PAPER

ACE-INPUTS: A Cost-Effective Intelligent Public Transportation System*

Jongchan LEE[†], ***Nonmember***, **Sanghyun PARK**[††a)], ***Member***, **Minkoo SEO**[†††], ***and*** **Sang-Wook KIM**[††††], ***Nonmembers***

SUMMARY With the rapid adoption of mobile devices and location based services (LBS), applications provide with nearby information like recommending sightseeing resort are becoming more and more popular. In the mean time, traffic congestion in cities led to the development of mobile public transportation systems. In such applications, mobile devices need to communicate with servers via wireless communications and servers should process queries from tons of devices. However, because users can not neglect the payment for the wireless communications and server capacities are limited, decreasing the communications made between central servers and devices and reducing the burden on servers are quite demanding. Therefore, in this paper, we propose a cost-effective intelligent public transportation system, ACE-INPUTS, which utilizes a mobile device to retrieve the bus routes to reach a destination from the current location at the lowest wireless communication cost. To accomplish this task, ACE-INPUTS maintains a small amount of information on bus stops and bus routes in a mobile device and runs a heuristic routing algorithm based on such information. Only when a user asks more accurate route information or calls for a "leave later query", ACE-INPUTS entrusts the task to a server into which real-time traffic and bus location information is being collected. By separating the roles into mobile devices and servers, ACE-INPUTS is able to provide bus routes at the lowest wireless communication cost and reduces burden on servers. Experimental results have revealed that ACE-INPUTS is effective and scalable in most experimental settings.

key words: *mobile public transportation system, heuristic routing algorithm, location-based services*

1. Introduction

With the rapid adoption of mobile devices such as cellular phones and PDAs, location based services (LBS) that exploits the location information of moving objects (i.e., mobile devices or automobiles) [3] are being devloped. Finding the location of a person with a specific phone number [19] and recommending a sightseeing resort near to a user [1], [13], [24] are typical examples of the LBS. Recently, there have been many research efforts [10], [14], [16], [19], [21], [23], [26] on *mobile traffic information systems* that incorporate the LBS for more efficient management of road networks or more accurate provision of route information.

In the mean time, as traffic congestion in mega-cities such as New York, Tokyo, and Seoul becomes more serious, *mobile public transportation systems* [2], [4]–[9], [17], [18], [22], which are the mobile traffic information systems for public transportation, are getting more attention. For example, given such a request as "find the bus routes to reach the city hall from the bus station nearest to my current location", mobile public transportation systems may give an answer "at Sinchon bus station, take the bus number 100 or number 200 that will arrive in 7 or 10 minutes, respectively". Most existing mobile public transportation systems, however, have the following drawbacks or limitations.

Firstly, many systems [2], [4], [6], [7], [9], [18], [22] paid little attention to the wireless communication costs of mobile devices and the burdens on a server.

In these systems, a mobile device is responsible only for the transmission of queries and the display of results, and completely entrusts the execution of complex algorithms to a server. Therefore, a mobile device itself can provide very little information about public transportation.

Moreover, the burdens on a server increase as the number of mobile devices being connected to the server increases. Since the cost for wireless communications is still high and the number of mobile devices is increasing rapidly, a scalable mobile public transportation system that charges a low communication cost is quite demanding.

Prefetching [15], [20], [28] is another idea to ease the burden on a server and mobile devices by storing data that is accessed recently or data that will be predicted to be accessed in the near future. However, prefetching do not entirely remove the need for the wireless communication and burden on a server even if the data could be stored in mobile devices compactly in the first place.

Secondly, to the best of authors' knowledge there are no, fairly few if any, routing algorithms which are based on real-time bus information. In contrast to existing shortest path algorithms in graph theory, the bus routing algorithm must consider various factors such as varying speed of buses, waiting time at bus stops, distances between adjacent bus stops, and the number of required bus transfers. However, the existing routing algorithms have limitations in that (1) they are not based on real-time bus and traffic information but only on its statistics [4], [6], [9], [22], [27], and that (2) they are designed only for automobile drivers [14], [16],

Manuscript received June 15, 2006.
Manuscript revised March 23, 2007.

[†]The author is with TmaxSoft Co., Ltd.
[††]Corresponding author. The author is with the Department of Computer Science, Yonsei University, Korea.
[†††]The author is with Google, Korea.
[††††]The author is with College of Information and Communications, Hanyang University, Korea.

[*]This research was partially supported by the MIC, Korea, under the ITRC support program supervised by the IITA (IITA-2005-C1090-0502-0009) and by the research fund of Hanyang University (HY-2005-1).

a) E-mail: sanghyun@cs.yonsei.ac.kr

DOI: 10.1093/ietisy/e90–d.8.1251

[19], [23]. Therefore, it is necessary to study the bus routing algorithms which consider both static (e.g., locations of bus stops) and real-time (e.g., current locations of buses) information.

Lastly, there has been little consideration of *future queries* or *leave later queries*, which are to find the bus routes adequate in the future. Since the algorithms proposed in [6], [9], [27] depend only on the statistics on bus and traffic information, they are not appropriate for future queries. MyBus [18] and BusView [8] are only capable of providing the information about current bus locations and thus not suitable for future queries either. Since it is highly possible for a user to leave for a destination some time after receiving route information, the support for future queries is essential to mobile public transportation systems.

In this paper, we propose a cost-effective intelligent public transportation system, ACE-INPUTS, which utilizes a mobile device to provide a set of bus routes to a destination. To accomplish its task, ACE-INPUTS maintains a small amount of information on bus stops and bus routes in a mobile device, and runs a heuristic routing algorithm based on the information. When a user requests more accurate route information or issues a future query, ACE-INPUTS entrusts the processing of that request to a server into which real-time traffic and bus location information is being collected. By separating the roles into a mobile device and a server, ACE-INPUTS is able to accomplish its task at the lowest cost for wireless communications, without imposing much burden on a server.

This paper is organized as follows. Section 2 surveys related work on routing algorithms for road networks and mobile transportation systems. The overall architecture, user interface, and database schema of ACE-INPUTS are described in Sect. 3. A detailed description of the proposed heuristic routing algorithm is given in Sect. 4. Our proposals are carefully validated in Sect. 5 through experimentations. Finally, concluding remarks follow in Sect. 6.

