements the Itinerary interface and keeps track of the current destination of the agent.
2. Agent. A base class of a mobile agent.
3. ConcreteAgent. A subclass of the Agent class that maintains a reference to a ConcreteItinerary object.

- **Collaboration**

1. ConcreteItinerary object keeps track of the current destination of the ConcreteAgent and can dispatch it to new destinations.
2. Whenever the ConcreteAgent is dispatched to a new destination, the ConcreteItinerary is also transformed, and their references to each other are restored at the target destination. Figure 2 shows collaboration of itinerary design pattern.

- **Consequences**

This pattern has three main consequences:

It bolsters variations in navigation. For instance, an alternate special case dealing with routine can be characterized if an agent neglects to dispatch itself to another destination: drop

the visit and come back to the starting point, attempt to go to another destination, and later attempt once more. This pattern makes it simple to give such varieties by essentially supplanting one Itinerary object with another or by characterizing Itinerary subclasses. The agent class isn't adjusted.

1. It encourages the sharing of visits by various agents. For instance, two agents may utilize a similar visit to multiple users' desktops, one to plan a gathering between all clients and the other to convey them notice messages. This pattern empowers agents to share visits by sharing itinerary objects, although not simultaneous.

2.It streamlines the usage of successive tasks. Errands can be typified in exceptional Task objects while an Itinerary class is reached out with an interface to relate Task objects with destinations. The itinerary object monitors the current tasks to perform. At whatever point the operator is dispatched to another destination, it triggers the execution of the current undertaking spared by its itinerary object. In Java-based agent frameworks, for example, Aglets and Odyssey in which agents are moved with just their code and information, and not with their whole execution express, the Itinerary design forestalls the need to physically monitor the execution condition of an agent (i.e. what the agent should do) when it travels.

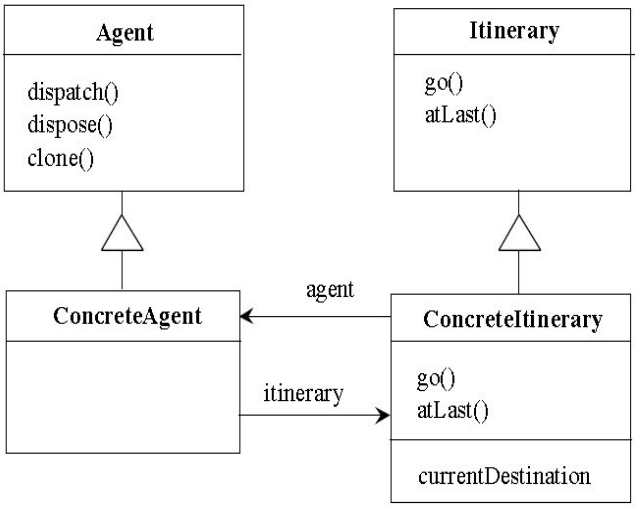


Figure 1: Structural Relationship of Itinerary Design Pattern

5.ANT-COLONY OPTIMIZATION ALGORITHM

The Ant Colony framework or the essential thought is developed by [5], [6], [7], as shown in Figure 2. The ants move in an orderly fashion to the food. The center picture represents the circumstance not long after an obstruction is embedded between the home and the food. To keep away from the impediment, at first, every subterranean ant decides to turn left or right indiscriminately. Let us expect that ants move at a similar speed keeping pheromone in the path consistently. Nonetheless, the ants that, by some coincidence, decide to turn left will arrive at the food sooner, while the ants that circumvent the hindrance turning right will follow a more drawn-out way, thus will set aside longer effort to dodge the deterrent. Thus, pheromone collects quicker in the shorter way around the obstruction. Since ants want to follow trails with bigger measures of pheromone, in the long run, all the ants join the shorter way around the hindrance.

