Cost Effective Mobile Agent Planning for Distributed Information Retrieval

Jin-Wook Baek, Jae-Heung Yeo, Gyu-Tae Kim, Heon-Young Yeom
School of Computer Science and Engineering
Seoul National University
{jwbaek, jhyeo, gtkim, yeom}@dcslab.snu.ac.kr

Abstract

The number of agents and the execution time are two significant performance factors in mobile agent planning. Fewer agents cause lower network traffic and consume less bandwidth. Regardless of the number of agents used, the execution time for a task must be kept minimal, which means that use of the minimal number of agents must not impact on the execution time unfavorably. As the population of the mobile agent application domain grows, the importance of these two factors also increases.

After careful review of these two factors, we propose two heuristic algorithms for finding the minimal number of traveling agents for retrieving information from a distributed computing environment, while keeping the latency minimal. Although agent planning, specifically Mobile Agent Planning (MAP), is quite similar to the famous Traveling Salesman Problem (TSP), agent planning has a different objective function from that of TSP. TSP deals with optimal total routing cost, while MAP attempts to minimize the execution time to complete tasks of information retrieval. In this paper, we suggest the cost-effective MAP algorithms, BYKY1 and BYKY2, which can be used in distributed information retrieval systems to find the factors mentioned above. At the end of each algorithm, 2OPT, a well-known TSP algorithm, is called to optimize each agent's local routing path. Experimental results show that BYKY2 produces near optimal performance. These algorithms are more realistic and applicable directly to the problem domains than those of previous works.

***Keywords**: Mobile Agents, Mobile Agent Planning, Mobile Computing, Distributed Agent System, Distributed Information Retrieval*

1. Introduction

Mobile agent technology can reduce network traffic, overcome network latencies, and enhance robustness and fault-tolerant capabilities of distributed applications [1][6]. One of the major potential application areas for mobile agents is information retrieval, which involves accesses to a huge amount of data across a network [13][14][15]. Instead of transmitting data across the network, an agent migrates to the host where the database is located, performs its task there, and then returns to the original host bearing result. Thus, the mobile agent can utilize the bandwidth of the network much more efficiently than can be achieved by accessing the distributed database remotely using direct connection. The total computation time taken can be shortened, especially when data transmission is the bottleneck of the task [2][3][11][16]. In this domain, information is spread over several hosts and it is common for these hosts to be geographically separated [9][10]. Should the mobile agents be ordered to retrieve information in information retrieval systems, all the pertinent nodes have to be specified, and these nodes must be fully covered.

The number of agents and the execution time are two significant performance factors in mobile agent planning. Using fewer agents causes lower network traffic and consumes less bandwidth. Agents consume network bandwidth when they travel over the designated set of nodes. Badly scheduled agents' itineraries can cause longer execution times as a result of higher routing costs. The number of agents created for a task also influences the total routing cost. Clearly, the greater the number of agents created for a task, the higher the overall routing cost. In order to reduce this overhead, as well as to provide better performance, agents must be scheduled before they are sent. Research has been carried out on information retrieval and agent planning. We will focus on agent planning techniques in information retrieval that we call Mobile Agent Planning (MAP).

In some papers such as [12], MAP has been recognized as a Traveling Salesman Problem (TSP). It is claimed that if the agent's task is to visit all the nodes, then the problem is reduced to the classical TSP, which is known to be NP-hard. However, TSP and MAP are different in that TSP deals with the optimal total routing cost with a given number of agents, whereas MAP attempts to minimize the execution time to complete the information retrieval. Another difference is that agents can visit any node more than once in MAP, but normally not in TSP. Being different from [5] and [11], where "succeed-stop" agent behavior is used, we deal with "full-visit" behavior, in which the planning covers all the nodes to be visited.

The number of agents for a task should not be fixed, otherwise a trade-off between performance and routing cost cannot be made, and consequently, the application system would not be flexible. Instead, we try to find the minimum number of agents required, while maximizing the performance. These two objectives are extremely important in mobile agent systems, since the whole system must satisfy both conditions: faster execution time, and lower network traffic caused by agents.