2. Related Work

The Fast Lee algorithm [14] is one of the representative routing algorithms for road networks. This algorithm computes the optimal route to reach a destination in the shortest time by utilizing the *congestion ratio* of each road, which is defined as the ratio of the current road speed to the maximum road speed. The algorithm, however, is only applicable to automobiles, and therefore is not suitable for public transportation systems we are considering in this paper.

Assuming that bus arrivals follow a Poisson distribution [25], Datar et al. [9] proposed a bus routing algorithm to reach a destination. However, this assumption is not realistic, as admitted by the authors [9]. Based upon the assumption that the probability of bus arrivals becomes higher as time goes by, Boyan et al. [6] extended the distribution of interarrival times to the Normal distribution [25] and the Gamma distribution [25]. However, the algorithm proposed by Boyan et al. [6] requires too much CPU cost and does not exploit real-time bus location information. Q. Wu and J. Hartley [27] proposed K-shortest path algorithms using the time table of buses, and did not incorporate real-time bus location information.

The Bus Catcher [4], [22], one of the popular mobile public transportation systems, computes the optimal bus route by considering various factors such as bus fares, waiting time at bus stops, and time to get to a bus stop on foot. Since the routing algorithm runs entirely in a server, the system increases a server's burden and also makes wireless communication unavoidable. Moreover, it does not support future queries and thus has a limitation in satisfying the various needs of mobile users.

Both MyBus [18] and BusView [8] are web-based applications for displaying current bus locations on electronic maps. Especially, MyBus has been implemented in WAP, so it can be run on a mobile device, too. However, both of them lack the provision of bus routing algorithms and the support of future queries.

3. System Overview

This section describes the system architecture, user interface, and database structure of ACE-INPUTS.

3.1 System Architecture

ACE-INPUTS consists of mobile devices and a server. A mobile database is maintained in each mobile device and a server database is maintained in the server. Figure 1 shows the architecture of ACE-INPUTS and the message flows between a mobile device and the server.

In Fig. 1, dotted lines represent wireless communications that charge costs. Solid lines represent jobs performed entirely on mobile devices. Therefore, communication costs are unnecessary in this case. ACE-INPUTS operates as follows:

(1) A user enters a query with a destination bus stop and maximum number of transfers as arguments, using the interface of a mobile device.
(2) The mobile device runs a heuristic algorithm to retrieve a list of bus routes to the destination and sort those bus routes according to their estimated elapsed times.

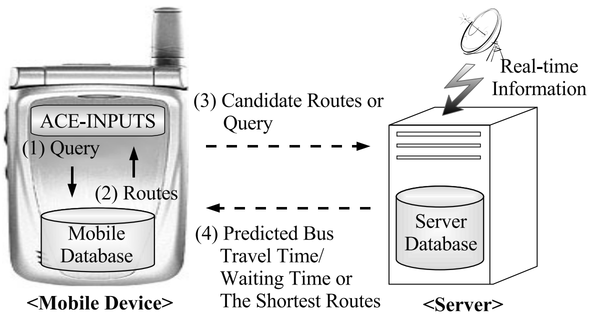
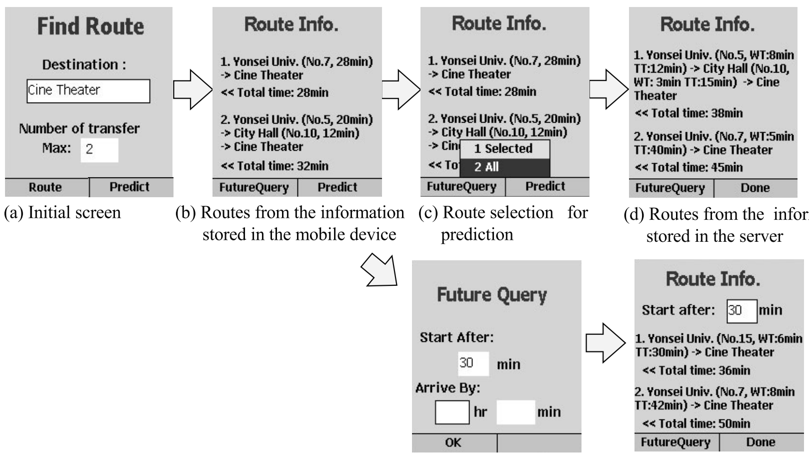


Fig. 1 Overall architecture and message flows in ACE-INPUTS.



Find Route
Destination :
Cine Theater
Number of transfer
Max: 2
Route | Predict

Route Info.
1. Yonsei Univ. (No.7, 28min) -> Cine Theater
<< Total time: 28min
2. Yonsei Univ. (No.5, 20min) -> City Hall (No.10, 12min) -> Cine Theater
<< Total time: 32min
FutureQuery | Predict

Route Info.
1. Yonsei Univ. (No.7, 28min) -> Cine Theater
<< Total time: 28min
2. Yonsei Univ. (No.5, 20min) -> City Hall (No.10, 12min) -> Cin
<< To
1 Selected
2 All
FutureQuery | Predict

Route Info.
1. Yonsei Univ. (No.5, WT:8min TT:12min) -> City Hall (No.10, WT: 3min TT:15min) -> Cine Theater
<< Total time: 38min
2. Yonsei Univ. (No.7, WT:5min TT:40min) -> Cine Theater
<< Total time: 45min
FutureQuery | Done

Future Query
Start After:
30 min
Arrive By:
hr min
OK

Route Info.
Start after: 30 min
1. Yonsei Univ. (No.15, WT:6min TT:30min) -> Cine Theater
<< Total time: 36min
2. Yonsei Univ. (No.7, WT:8min TT:42min) -> Cine Theater
<< Total time: 50min
FutureQuery | Done

(a) Initial screen (b) Routes from the information stored in the mobile device (c) Route selection for prediction (d) Routes from the information stored in the server

(e) Future query setting (f) Future query result

Fig. 2 User interface of ACE-INPUTS.