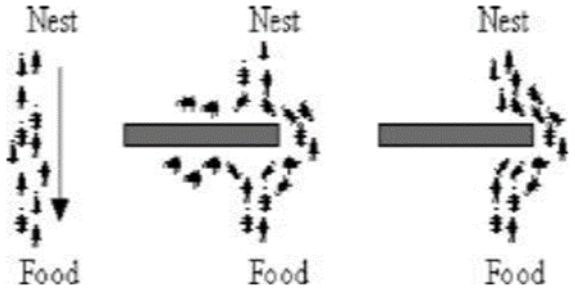


Figure 2: The Behavior of Real Ant

An artificial Ant Colony System (ACS) is an agent-based framework, which reproduces the regular conduct of ants and creates instruments of participation and learning. ACS was proposed by (M., Dorigo and L.,Gambardella, 1997) as another heuristic to take care of combinatorial improvement issues. This new heuristic, called Ant Colony Optimization (ACO) has been seen as both hearty and adaptable in taking care of a wide scope of combinatorial advancement issues.

The principle thought of ACO is to show an issue as the quest for a base cost way in a chart. Fake ants as though stroll on this chart, searching for less expensive ways. Every subterranean insect has fairly basic conduct fit for finding generally costlier ways. Less expensive ways are found as the emanant after effect of the worldwide participation among ants in the state. The conduct of fake ants is roused from genuine ants: they lay pheromone trails (clearly in a scientific structure) on the chart edges and pick their way as for probabilities that rely upon pheromone trails. These pheromone trails dynamically decline by vanishing. Also, counterfeit ants have some additional highlights not found in their partner in genuine ants. Specifically, they live in a discrete world (a chart) and their moves comprise of advances from hubs to hubs.

The ACO varies from the traditional subterranean ant framework as in here the pheromone trails are refreshed in two different ways. Firstly, when ants develop a visit, they locally change the measure of pheromone on the visited edges by a local updating role. Secondly, after all the ants have manufactured their visits, a global updating rule is applied to

adjust the pheromone level on the edges that have a place with the best subterranean insect visit found up until now.

5. PROPOSED METHODOLOGY

Proposed methodology as given by Figure 3, is to create a mobile agent, that moves between a set of hubs in the network to perform a task. Then, we created the itinerary pattern, which concerned with routing among multiple destinations and always knows where to go next and finally return to resource. The second step was to calculate the time needed to process the data between each hub and the other hubs from the source to the destination by using the push-all data migration strategy. In the migration process, a variable data size was used to calculate the change that affects the process. We then take a test case, which is seven test cases for each case, and we collect and take the average among them. The third stage computing the shortest path from the source to the destination by sending number of ants then the route that most ants take is the shortest path.

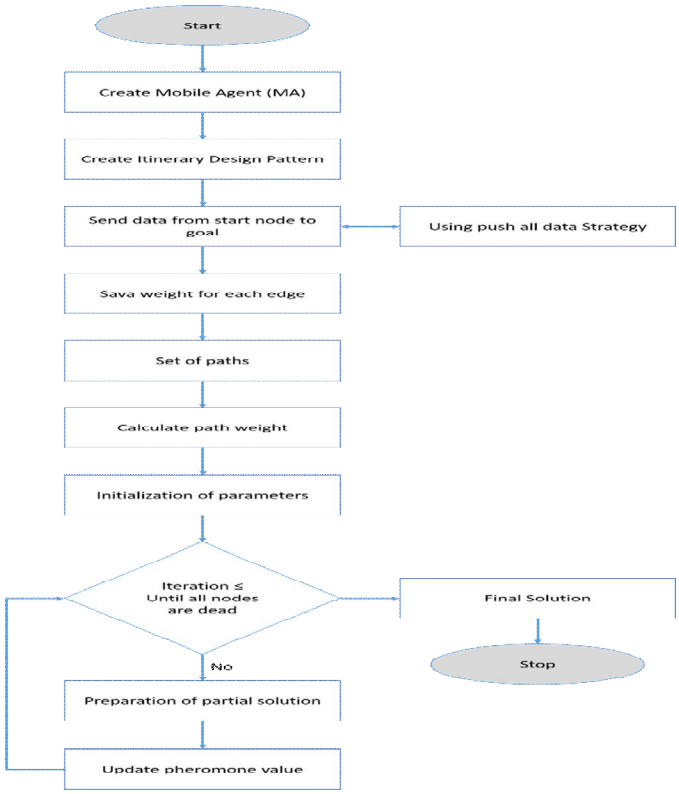


Figure 3: The Proposed Methodology

5.1 Creating Mobile Agent

In the primary stage, the mobile agent is created, which is an autonomous code that moves starting with one PC then onto the next in a similar network. Likewise, this code is portrayed by different highlights, including the progress starting with one hub then onto the next to do undertakings without client intercession. The assignment actualized in our work is handling information in hubs from the beginning stage to the interface and afterward restoring the outcomes to the source.