Instead of employing unrealistic assumptions to solve the problem in polynomial time, it is preferable to try to find reasonable heuristic solutions even under the NP-hard conditions. In this paper, we suggest two cost-effective MAP algorithms named BYKY1 and BYKY2. They can be used in distributed information retrieval system to find the two performance factors. Although not primary goal, total routing cost is still important in MAP problems, since it decides the traveling time of an agent allocated in the local network. Thus, it is desirable to tackle the routing cost to a certain degree. This is the reason that 2OPT, a simple TSP algorithm, is employed in the proposed algorithms. It optimizes each agent's local routing path. The experimental results show BYKY2 produces a near optimal solution. Adopting these algorithms, the information retrieval system can retain the best performance while achieving a minimum routing cost with the minimum number of agents.

In Section 2, related works are described. Section 3 and Section 4 describe the cost-effective MAP problem and its algorithms, i.e., BYKY1, and BYKY2, respectively. In Section 5, experimental results are presented and analyzed, and the paper's conclusion is in Section 6.

2. Related Works

Until recent years, the MAP problem has not been focused on by many researchers. Simply, it has been regarded as an ordinary routing problem such as the TSP. The TSP is clearly NP-hard [8]. Efforts to solve MAP have been made using greedy methods, dynamic programming, etc. To reduce the agent planning problem into greedy or dynamic programming, many assumptions had to be made.

Careful consideration must be given in employing the assumptions, since they limit the applicability and extensibility of the idea. Brewington et al. [5] formulate a method of MAP, namely the Traveling Agent Problem (TAP), which is analogous to the TSP [7], to decide the sequence of nodes to visit to minimize the total execution time until the desired information is found. This formula denotes that the agent has to return to the home node if at any time during its tour success occurs. We call this succeed-stop behavior. The other important agent behavior that came from TSP is making at the most, one visit, which means the agents never revisit nodes.

Resource allocation is essentially an allocation problem. Moizumi [17] explores how agents can efficiently spend their time traveling throughout the network completing a set of tasks. He poses the problem as a traditional "traveling salesman" problem in an environment where latencies, possibly stochastic, represent distances. While the TSP is NP-hard, Moizumi uses dynamic programming to solve the problem in polynomial time by assuming that latencies are constant.

While not directly related to mobility, Wellman et al. used dynamic programming to derive an algorithm to find the shortest stochastic cost path through a network [10]. Their algorithm could be applied to Moizumi's problem if the sequence of the tasks to be performed is known. Additionally, Wellman et al. augment the algorithm to allow path refinement as the algorithm's user travels.

3. The Cost-Effective Mobile Agent Planning (CE-MAP) Problem

Table 1 summarizes the notation used in this paper.

Symbols	Description
N	Number of nodes excluding the home node
r	Number of mobile agents employed for a task
H	Home node
δ	Execution time to complete a task
$h_1,h_2,\ldots,h_n$	Node identifiers
$A_1,A_2,\ldots,A_r$	Agent identifiers
tour	Sequence of nodes visited by an agent
$Tour(A_i)$	*tour* of agent A_i, e.g., $(i_1,i_2,\ldots i_k)$ where i_j, $1\leq j\leq k$ is node index and $(i_1,i_2,\ldots i_k)$ be the *tour* S_i
$Comp(h_i)$	Computation time at node h_i
$L_s(h_i,h_j)$	Shortest latency between nodes h_i and h_j
$Union(S_i,\ldots,S_j)$	Concatenation of *tours*, where $S_i,\ldots,S_j$ represent *tours*
$TourT(S_i)$	Routing time, namely execution time, for *tour* S_i
$First(S_i)$	First entry of *tour* S_i, i.e., i_1
$Last(S_i)$	Last entry of *tour* S_i, i.e., i_k

Table 1. Notations used in this paper.

We introduce three important definitions in Table 1 as follows:

Definition 1 (The tour of an agent: $Tour(A_i)$) *$Tour(A_i)$ is the tour of agent A_i, which consists of the nodes to be visited for processing an agent's task in order.*

Definition 2 (The smallest latency: $L_s(h_i,h_j)$) $L_s(h_i,h_j)$ is the smallest latency between nodes h_i and h_j, which is found by evaluating the shortest paths [8] between all pairs.

Definition 3 (The tour time: *TourT(S)*)

$$TourT(S) = \begin{cases} 2 * Ls(H, h_1) + Comp(h_1), & if\ S = (h_1) \ldots\ldots\ldots (a) \\ TourT(S1) + TourT(S2) - Ls(H, First(S1)) - Ls(H, Last(S2)) + Ls(First(S1), Last(S2)), & if\ S = \bigcup(S1, S2) \ldots\ldots (b) \end{cases}$$

where *S*, *S1*, and *S2* represent *tours*.