(3) When a user needs more accurate time information on a specific route, wants to run a routing algorithm using real-time traffic information in a server, or issues a future query at the cost of wireless communications, the mobile device entrusts the processing of that request to the server.
(4) The server uses real-time traffic information to produce more accurate time information on a specific route, a list of bus routes to the destination, or the result of the future query. Then, the result is sent back to the user via wireless communication.

Steps 2 and 3 can be bypassed if a user does not want to execute a heuristic routing algorithm in his or her mobile device but only to perform a server-side routing algorithm directly.

3.2 User Interface

The user interface of ACE-INPUTS is shown in Fig. 2. To make a request, a user first specifies the destination bus stop and maximum number of bus transfers on the screen shown in Fig. 2 (a), and then selects the menu item 'Route'. The screen in Fig. 2 (b) illustrates how a list of bus routes obtained from the proposed heuristic algorithm looks like. If the user wants to get more accurate information, he or she selects the menu item 'Predict' which consists of two sub-items, 'Selected' and 'All'. If the user selects the sub-item 'Selected', then a specific route chosen by the user is sent to the server where its accurate time information is obtained. On the other hand, when the user selects the sub-item 'All', the mobile device sends the destination bus stop and maximum number of bus transfers to the server. Then, the server executes the routing algorithm again using real-time traffic information and sends the result back to the mobile device. The screen in Fig. 2 (d) shows the routes obtained from the server. Here, TT and WT stand for travel time and waiting time, respectively.

Note that the routes in Fig. 2 (d) are ordered differently from those in Fig. 2 (b). This situation may occur because the routing algorithm of a server is based on real-time traffic information unlike the routing algorithm of a mobile device. For example, the route which goes to Cine Theater directly from Yonsei Univ. was suggested as the best plan in Fig. 2 (b), but the route which goes through Yonsei Univ., City Hall, and Cine Theater was ranked as the top in Fig. 2 (d). A user can choose 'FutureQuery' menu item to issue a query like 'will start in 30 minutes' or 'must arrive by 14:20' using the interface of Fig. 2 (e). The result of a future query is displayed as shown in Fig. 2 (f).

3.3 Database Structure

3.3.1 Database in Mobile Devices

To provide the routes to a destination without a wireless communication cost, ACE-INPUTS maintains the information on bus routes and bus stops in mobile devices. Table 1 shows the schema of the two tables where underlined columns represent primary keys. The bus route table stores the identifiers and ordinal numbers of bus stops on each route and the bus stop table collects the names and locations of each bus stop. The travel time between two adjacent bus stops can be estimated by dividing their distance, which is easily obtained from the location information stored in the bus stop table, by the average bus speed.

3.3.2 Database in Server

The server of ACE-INPUTS collects real-time bus location information and stores it into its own database. Given a specific route, ACE-INPUTS utilizes such information to estimate the time for a bus to travel from one bus stop to another and the time for a user to wait at a bus stop. ACE-INPUTS also uses such information to run the server-side routing algorithm. The database maintained in the server consists of three tables: the bus table, bus stop table, and route table. The schema of the bus stop table in the server is

Table 1 Schema of bus route table and bus stop table in mobile devices.

Bus Route Table (M_ROUTE)

Column Name	Description
ROUTE_NUM	Route number
SEQ	Ordinal number of this bus stop in the corresponding route
SID	Bus stop identifier

Bus Stop Table (M_BUS_STOP)

Column Name	Description
SID	Bus stop identifier
LOC_X	X coordinate of the location of this bus stop
LOC_Y	Y coordinate of the location of this bus stop
SNAME	Bus stop name

Table 2 Schema of bus table and bus stop table in server.

Bus Table (S_BUS)

Column Name	Description
BUS_ID	Bus Identifier
ROUTE_NUM	Route number
TIMESTAMP	Time at which the location information is received
LAST_SID	Identifier of the previous bus stop
TIME_FROM_LAST_BS	Time elapsed after departing the previous bus stop
LOC_X	X coordinate of the current bus location
LOC_Y	Y coordinate of the current bus location

Bus Route Table (S_ROUTE)

Column Name	Description
ROUTE_NUM	Route number
SEQ	Ordinal number of this bus stop in the corresponding route
SID	Bus stop identifier
TIMESTAMP	Time slot to which PREDICTED_TT is applied
PREDICTED_TT	Predicted travel time to the next bus stop

exactly same as the one in a mobile device. Table 2 shows the schema of other two tables. The bus table keeps track of the current locations of buses. That is, whenever the server receives new information on the current location of a bus, it retrieves and then updates the corresponding tuple. More specifically, the server assigns the time at which it receives the information to column TIMESTAMP, XY coordinates of the current bus location to columns LOC_X and LOC_Y, the identifier of the bus stop visited last by this bus to column LAST_SID, and the time elapsed after departing the previous bus stop to column TIME_FROM_LAST_BS.

The bus route table maintains the estimated travel times between adjacent bus stops for each time slot of a day, in addition to storing the identifiers and ordinal numbers of bus stops on each route. Column TIMESTAMP specifies the time slot to which column PREDICTED_TT applies. As an example, suppose that a day is divided into 144 time slots of 10 minutes each. If it is expected that a bus with route number 5 takes 7 minutes to reach the second bus stop from the first one in time slot [9:00, 9:10) and 5 minutes in time slot [9:10, 9:20), then two tuples, (ROUTE_NUM, SEQ, SID, TIMESTAMP, PREDICTED_TT) = {(5, 1, STOP_1, 9:00, 7), (5, 1, STOP_1, 9:10, 5)}, will be stored in the bus route table.

4. Query Processing

This section presents query processing algorithms in ACE-INPUTS. Section 4.1 describes the routing algorithm running in mobile devices, and Sect. 4.2 reports the routing algorithm in a server. Section 4.3 explains the method to estimate the time required for a bus to move between two adjacent bus stops. Section 4.4 presents a method to process future queries.

4.1 Routing Algorithm in Mobile Devices

ACE-INPUTS finds the routes to a destination by using only the information stored in mobile databases. To this aim, it creates a *directed multi-graph*[†] *G* from information on bus routes and bus stops, sets starting and destination nodes, and then calls the routing algorithm. In the directed multi-graph *G*, a node represents a bus stop and a directed edge models a move between two adjacent bus stops. Since a node stores all the route numbers of the buses going by way of its corresponding bus stop, multiple edges may exist between any two nodes. A directed edge also stores the time required for the bus to move between bus stops as its weight.