5.2 Creating Itinerary Design Pattern

The first step is to create an "ItineraryAgent", whose type is "Agent". This agent triggers "Itinerarybehaviour", whose type is "OneShotBehaviour", that sets the itinerary theat agent
must follow, and begins the relocation. The agent execution is constrained by two main methods "beforeMove" and "afterMove" that controls the relocation and permits the execution of the activity, individually. The operator work is executed in "JobBehaviour" (of type "OneShotBehahviour") class, added to the agent at whatever point it shows up to the new goal. A significant detail: before moving, the specialist confirms, through agent management system, if the goal compartment exists. If it doesn't exist, it attempts to relocate to the following goal in its schedule. This check is given in class "GetAvailableLocationBehaviour" which is accessible in the model codes given by the framework.

5.3 Push All Data Migration Strategy

In the subsequent stage, the time between every hub and the other is determined by utilizing the push-all information movement strategy. This relocation system depends on the formation of a diagram dependent on the schedule from the beginning hub to the end hub, through the middle of the road hub that moves information. Relocation is completed to all goals that will be visited by the specialist, which requires the operator to realize every one of its goals ahead of time. In the three periods of this stage, time will be determined between every two edges. Agent moved with three agent data collections, the first with a small size of (10) KB and the second with a medium size of (50) KB and the third with a large size of (100) KB.

5.4 Implementing Ant Colony Optimization Algorithm

After pushing the data: first, when ants develop a visit, they locally change the measure of pheromone on the visited edges by a local updating role, second, after all the ants have implemented their visits, a global updating rule is applied to adjust the pheromone level on the edges that have a place with the best visit found up until now, then determining the route that most ants take to determine the shortest path.

6. ANALYSIS OF RESULTS

Results are reported according to three different stages: the first stage concerned with creating the agent and itinerary design pattern using Java Agent Development Environment (JADE) as shown in Figure 4. After running the mobile agent and the itinerary design pattern, the time taken between all edges in the network is calculated using the push-all data strategy, where variable data was taken to calculate the time between a set of edges from the source to the destination. The time is calculated in milliseconds. It is noted that implementation has taken place on personal

computer with the following specifications: Processor: Inter(R) Core (TM) i7-3632QM CPU @ 2.20 GHz. (RAM): 16.0 GB. System Type: 64-bit Operating System. The implementation resents the case when the agent's size is 10 KB., 50 KB, and 100 KB respectively.

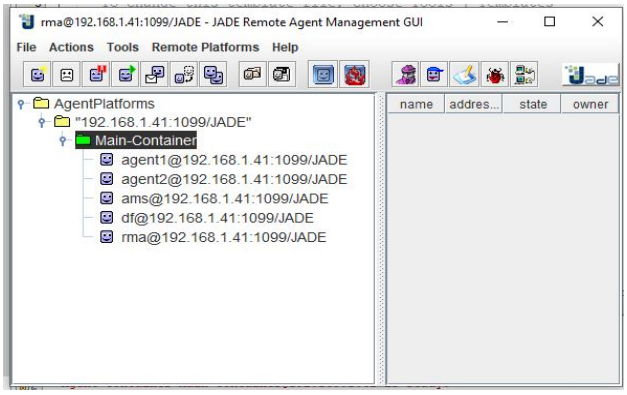


Figure 4: Running Agents on JADE Platform

The time is calculated and comparison was conducted between this research which uses itinerary design pattern and Ant Colony Algorithm (ACO) and results reported from implementing master-slave design pattern and Genetic Algorithm (GA), as shown in Table 1.