Figure 1 (a) explains the definition 3-(a). Figure 1 (b) represents the merging of two tours, *S1* and *S2*, into *S*, where *Last(S1)*= i_k, *Last(S2)*= i_{k+1}. Tour *S* consists of two sub-tours, *S1* and *S2*. Thus, *TourT(S)* is defined by the addition of all the latencies from *H*, i_1, …, i_n, *H*, and the computation time at each node. If *S1* and *S2* are merged then *TourT(S)* has to be reconfigured in accordance with definition 3-(b).

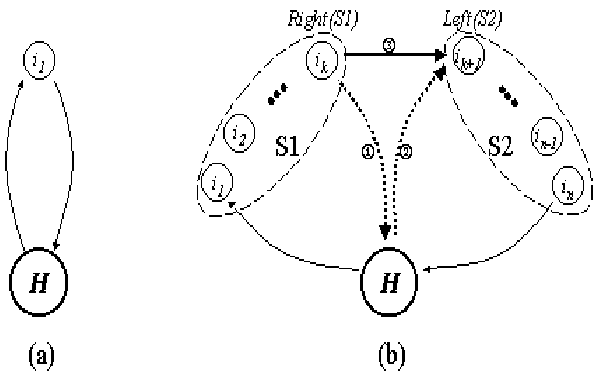

Figure 1. Tour and merging tours

3.1. Problem definition

We assume that a MAP module knows the network statistics and the history in a network monitoring service. This service enables mobile agents to acquire best route information for the task. The mobile agent problem can be described as follows:

> ***Mobile Agent Planning Problem*** - There are *n+1* nodes, *(H,h₁,h₂,h₃,…h_n)*, where *H* is the home node and the other numbers are node indexes. Each node has a computation time required for a mobile agent to perform the task at node h_i. Latencies for the mobile agent to move between each node h_i and h_j are also known. For the home node *H*, the computation time at H, i.e. t_H, is equal to zero. The mobile agent problem is to minimize the execution time δ, and the number of mobile agents *r*, to successfully complete the task.

As expressed below, the execution time (δ) can be defined by the longest routing time (*TourT*) among all the nodes an agent can visit. This comprises network latency from home (*H*) to node (*i*) plus the execution time on it.

$$\delta = Max\{TourT(h_i)\}, \quad 0 \le i \le n \quad (1)$$

A solution to the MAP problem is to find a sequence of agents to visit the nodes; this solution consists of the minimum total execution time and the number of agents needed. This is considerably different approach from that of [11].

Two important objectives in MAP are 1) the minimum execution time; and 2) the least network bandwidth consumed by agents. The minimum execution time for a task is not smaller than that of special case when *n* agents are sent to *n* different nodes. Therefore, it totally depends on the node that has the largest sum of latency and computation time. The number of agents affects network bandwidth. Therefore, the number of agents for a task cannot be fixed, otherwise no trade-off between execution time and routing cost is possible, and consequently, the system would not flexible. Therefore, we try to find the minimum number of agents we need to prepare just before sending the agents, while achieving the minimum execution time for a task.

Our goal is to decide the sequence of nodes for each mobile agent to visit, the minimum number of agents needed, and the minimum execution time, i.e., δ, for a task. The CE-MAP is formally defined as follows.

Cost-Effective MAP:

$$Minimize\ wr + \sum_{1 \le i \le r} TourT(Tour(A_i)) \le \delta, \quad 1 \le i \le r \qquad (2)$$

Subject to

$$TourT(Tour(A_i)) \le \delta, 1 \le i \le r \qquad (3)$$

$$|\bigcup_{i=1}^{r} Tour(A_i)| = n,\ Tour(A_i) \bigcap Tour(A_j) = \phi, i \ne j \qquad (4)$$

$$\delta = \max(TourT(h_i)), 1 \le i \le n \qquad (5)$$

In expression (2), let *w* be a value larger than the total routing cost, namely the sum of the tour times all of the agents that appeared in the second term of (2). Therefore, expression (2) has two targets: 1) the minimum number of agents; and 2) the minimum overhead, i.e., the total routing cost. The lines from (3) to (5) are constraints for this objective function. Expression (3), the routing cost of each agent is always smaller than this minimum, namely δ. This means that a given task can always be completed within the minimum, which is determined by the node with the largest sum of latency and computation time. Expression (4) requires that each node be processed exactly once. Since this problem is NP-hard, we must develop cost-effective solutions based on heuristics.