[†]A directed multi-graph is a graph that allows multiple directed edges between any two adjacent nodes [11].

```
Input  : Directed multi-graph $G$, Start node $S$, Destination
         node $D$, Maximum number of transfers $MT$
Output: Set of routes $R$
1  SMALLEST_COST := $\infty$;
2  priority_stack.push($S$);
3  while priority_stack.not_empty() do
4      $V$ := priority_stack.pop();
5      if $V$ is $D$ then
6          $R$.add($V$.current_route);
7          continue;
8      end
9      ADJ := $V$.get_adjacent_nodes($G$);
10     foreach $V'$ in ADJ do
11         $V'$.compute_num_transfers($V$);
12         if $V'$.num_transfers > $MT$ then
13             continue;
14         end
15         EC := $V'$.estimate_cost($V$, $D$);
16         if EC > (BETA $*$ SMALLEST_COST) then
17             continue;
18         else if EC < SMALLEST_COST then
19             SMALLEST_COST := EC;
20         end
21         $V'$.compute_angle_of_vectors($V$, $S$, $D$);
22         priority_stack.push($V'$);
23     end
24 end
25 $R$.sort();
26 return $R$;
```

Algorithm 1: Find_Routes, a routing algorithm in mobile devices.

4.1.1 Routing Algorithm

Algorithm 1 shows Find_Routes, the routing algorithm proposed in this paper. It takes a directed multi-graph G, a starting bus stop S, a destination bus stop D, the maximum number of transfers MT as input arguments, and returns R, a set of routes from S to D, in the ascending order of the estimated times spent on those routes. By considering a limited amount of resources in mobile devices, this algorithm employs the priority stack for its execution. The priority stack preferentially visits the nodes with small estimated costs over others, thereby reducing the search space significantly compared with the queue. For faster execution, the proposed algorithm does not search all the possible routes to a destination. Instead, it cuts off such routes, which have a low possibility to belong to the final result, at an earlier stage. That is, by searching the graph in the best-first fashion using the priority stack, it poses a tighter condition on visiting nodes as the algorithm proceeds, and therefore reduces the search space remarkably.

The priority stack has a constraint that a node in a level has a cost smaller than those in its lower levels. As in ordinary stacks, the priority stack takes the uppermost node for its pop operation. For a push operation, unlike ordinary stacks, the priority stack inserts a new node into an appropriate position in the stack by reflecting its cost.

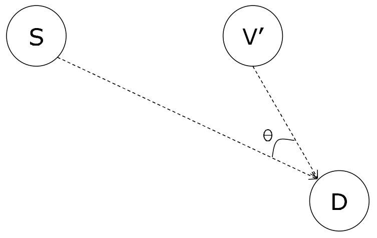


Fig. 3 Angle θ between two vectors $\overrightarrow{SD}$ and $\overrightarrow{V'D}$.

Algorithm Find_Routes starts from S, and searches a directed multi-graph G in the best-first fashion. Let V be the node currently visited. Find_Routes first checks whether V is a destination. If V is not the destination, it performs the following steps on each V', which is adjacent to V.

(1) It computes the number of transfers in the current route, assuming that we move to V' from V. If the number of transfers exceeds MT, it takes the next node to be visited by popping the priority stack.
(2) It estimates the cost for the current route, under the assumption that we move to V' from V. The function for cost estimation will be discussed in the next subsection. If the estimated cost becomes larger than BETA $*$ SMALLEST_COST, it pops the next node from the priority stack for proceeding the traverse. This is because there is little possibility of reaching D with the current route to V' in a reasonable cost when the estimated cost at this point is larger than BETA $*$ SMALLEST_COST. SMALLEST_COST is the smallest one among the costs of the routes to D found up to now. Thus, Find_Routes continues to take the route to V' only when its estimated cost is smaller than or equal to BETA $\times$ SMALLEST_COST. BETA is a real number no smaller than one, which is set by considering the tradeoff between the accuracy and the search performance.
(3) As shown in Fig. 3, it also computes the angle between the vector from S to D and the vector from V' to D. If this angle is small, it implies that V' is located close to the direct route from S to D.
(4) It pushes V' into the priority stack. As stated earlier, the node with a smaller estimated cost resides in a higher level. In case two nodes have the same estimated cost, the node with a smaller angle computed in step (3) resides in the upside. This is to reflect a heuristic that, among the two nodes with the same estimated cost, it is better to choose the one located close to the direct route from S to D.

4.1.2 Cost Estimation Function

Suppose we move from a node V to a node V' in a current route. We estimate the cost of the route by using the following function, which considers distances, moving speeds, and transfers.

```
$V'.estimate\_cost(V, D)$ =
1 : $V.current\_cost$ +
2 : $V.distance\_to(V')$ / $avg\_moving\_speed$
3 : $V'.distance\_to(D)$ / $avg\_moving\_speed$
4 : $V.does\_need\_to\_transfer(V') * avg\_trans\_time$
```

In the function, line 1 represents the time spent from the starting bus stop to V via the current route. Line 2 is to estimate the time spent for moving from V to V' by the formula that divides the distance between V and V' by the average moving speed of the bus. Line 3 is to estimate the minimum time spent for moving from V' to D by dividing the distance of the direct route from V' to D by the average moving speed of the bus. Line 4 adds the average time for transfer to the total time in case a transfer is needed in V'.

4.2 Routing Algorithm in a Server

When a user requests, ACE-INPUTS sends one of two query types to a server. The first type is to request more accurate information on a selected route, which has been obtained in a mobile device. The server computes the time for a user to wait for buses in a bus stop, the time for a bus to move between two bus stops, and the total time required for reaching a destination by using real-time information on bus locations stored in the bus table and the estimated time for moving pairs of two adjacent bus stops stored in the bus route table.