Table 1: Calculating time using ACO and GA

Start Node	End Node	Time by the size of data in MS					
		10 KB		50 KB		100 KB	
		ACO	GA	ACO	GA	ACO	GA
1	2	1112	1175	2755	2910	2723	2877
1	3	1084	1145	2582	2728	2487	2627
1	4	1243	1313	2836	2996	2641	2790
2	4	764	807	1260	1331	832	879
3	4	934	987	1441	1522	851	899
3	6	415	438	1871	1547	1748	1847
2	5	426	450	1464	1977	2407	2543
6	9	500	528	3381	1764	1947	2057
5	8	924	976	3596	3799	4425	4675
5	11	768	811	3135	3312	3932	4154
4	7	691	730	1670	3572	4462	4714
7	8	926	978	2377	2511	2410	2546
8	10	941	994	1760	1859	1366	1443
8	13	868	917	2770	2926	3154	3332
9	13	1128	1192	2369	2503	2062	2178
10	11	736	778	2662	2812	3196	3376
10	12	926	978	2591	2737	2763	2919
10	13	1122	1185	2379	2513	2088	2206
12	19	878	928	2898	3061	3351	3540
14	19	685	724	2400	2535	2475	2615
13	16	771	815	2261	2389	2847	3008
11	14	892	942	2619	2767	2866	3028
12	15	809	855	2228	2354	2371	2505
15	19	748	790	2998	3167	3732	3943
15	17	789	834	2348	2481	2590	2736
16	17	954	1008	2259	2386	2168	2290
17	20	766	809	2500	2641	2880	3042
17	18	777	821	2735	2889	3249	3432
18	20	1016	1073	2540	2683	2529	2672
19	20	1076	1137	2471	2610	2315	2446

To determine the shortest path at this stage, the NetBeans Platform is used the artificial intelligence technique to calculate the shortest path by optimization the execution time. For this purpose, an ACO algorithm was used, then a specific number of artificial ants are sent, and the ants follow their path according to the pheromone, which depends based on the total weight of all edges, then calculate the time taken for the optimal path in the case of the algorithm mentioned.

The case is used with the following parameters:
Q: 0.0005

The parameter is used to justify the amount of pheromone deposited.

RHO: 0.2

The parameter used for varying the level of pheromone evaporation.

ALPHA: 0.01

The parameter used for controlling the importance of the pheromone trail.

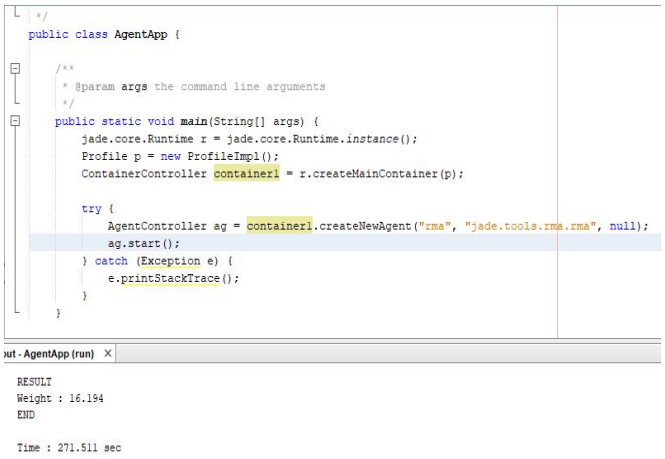
BETA: 9.5

The parameter used for controlling the importance of the distance between source and destination.

NUMBER_OF_ANTS: 500

The parameter used for the number of ants that were sent.

Results are extracted from this case that shows the path picked with minimal weight. The implementation for this situation is given in Figure 5.



```
*/
public class AgentApp {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        jade.core.Runtime r = jade.core.Runtime.instance();
        Profile p = new ProfileImpl();
        ContainerController containerl = r.createMainContainer(p);

        try {
            AgentController ag = containerl.createNewAgent("rma", "jade.tools.rma.rma", null);
            ag.start();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

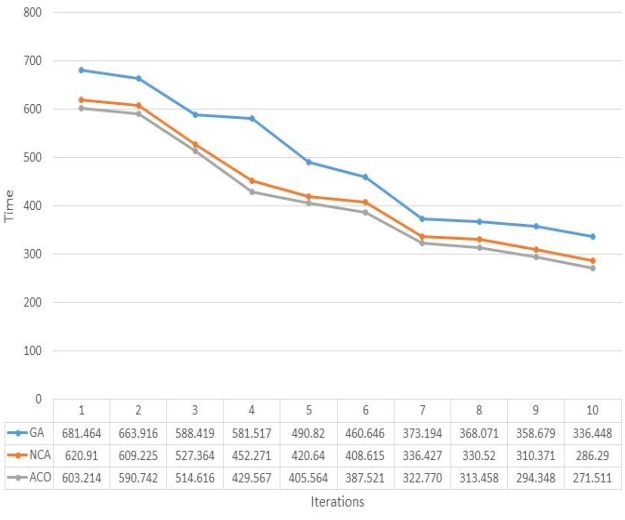
```
ut - AgentApp (run) ×
RESULT
Weight : 16.194
END