Figure 2 (a) shows a network configuration that can be partitioned into two parts. The weight of an edge represents the expected latency for the connection between a pair of nodes. The computation time on a node appears on the corresponding node. The minimum execution time is even smaller than *50ms*, since the node h_1 has the maximum routing cost of the configuration. The execution time for node h_1 is *50ms* (*=10+30+10*), whereas the costs for nodes 2 and 3 are *21ms* and *25ms*, respectively. Here, the minimum number of agents for the example is two. The first agent covers node h_1 as shown in Figure 2(b), and the second agent covers nodes h_2 and h_3 as shown in Figure 2(c). The total execution time is *88 ms* (*=50+38*). Note that the sequence in the tour taken by the second agent is (H,h_3,h_1,h_2,H). Due to usage of the shortest path over all pairs, the second agent makes use of a bypass rather than the direct connection between node h_2 and h_3. Thus, *Tour*(A_1)=(h_1) and *Tour*(A_2)=(h_3,h_2).

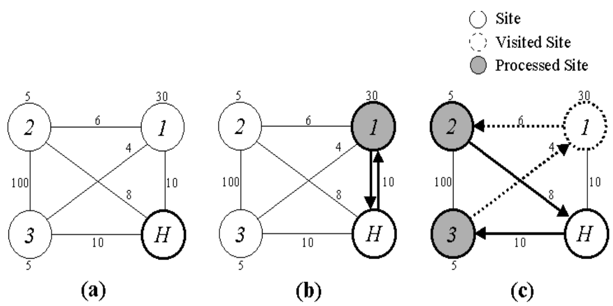

Figure 2. An example network configuration

4. The Proposed Planning Methods

In this chapter, we suggest two cost-effective MAP methods in detail. Assuming that such network statistics as the latency and bandwidth of the links between hosts, and the load on each host are collected by a network monitoring module in the system, we could obtain all the latencies between existing links. We assume that the network latencies vary. We also assume that the agents are created only at home and are not cloned in other nodes. We do not consider any probability of success in retrieving the required data at each node, as was done in [11].

4.1. Preprocessing all pairs shortest path

One of our goals is to find the lower bound of execution time. The number of participating agents must not affect this lower bound. The fastest way to complete a task is to send agents one by one to each node simultaneously. Then the execution time depends totally on the slowest link between a node and home. The time value comprises the network latency and the computation time. How can we find the latencies of all the possible pairs of nodes? This is not a simple job, since we have to search all the combinatorial links of any two selected nodes. Without this information, there is no way to find the lower bound of the execution time. To get the latencies of all pairs of nodes, our algorithms, before entering the main body, process the all-pairs shortest-path algorithm, and construct a shortest-latency network graph. Refer back to Figure 2(c). The graph shows a bypass of an agent. Although the latency on the direct link between nodes h_2 and h_3 is *100ms*, the agent is able to take an indirect *tour* of (H,h_3,h_1,h_2,H), instead of (H,h_3,h_2,H). The cost of the former is *38 ms* while the cost of the latter is *128ms*. This process is required as pre-processing stage before the start of our algorithms.

4.2. The algorithms

Providing that all the smallest latency information between any two nodes h_i and h_j, $L_s(h_i,h_j)$, is available as described in the previous section, the algorithm proceeds as follows:

1. Sort the nodes in decreasing order of the routing time of each node h_i, which is *TourT*(h_i).
2. Set the threshold δ with the execution time of the first node in the sorted list.
3. Partition the given network into several parts by gathering nodes so that the execution time of each part does not exceed the threshold and build a routing path for each partition.
4. Run TSP algorithm to optimize each routing path.
5. Allocate and send an agent for each partition.

We developed two algorithms named BYKY1 and BYKY2 that perform the same task, but use a slightly different partitioning (step 3) method. They are different in that the latter is more dynamic than the former. BYKY1 tries to find the next possible partition by calculating all the latencies always from home, whereas BYKY2 searches the next node from the current node where the agent resides. The first algorithm, BYKY1 is presented in Figure 3. At the end of the algorithm, it calls a TSP program. We use the 2OPT algorithm [4] because it is simple and effective. It produces an asymptotically optimal routing path for each given network partition. If more than one node has the same *TourT* value, the selection of the node is arbitrary.