The second type is to send a user query including the starting bus stop, the destination bus stop, and the maximum number of transfers to a server. The server first creates a directed multi-graph G by using the bus route table and the bus stop table. Then, it finds the routes to a destination by using the routing algorithm same as in Sect. 4.1. The cost function is defined to be a summation of the moving times for pairs of bus stops and the times for waiting in bus stops for transfers. The result is more accurate than that obtained from a mobile device in that it is computed by using the transportation information gathered in real time.

The routing algorithm in the server is identical to that in mobile devices except for the cost function. The server has a computing power much higher than mobile devices. This makes it possible to employ a larger BETA in the routing algorithm, thereby improving the accuracy of the routing algorithm by trading the overhead of searching larger space in G.

4.3 Estimating a Moving Time between a Pair of Bus Stops

The moving time between a pair of adjacent bus stops is affected by various factors such as road conditions, seasons, days of the week, and time slots of a day. In this paper, we estimate the moving time between a pair of bus stops by using the *exponential moving average* [12], one of history data based approaches. Let P_n and T_n the nth estimated and actually measured times, respectively. P_{n+1}, the $(n+1)$th estimated time, is computed as follows.

$$P_0 = T_0 \tag{1}$$

$$P_{n+1} = \alpha T_n + (1-\alpha)P_n \quad (n \geq 0,\ \ 0 \leq \alpha \leq 1) \tag{2}$$

α is a parameter that determines how much the nth actually measured time is reflected in the $(n+1)$th estimated time. A larger α makes P_{n+1} place more importance on T_n, the actually measured value, than P_n, an accumulated history. In this paper, we assume that the moving time is measured at every 10 minute. At each time, P_{n+1} is computed by using Eq. (2), and then is stored in the column PRETICTED_TT within the bus route table. Also, we can estimate the time for a user to wait in a bus stop by referring to PREDICTED_TT in the bus route table and TIME_FROM_LAST_BS in the bus table.

4.4 Future Query Processing

The future query is classified into two types. The first type specifies the *time at which a user will start for a destination*, and the second type specifies the *time by which a user has to arrive at a destination*. For processing of these two types of queries, a server uses PREDICTED_TT in the bus route table and TIME_FROM_LAST_BS in the bus table. For instance, for processing a query of "which is the optimal route to City Hall from our school when we start after 15 minutes?", the server performs the routing algorithm based on the expected locations of buses after 15 minutes and the expected moving times between pairs of adjacent bus stops at that time. The future query of the second type can be also handled in a similar way.

5. Experiments

In this section, we describe the results of experiments to verify the performance of ACE-INPUTS in terms of the efficiency and accuracy of the proposed algorithm. In Sect. 5.1, we explain the environment for experiments as well as the method for generating data and queries, and define the measure for evaluating the accuracy of a result set. In Sect. 5.2, via preliminary experiments, we determine the optimal value of BETA, which affects both the efficiency and the accuracy of ACE-INPUTS considerably. In Sect. 5.3, we extensively evaluate the performance of ACE-INPUTS using the value of BETA thus obtained.

5.1 Experimental Environment

5.1.1 System Settings

We carried out experiments using PC with Pentium Mobile Centrino CPU of 1.0 GHz and 512 M memory. We used the JDK 1.4.2 version for implementation and the MySQL version 4.1.9 for database construction.

5.1.2 Method for Generating Data and Queries

For evaluating the performance of ACE-INPUTS, it is ideal to use the real-world data, for example, the information on real-world bus stops and routes. However, it is actually too difficult to collect all of these data. Therefore, we generated synthetic data that reflects the characteristics of real-world one well.

(1) **Getting bus stops:** Assume that N is the total number of bus stops and that the intervals between any two adjacent bus stops have similar distances. We first divided the space into N grid points using $\sqrt{N}$ vertical and $\sqrt{N}$ horizontal lines[†]. The intervals between any two adjacent lines were set to 100 m. Then, we located a bus stop $b_{i,j}$ on the each grid point ($1 \leq i, j \leq \sqrt{N}$). Finally, to give variations to the intervals, we randomly moved each bus stop inside of the circle of radius 25 m as in Fig. 4. Each black point represents a bus stop moved from its corresponding grid point.

(2) **Generating bus routes:** First of all, we selected two arbitrary bus stops for the start and the destination of a route. The distance between the start and the destination should not be too short. To the end, we made a constraint that the distance between the start and the destination must be at least $\sqrt{N}/2$ in each axis. That is, we determined the two locations of $b_{sx,sy}$ and $b_{ex,ey}$ within the range satisfying the two conditions shown below.

$$e_x \leq s_x - \sqrt{N}/2 \quad \text{or} \quad e_x \geq s_x + \sqrt{N}/2 \tag{3}$$
$$e_y \leq s_y - \sqrt{N}/2 \quad \text{or} \quad e_y \geq s_y + \sqrt{N}/2 \tag{4}$$

After deciding the locations of the start and the destination, we chose intermediate bus stops of that route. For this, we selected one of the bus stops located in the eight different directions around the start $b_{sx,sy}$, which are $b_{sx\pm1,sy}$, $b_{sx,sy\pm1}$, and $b_{sx\pm1,sy\pm1}$, and assigned it to the first intermediate bus stop as in Fig. 5. We selected the subsequent intermediate bus stops in a recursive way until arriving at the destination. We did not choose such bus stops chosen earlier for this bus route. So, we could determine all the intermediate bus stops, and thus completed the route.

In a real world, a route tends to direct intermediate bus stops towards its destination. So, it is not appropriate to presume that the bus stops in eight directions have the same probability to be chosen as an intermediate bus stop. For example, let V and D be current and destination bus stops, respectively, as in Fig. 6. If we selected 1, 4, 6, 7 or 8 as a next bus stop, the route got too far from the destination, which disagrees with the real situation. To resolve this problem, we set the probability to select 2, 3, and 5 higher than the others. In our experiments, we set 70% for 2, 3, and 5, and 30% for 1, 4, 6, 7, and 8.

(3) **Generating queries:** A user query in ACE-INPUTS is composed of three arguments: the start bus stop, the destination bus stop, and the maximum number of transfers. The maximum number of transfers was set to one out of 0, 1, and 2 depending on the purpose of experiments. The start and destination bus stops were determined as follows: A bus stop was randomly chosen among N bus stops as a start. Then, a destination was selected randomly among those far from the start by the distance between $\sqrt{N}/3$ and $\sqrt{N}/2$.