Time : 271.511 sec
```

Figure 5: Time Calculation using ACO Algorithm

7. CONCLUSIONS

Figure 6 shows a comparison in computation of time in milliseconds between two different algorithms implemented previously namely Node Compression Algorithm (NCA), Genetic Algorithm (GA) and Ant Colony Algorithm. We carry out independent iterations and reporting results. Results concluded that time computed using ACO gives best results comparing to other two algorithms (NCA) and (GA).



	1	2	3	4	5	6	7	8	9	10
GA	681.464	663.916	588.419	581.517	490.82	460.646	373.194	368.071	358.679	336.448
NCA	620.91	609.225	527.364	452.271	420.64	408.615	336.427	330.52	310.371	286.29
ACO	603.214	590.742	514.616	429.567	405.564	387.521	322.770	313.458	294.348	271.511

Figure 6: Comparison Results for Time Computation using NCA, ACO, and GA

8. FUTURE WORK

In this research, a model based on mobile agent and Itinerary design pattern has been proposed to calculate the shortest path for the mobile agent migration from source to destination in the minimum time using the push all-data strategy as well as one of the artificial intelligence algorithms called the ACO algorithm. We have used the ACO algorithm because it provides good features in choosing the shortest path among a set of alternative paths. By comparing the results of the ACO algorithm with the GA algorithm and with the NCA algorithm, the ACO has proved to be more effective. Finally, the proposed model was designed based on the JADE in the Java Devel The researchers suggest some important research points that should take into considerations for future work including, the following research areas:

- Implementing another agent design pattern that explores the agent's behavior.
- Using another platform for implementation to carry out computations.
- Suggesting another development algorithm such as the Dijkstra algorithm and comparing its results with those of similar works.
- Applying the pull-all data strategy which is the opposite of the push-all data strategy.

ACKNOWLEDGEMENT

The authors owe thanks to Scientific Research Deanship at Isra University for facilitating procedures of conducting this research and its financial support for this research.

REFERENCES

1. F. Al-Shrouf,A.Turani, A. Abu Baker, A.Al Omari. **Analysis of Mobile Agent Optimization Patterns**, *British Journal of Applied Science & Technology*, Vol. 4, No. 12, pp. 1841-1857, 2014.
2. R. Allawi, A. Al-Hroob, F.Al-Shrouf. **Development of Optimized Mobile Agent Design Pattern using Push—All Strategy.** MS.c Thesis, Faculty of Information Technology, Department of Software Engineering, Isra University, Amman- Jordan, 2019.
3. F. Al-Shrouf, Sh. Masadeh, F. Al-Zyoud. **Mathematical Model for Comparing Performance Evaluation of Mobile Agent Platforms**, *International Journal of Theoritical and Applied Information Technology*, Vol. 98, No.2, pp. 338-348, 2020.
4. Y. Aridor, D. Lange. **Agent Design Patterns: Elements of Agent Application Design**". *In Proceedings of 2nd International Conference on Autonomous Agents*,USA, pp. 108-115.1998.

5. V. Selvi, R. Umarani**. Comparative Analysis of Ant Colony and Particle Swarm OptimizationTechniques**, *International Journal of Computer Application,* Vol. 5, No. 4, pp. 1-6,2010.
6. M.Dorigo, M. Birattari, T. Stuzle, **Ant Colony Optimization.**, *IEEE Computational Intelligence,* Vol. 1, No. 1, pp. 53-66. 2006.
7. H. Mei, Y. Tian, L. Zu, **A Hybrid Ant Colony Optimization Algorithm for Path Planning of Robot in Dynamic Environment**, *International Journal of Information Technology,* Vol. 12, No. 3, pp.78-88, 2006.