Note that each node is ordered in decreasing order of the execution time of each tour. BYKY1 estimates the agent's traveling time by adding the costs of the next adjacent set of nodes in the sorted list. Exceeding the threshold is not allowed in calculating the traveling time. As described above, the threshold is the execution time of the first node in the sorted list. The cost for each node is the cost of the round-trip time from home. In contrast, BYKY2 adds the cost of a different set of nodes. The node selected by BYKY2 always has the minimum traveling time from the last added node, not from home. Thus, the results of these two algorithms may have different routing paths for a given agent. Figure 4 is the algorithm for BYKY2.

```
Algorithm 1. The Planning Algorithm BYKY1
Phase 1.  Sorting the nodes
  Sort the nodes in terms of the tour time, TourT by
  decreasing order
  Let the sequence (h_a1,h_a2,h_a3,…,h_an) be the resulting
  sequence
  δ = TourT(h_a1)
Phase 2.  Planning agents
  for j=1 to n
  {
    Tour(j) = phi
    for k=1 to n
    {
      if h_ak is not "processed" and TourT(Union(A_j,
         (h_ak)) ≤ δ
        Union(A_j,(h_ak));
        mark h_ak as "processed"
    } end of inner for-loop
    if A_j = φ terminate
  } end of outer for-loop
Phase 3.  Optimization of agent's local path
  For each agent A_i
    2OPT(A_i)
```

Figure 3. The BYKY1 algorithm

```
Algorithm 2. The Planning Algorithm BYKY1
Phase 1.  Same as the phase 1 in the Algorithm 1.
Phase 2.  Planning agents
  for j=1 to n
  {
    Tour(A_j) = φ
    Select "unprocessed" h_ak where k is minimum
    Union(Tour(A_j),(h_ak)); mark h_ak as "processed"
    while(TRUE)
    {
      sort nodes in terms of L_s(h_ak,h_al)|1 ≤ l ≤ n and
      h_al is "unprocessed" by increasing order
      let the sequence (h_b1,h_b2,…,h_bm) be the resulting
      sequence
      previous_k = k
      for x=1 to m {
        if TourT(Union(A_j,h_bx) ≤ δ {
          Union(A_j,h_bx); mark h_bx as "processed"
          find l where a_l = b_x
          break for-loop
        } end if
      } end of inner for-loop
      if previous_k = k break while-loop
    } end of while-loop
    if Tour(A_j) = φ, terminate
  } end of outer for-loop
Phase 3.  Same as the phase 3 in the Algorithm 1.
```

Figure 4. The BYKY2 algorithm

5. Simulation

5.1. Simulation environment

The simulation is based on practical perspective of the mobile agent environment for the distributed information retrieval system, which means that all the simulation measurements and assumptions are chosen for surroundings that are more realistic. Network topology and computation time are the two major measurement factors of the simulation. The network topology is one of the following:

- All the nodes are located in a sub-network (LAN)
- Each sub-network includes exactly one node (WAN)
- Each sub-network includes more than one node (Clustered-WAN)

The network topology is related to the way we assign the latencies to each pair of nodes in a network. When all the nodes that agents must visit are in a local sub-network (LAN), we perceive that the standard deviation of the latencies that exists between each pair of nodes in the network is very small. In this case, the only factor we must take into account is the computation time for the nodes. Thus, the problem space is drastically reduced. In contrast, if all the nodes are scattered over a wide area network (WAN) such as the Internet, the latencies will be arbitrary. In this case, the search space of the problem would be larger than that of the former. The last topology is a mixture of the two (Clustered-WAN). This topology consists of several sub-networks distributed over a wide area network. If we choose two nodes in this topology, they may be located in either the same sub-network or different ones. Thus, the number of agents needed can also be related, but not necessarily so, to the number of sub-networks. In summary, the latencies measured in the experiment for all the different topologies are as listed in Table 2.

Communication Type	Average Latency Range
Intera-LAN	*3*ms *15*ms
Inter-LAN	*15*ms *200*ms

Table 2. Latency ranges

We can classify the computation behaviors into two types: simple retrieval query, and complex query. Simple retrieval query requires a small computation time. Headline news gathering is of this type. Complex query such as meta-search requires relatively large amount of computation time. This type of retrieval requires an extensive search and summarizing process. Simple retrieval query has computation time range of *1~50ms*, whereas complex query consumes *500~10,000ms*. Table 3 shows the "Clustered-WAN" configuration. The numbers of nodes are 13, 14, 15, and 16. They are clustered into four subnets (clusters).