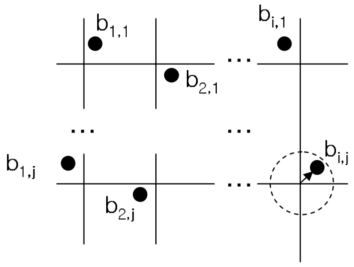


Fig. 4 Determining the locations of bus stops.

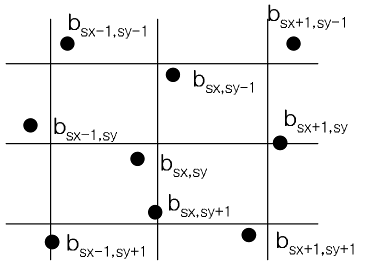


Fig. 5 Eight bus stops around the starting one $b_{sx,sy}$.

5.1.3 Measure to Evaluate the Accuracy

For performance evaluation, we used the size of a result set, the query processing time, and the accuracy of a result set as performance factors. The size of a result set indicates the number of routes produced by the system after query processing. The query processing time indicates the time spent for receiving the result after querying. The routing algorithm proposed in this paper prunes such routes having low probability to be included in the final result in an early stage in order to achieve faster execution owing to reduced search space. This may cause to miss some routes in the final result. So, we need to evaluate the accuracy of the

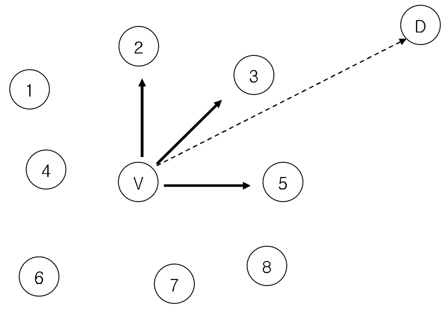


Fig. 6 Selecting an intermediate bus stop considering the route direction.

[†]Without loss of generality, we assume that $\sqrt{N}$ is an integer number for simplicity.

proposed algorithm.

To evaluate the accuracy of a result set produced by pruning, we need to use the result set obtained from the exhaustive search as a true result set. Let us denote *PathSetE* and *PathSetH* the result sets obtained by the exhaustive search and our pruning method, respectively. To evaluate the accuracy of *PathSetH*, $RawScore(P_i, PathSetE)$ and $NormScore(P_i, PathSetE)$, which are the original and normalized scores, are computed for each route P_i contained in *PathSetE* as follows.

$$RawScore(P_i, PathSetE) = \frac{\sum_{P_j \in PathSetE} T(P_j) - T(P_i)}{\sum_{P_j \in PathSetE} T(P_j)} \tag{5}$$

$$NormScore(P_i, PathSetE) = \frac{RawScore(P_i, PathSetE)}{|PathSetE| - 1} \times 100 \tag{6}$$

In Eq. (5) and Eq. (6), $T(P_i)$ denotes the expected elapsed time of route P_i and $|PathSetE|$ dose the number of routes in *PathSetE*. $RawScore(P_i, PathSetE)$ has a value between 0 and 1, and becomes bigger as $T(P_i)$ gets smaller. The sum of the raw scores of all routes in *PathSetE* is equal to $|PathSetE| - 1$. *RawScore* is normalized so that the sum of the scores of all the routes becomes 100 (Eq. (6)). As in Eq. (7), the accuracy of *PathSetH* is represented as the sum of the normalized scores for all the routes P_i in *PatheSetH*. If *PathSetH* is equal to *PathSetE*, its accuracy becomes 100. If *PathSetH* does not contain any route in *PathSetE*, the accuracy becomes 0.

$$Accuracy(PathSetH) = \sum_{P_i \in PathSetH} NormScore(P_i, PathSetE) \tag{7}$$

5.2 Determining BETA

BETA is an important factor that regulates the number of routes pruned in the proposed method. With a small value of BETA, a large number of routes get pruned. So, the value of BETA significantly affects the efficiency and the accuracy of the proposed algorithm.

Before evaluating the performance of ACE-INPUTS extensively, this section determines an optimal value of BETA through a preliminary experiment. For this experiment, we generated 200 bus stops and 100 bus routes using the same method as described in Sect. 5.1. Also, we generated three sets of one hundred queries each, each of which allows the maximum number of transfers to be 0, 1, and 2, respectively. With increasing BETA from 1 to 10, we computed the size of a result set, the query processing time, and the accuracy of a result set for each pair ⟨the maximum number of transfers, BETA⟩.

5.2.1 Size of a Result Set

Figure 7 shows the average size of a result set. As expected,

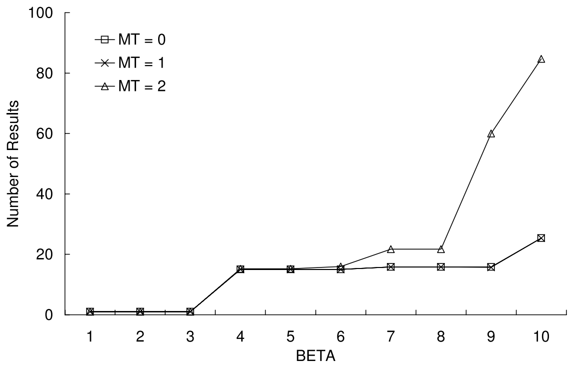


Fig. 7 Average size of a result set as BETA increases.

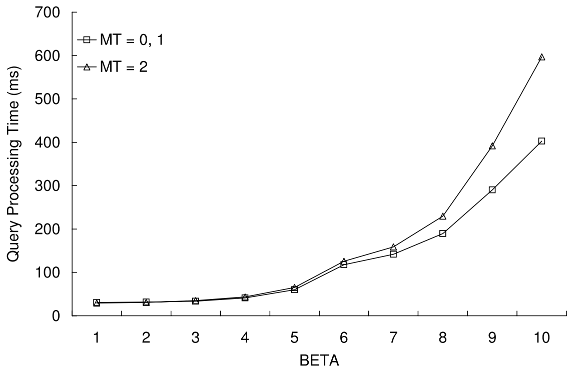


Fig. 8 Average query processing time as BETA increases.

the size of a result set increases as BETA gets bigger. The size of a result set is about 1 on average when BETA has a value between 1 and 3. When BETA has a value between 4 and 6, it is around 15. If only one route is recommended, users are not likely to be satisfied. Therefore, we conclude that BETA should be at least 4 from the perspective of the size of a result set.

5.2.2 Query Processing Time

Figure 8 shows the average query processing time with increasing BETA when the maximum number of transfers MT is 0, 1, and 2. When MT is 0 and 1, the results are almost the same, so we represent them with a single graph. As in the previous experiment on the size of a result set, the average query processing time increases as BETA grows. When BETA is in the range of 1 to 5, the average query processing time increases gradually regardless of MT. However, with increasing BETA larger than 6, it increases dramatically. Also, the gap of average query processing times for MT of 1 and 2 gets much larger. Overall, the average query processing time is less than one second in all cases. In particular, with BETA less than or equal to 5, it is less than 100 milliseconds.

5.2.3 Accuracy of a Result Set

Figure 9 shows the accuracy of a result set as BETA in-

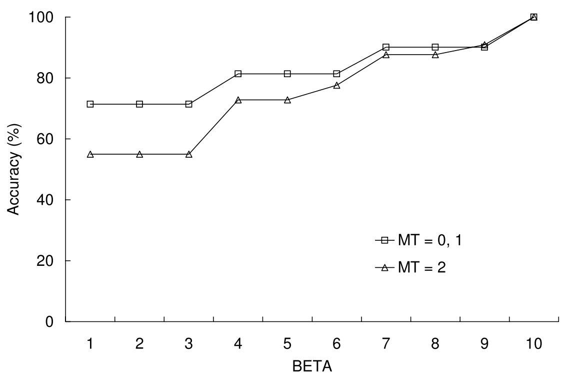


Fig. 9 Accuracy of a result set as BETA increases.

creases when MT is 0, 1, and 2. With MT of 0 and 1, the results are almost the same, so we represent them with a single graph. As BETA increases, the accuracy of a result set becomes higher. A larger BETA makes the pruning effect get less, and examines more routes. If we look into Fig. 9 in more detail, the increase of the accuracy is remarkable when BETA changes from 3 to 4 or 6 to 7. In conclusion, if we want to keep the accuracy more than 70% regardless of MT, we should set BETA to be at least 4.

5.2.4 Deciding BETA

The results of the previous experiments are summarized as follows. First, if we only consider the size of a result set, we must set BETA to be at least 4 regardless of MT. Second, if we only consider the average query processing time, we must set BETA to be at most 5 regardless of MT. Third, if we want to achieve an accuracy more than 70% regardless of MT, we must set BETA to be at least 4. Thus, we conclude that only BETA of 4 or 5 satisfies the three conditions above. In the next experiments, we employed BETA of 4.

5.3 Efficiency and Accuracy of the Proposed Algorithm

In this section, we evaluate the efficiency and the accuracy of the routing algorithm proposed in this paper with BETA determined from the previous section. For these experiments, with the increasing number of bus stops from 200 to 1000, we measured the accuracy of a result set and the average query processing time of algorithm Find_Routes and algorithm Find_Routes_No_Pruning. Algorithm Find_Routes denotes the routing algorithm proposed in this paper, and algorithm Find_Routes_No_Pruning denotes the routing algorithm obtained by removing the pruning code (from line 15 to line 17 in Algorithm 1) from algorithm Find_Routes. The number of bus routes was set to half of the number of bus stops, and the maximum number of transfers MT was fixed to 2. BETA was set to 4 as described in Sect. 5.2.

5.3.1 Efficiency

Figure 10 shows the tendency of the average query processing times of algorithm Find_Routes and algorithm

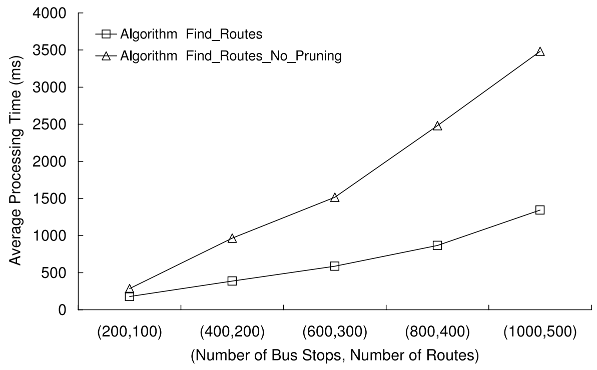


Fig. 10 Average query processing time as a data size increases.

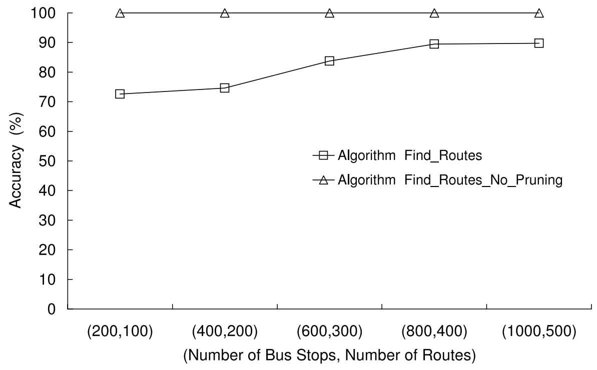


Fig. 11 Accuracy of a result set as a data size increases.

Find_Routes_No_Pruning performed with different data sizes. Algorithm Find_Routes shows a response time less than 1.5 second in all cases, and runs two or three times faster than algorithm Find_Routes_No_Pruning. Moreover, as a data size gets bigger, the gap of their average query processing times increases gradually. We can observe that algorithm Find_Routes proposed in this paper is efficient as well as scalable.

5.3.2 Accuracy

Figure 11 shows the tendency of the accuracy of a result set obtained from algorithm Find_Routes and algorithm Find_Routes_No_Pruning performed with different data sizes. Algorithm Find_Route_No_Pruning, which performs no pruning, always shows 100% accuracy regardless of data sizes, because it examines all the candidate routes and thus does not miss any true answers in the result set. Algorithm Find_Routes shows accuracy of minimum 73%, maximum 90%, and average 82%. Also, the accuracy increases slightly as a data size grows.