# of Nodes	# of Clusters	Cluster Configurations
13	4	4+3+3+3
14	4	4+4+3+3
15	4	4+4+4+3
16	4	4+4+4+4

Table 3. Number of nodes and cluster configuration

The simulation was performed under the environment described for three types of topology and two kinds of query with four different numbers of nodes. Since the optimal solution with more than 16 nodes requires a tremendous amount of search space and time, and is difficult to perform in reality, our experiment is performed with a maximum of 16 nodes. Instead, with up to 300 nodes, the experiment is performed to find the number of agents, the latency costs consumed by all the agents, and the actual planning time by BYKY1 and BYKY2 for a given task. This is compared with the worst-case algorithm that launches as many agents as there are nodes.

The simulation was performed on three different kinds of machines: Sun Sparc 10, Intel Pentium III 500MHz with 64MB RAM, and one Intel Celleron 533MHz with 64MB RAM. All the results gathered and presented from those machines were averaged to reflect the average computing environment. Note that, as the computing power increases, the calculated performance definitely improves.

5.2. Simulation Results

Table 4 shows the number of agents and the total cost required for each combinatorial configuration, as found by three different algorithms: BYKY1, BYKY2, and Optimal; number of nodes; network topology (LAN, WAN, Clustered-WAN); and query type (Simple and Complex).

The optimal solution represents the fact that the simple computation type requires more agents than that of the complex one in the LAN configuration. This is because the simple one consumes less time (*1~50ms*) than the complex one (*500~10,000ms*). In this case, the total computation time is evenly distributed. Since the task must be completed under a fixed latency, the number of agents is close to the number of nodes that provide information retrieval services.

In contrast, when the computation time at each node is large (complex query), the total computation time may depend upon some nodes that take large computation times. Then the agents, which cover the other nodes that consume less time, can visit more nodes under the same latency; thus, they require fewer agents.

# of nodes	Topo logy	Query Type	BYKY1		BYKY2		OPTIMAL	
			#agens	cost	#agents	cost	#agents	cost
13	LAN	Simple	12	2,720	12	2,720	12	2,720
		Complex	9	73,706	9	73,725	9	73,699
	WAN	Simple	10	5,735	7	4,756	7	4,635
		Complex	8	56,661	7	56,226	7	56,010
	Clustered -WAN	Simple	9	4,319	8	4,056	6	3,295
		Complex	6	39,784	6	39,761	6	39,029
14	LAN	Simple	13	2,940	13	2,940	13	2,940
		Complex	10	82,878	10	82,897	10	82,871
	WAN	Simple	12	6,139	10	5,309	9	5,130
		Complex	9	61,749	8	61,223	8	61,098
	Clustered -WAN	Simple	11	4,756	9	4,253	9	4,242
		Complex	7	44,399	7	44,092	7	43,568
15	LAN	Simple	14	3,165	14	3,165	14	3,165
		Complex	7	58,955	7	58,897	7	58,868
	WAN	Simple	11	6,903	9	5,639	7	4,851
		Complex	10	80,581	9	79,981	9	79,550
	Clustered -WAN	Simple	10	5,128	9	4,658	9	4,512
		Complex	13	98,269	12	97,984	7	97,826
16	LAN	Simple	15	3,389	15	3,389	15	3,389
		Complex	7	59,623	7	59,557	7	59,520
	WAN	Simple	11	6,929	10	6,103	8	6,000
		Complex	10	85,965	9	85,328	9	84,976
	Clustered -WAN	Simple	9	5,161	7	4,359	7	4,300
		Complex	13	99,771	12	99,688	12	99,618

Table 4. The number of agents and total cost

In general, the Clustered-WAN requires fewer agents than the LAN. The WAN is the middle. This result can be explained by the differences in the latencies that exist in each network topology. The latency of inter-LAN communication varies more than intra-LAN latency. The Clustered-WAN is a mixture of LAN and inter-LAN communication, resulting in the biggest time gap between nodes. Under this biased time gap, some agents spend their time on few nodes while others can cover relatively more nodes during the same given amount of time. Notice that there is no difference resulting from using different algorithms in the LAN. The result says that in the LAN configuration, the complex agent planning algorithm does not affect the performance. Thus, in this case, the simplest way to deploy agents may be to send an agent to each node.

In all cases, BYKY2 performs slightly better than BYKY1. They differ in how they find the next node to visit from the current position. BYKY1 always selects the farthest node from home, while BYKY2 selects the nearest node from the current node. In the LAN configuration, both algorithms look similar. However, they show different behaviors between the WAN and Clustered-WAN configurations. The information on the distance of node from home alone may not be sufficient for the calculations, since the new node may not be close to the current node, resulting in longer travel time of the new distance. Consequently, BYKY1 always shows performance worse or equal to that of BYKY2. In a network with n nodes, BYKY1 has time complexity $O(nlogn)$, while for BYKY2 this is $O(n^2logn)$. As the network size grows, the gap in algorithm execution time grows. In contrast, Table 5 shows that as the number of nodes increases, the difference between those two algorithms decreases. Thus, we may be able to use BYKY1, the cheaper one, in certain configurations.

Figure 5 shows the required number of agents for three different algorithms: BYKY1, BYKY2, and Worst. The numbers of agents of BYKY1 and BYKY2 are *35%* and *46%* fewer than for the Worst case on average.

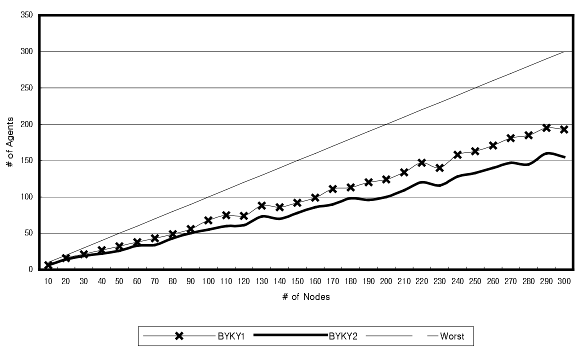

Figure 5. Comparison of the number of agents

The calculated routing costs are depicted in Figure 6. BYKY1 and BYKY2 reduce the cost down to *15%*. An *85%* reduction in total routing cost is a significant enhancement in agent planning technology. In particular, these performance enhancements can be obtained by allowing only a few seconds before launching the agents. The computation times for the algorithms are indicated in Table 5. It shows that BYKY1 and BYKY2 spend *3.43* and *5.92 seconds*, respectively, for the 300-node Clustered-WAN configuration. Compared to the performance a task can attain, these time consumptions are acceptable in every kind of agent planning. As indicated in Figure 6, the results of BYKY1 and BYKY2 are very similar to each other, since the experiment is performed under Clustered-WAN and Complex query-type conditions.

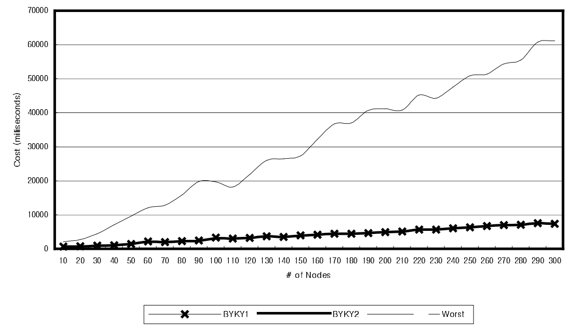

Figure 6. Comparison of the costs

# of nodes	10	20	30	40	50	60	70	80	90	100
BYKY1	0.00	0.00	0.01	0.01	0.02	0.03	0.05	0.07	0.10	0.13
BYKY2	0.00	0.00	0.01	0.02	0.03	0.05	0.08	0.12	0.17	0.23
# of nodes	110	120	130	140	150	160	170	180	190	200
BYKY1	0.16	0.21	0.27	0.33	0.42	0.52	0.62	0.74	0.88	1.03
BYKY2	0.30	0.37	0.50	0.58	0.75	0.90	1.06	1.29	1.50	1.75
# of nodes	210	220	230	240	250	260	270	280	290	300
BYKY1	1.19	1.38	1.57	1.76	2.01	2.26	2.49	2.79	3.07	3.43
BYKY2	2.06	2.40	2.70	3.05	3.48	3.93	4.36	4.86	5.42	5.92

Table 5. The planning time of BYKY1 and BYKY2

In the near future, according to the fruits of the current research on mobile agent technology, a great many network applications will be designed and run using mobile agents. In this world, as extracted from Figure 6, it is clear that the suggested MAP methods can reduce tremendously the burden on the network of the heavy traffic caused by many moving agents.

6. Conclusions

We proposed two MAP algorithms that can be used in a distributed information retrieval system that uses mobile agents. By simulations, we showed that a significant improvement has been made by the proposed cost-effective MAP algorithms. We know the minimum execution time can be obtained when an agent is deployed for each node. The proposed planning algorithms try to find the minimum number of agents, and the total routing cost consumed by those agents while preventing the total execution time from exceeding the minimum. In most cases, algorithm BYKY2 found near-optimal solutions in the number of agents. These solutions drastically cut down the total computation time of the information retrieval system using mobile agent technology.

Experimental parameters were the combinations of network topology, the query type for information retrieval on each node, and the number of nodes. The results show that when all the nodes are in a local area network, regardless of the degree of the problem's complexity, the required number of agents was almost the same as the number of nodes. Consequently, complex planning algorithms do not have a significant effect on the performance in this configuration. In this particular case, good strategy might be launching an agent to each node. In all other cases, as the computation time of d grows, and/or the network latencies are not uniformly distributed, more precise algorithms such as BYKY1 and BYKY2 are highly recommended.

In our experiments, we assumed that the sizes of mobile agents were fixed on each node. We can calculate the latency values including the time needed to transmit the agents. If the size of mobile agent is being increased while retrieval operations are performed, then the effective bandwidth varies from link to link. Since this problem is included in several considerations, we have prepared future works include various planning mechanisms.

7. References

[1] Y. Aridor and M. Oshima. Infrastructure for mobile agents: requirements and design. In *Proc. on Int'l Workshop on Mobile Agents*, 1998.

[2] A. Athan and D. Duchamp. Agent-mediated message passing for constrained environments. In *USENIX Mobile and Location-Independent Computing Symposium*, 1993.

[3] S. Bandyopadhyay and K. Paul, Evaluating the performance of mobile agent-based message communication among mobile hosts in large ad hoc wireless network. In *Proc. of 2nd ACM Int'l Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, August 1999.

[4] H. Braun. On traveling salesman problems by genetic algorithms. In *Workshop on Parallel Problem Solving from Nature*, pages 129-133, 1990.

[5] B. Brewington, R. Gray, K. Moizumi, D. Kotz, G. Cybenko and D. Rus. Mobile agents in distributed information retrieval, In *Intelligent Information Agents*, pages 355-395, 1999.

[6] William R. Cockayne and Michael Zyda. *Mobile Agents*. Manning Publications Co., 1998.

[7] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[8] Ellis Horowitz, Sartaj Sahni. *Fundamentals of Computer Algorithms*. Computer Science Press, 1989.

[9] O. de Kretser, A. Moffat, T. Shimmin, and J. Zobel. Methodologies for distributed information retrieval. In *Proc. of the Eighteenth Int'l Conference on Distributed Computing Systems*, pages 26-29, May 1998.

[10] L. Miller, J. Yang, V. Honavar, and J. Wong. Intelligent mobile agents for information retrieval and knowledge discovery from distributed data and knowledge sources. In *Proc. of the IEEE Information Technology Conference*, 1998.

[11] K. Moizumi. *Mobile Agent Planning Problems*. PhD thesis, Dartmouth College, 1998.

[12] K. Moizumi and G. Cybenko. The Traveling Agent Problem. *Mathematics of Control, Signals and Systems*, January 1998.

[13] G. P. Picco, A. Fuggetta and G. Vigna. Understanding code mobility. *IEEE Transaction on Software Engineering*, January 1998.

[14] G. P. Picco, A. Carzaniga, and G. Vigna. Designing distributed applications with mobile code paradigms. In *Proc. of 19th Int'l Conf. on Software Engineering*, July 1997.

[15] D. Rus, R. Gray, and D. Kotz. Autonomous and adaptive agents that gather information. In *AAAI'96 International Workshop on Intelligent Adaptive Agents*, August 1996.

[16] K. K. Sabnani, T. F. La Porta, T. Woo, and R. Ramjee. Experiences with network-based user agents for mobile applications. *Mobile Networks and Applications*, 1998.

[17] M. P. Wellman, M. Ford, and K. Larson. Path planning under time-dependent uncertainty. In *Proc. of the Eleventh Conference on Uncertainty in AI*, pages 532-539, August 1995.