5.4 Schema Evaluation

Questions may arise whether the proposed schemas are properly designed. Hence, we have compared alternative design of bus route table and bus stop table in mobile devices shown in Table 3 with the proposed schema shown in

Table 3 Alternative schemas of bus route and bus stop table in mobile devices.

Bus Route and Bus Stop Table (M_ALT_ROUTE_BUS_STOP)

Column Name	Description
<u>ROUTE_NUM</u>	Route number
<u>SEQ</u>	Ordinal number of this bus stop in the corresponding route
<u>SID</u>	Bus stop identifier
LOC_X	X coordinate of the location of this bus stop
LOC_Y	Y coordinate of the location of this bus stop
SNAME	Bus stop name

Table 4 Table sizes of proposed and alternative schemas.

Data Size	Proposed (in Bytes)	Alternative (in Bytes)
(200, 100)	39,344	60,656
(400, 200)	117,952	199,104
(600, 800)	205,472	357,952
(800, 400)	332,306	592,484
(1000, 500)	448,840	809,268

Table 5 Average database access time of proposed and alternative schemas.

Data Size	Proposed (in ms)	Alternative (in ms)
(200, 100)	0.97	0.94
(400, 200)	1.84	1.94
(600, 800)	4.84	4.79
(800, 400)	7.34	7.30
(1000, 500)	9.70	9.76

Table 1.

In this evaluation, a Intel quad-core 2.4 GHz CPU, 2 GB memory, 300 G(7200 rpm) SCSI HDD, and Ubuntu Dapper installed computer was used.

The size and the query processing speed of the proposed schemas and the alternative schemas were measured while changing the data size. Table 4 describes the size of the tables in bytes.

As shown in the table, proposed schema is more storage efficient due to normalization. Moreover, it should be pointed out that the alternative schema has serious drawback of insertion/deletion/update anomaly. As an example, suppose that there is a bus stop being built, but there are no routes yet which goes through it. In this case, alternative schema can not represent such a bus stop because ROUTE_NUM and SEQ columns are primary keys which can not be NULL. Such a situation can happen because there is need for letting people know that a bus stop will be in action in the future, but no bus route is available until then.

Table 5 shows average database access time of the line #9 of Algorithm 1 measured in milliseconds. Note that other lines of the algorithm do not necessarily need database access.

As shown in the table, alternative schema performs more efficiently in terms of time. This is due to the denormalization which decreased the join between M_ROUTE and M_BUS_STOP of proposed schema. However, the amount of gain was not noticeable.

6. Conclusions

In this paper, we have proposed a cost-effective intelligent public transportation system, ACE-INPUTS, which utilizes a mobile device to provide a set of bus routes to reach a destination at the lowest cost for wireless communications. The major contributions of our work are summarized as follows:

(1) The effective and scalable architecture which enables distributing the burden of routing algorithm execution among mobile devices and a server is proposed. The architecture reduces communication cost and burden on the server.
(2) The heuristic routing algorithm which is able to run on a small amount of information on bus stops and routes are designed and evaluated. Especially, experiments have shown that the proposed system can answer a user query within 1.5 seconds with approximately 82% accuracy even if the data set size increases.
(3) The solution for supporting future queries, which have been rarely considered so far in the literature, is provided in order to serve various needs of mobile customers.
(4) The systematic method to generate synthetic data using real-world bus and traffic statistics is discussed.

As a further study, we have a plan to extend our work in the following directions: (1) combining bus and subway routes, (2) recommending the optimal bus stop for departure by considering the locations of a user, bus stops, and buses, (3) allowing a few minutes walk for transfer, and (4) managing user profiles that store individuals' preferences regarding time slots and fares, and incorporating them into the routing algorithm.

References

[1] G. Abowd, C. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton, "Cyberguide: A mobile context-aware tour guide," Wirel. Netw., vol.3, no.5, pp.421–433, 1997.
[2] M. Banatre, P. Couderc, J. Pauty, and M. Becus, "Ubibus: Ubiquitous computing to help blind people in public transport," Mobile-HCI, pp.310–314, 2004.
[3] B. Benatallah and Z. Maamar, "Introduction to the special issue on m-services," IEEE Trans. Syst. Man Cybern., A, Syst. Humans, vol.33, no.6, pp.665–666, 2003.
[4] M. Bertolotto, G. O'Hare, R. Strahan, A. Brophy, A. Martin, and E. McLoughlin, "Bus catcher: A context sensitive prototype system for public transportation users," IEEE Intl. Conf. on Web Information Systems Engineering, pp.64–72, 2002.

[5] P. Borne, B. Fayech, S. Hammadi, and S. Maouche, "Decision support system for urban transportation networks," IEEE Trans. Syst. Man Cybern., C, Appl. Rev., vol.33, no.1, pp.67–77, 2003.

[6] J. Boyan and M. Mitzenmacher, "Improved results for route planning in stochastic transportation networks," 12th Annual ACM-SIAM Symposium on Discrete Algorithms, pp.895–902, 2001.

[7] D. Chu, C. Song, B. Zhang, and M. Humphrey, "Uvabus.net: Enhancing user experiences on smart devices through context-aware computing," IEEE Conf. on Consumer Communications and Networking, pp.511–515, 2004.

[8] D.J. Dailey, F.W. Cathey, and S.D. Maclean, "Design and realization of multi-modal/multi-agency transit management and information system," IEEE Conf. on Intelligent Transportation Systems, pp.1664–1669, 2003.

[9] M. Datar and A. Ranade, "Commuting with dealy prone buses," 11th Annual ACM-SIAM Symposium on Discrete Algorithms, pp.22–29, 2000.

[10] J.F. Dillenburg, O. Wolfson, and P.C. Nelson, "The intelligent travel assistant," Intl. Conf. on Intelligent Transportation Systems, pp.691–696, 2002.

[11] http://www.utm.edu/departments/math/graph/glossary.html

[12] http://www.investorwords.com/5561/exponential_moving_average.html

[13] O. Eriksson, "Location based destination information for the mobile tourist," Intl. Conf. on Information and Communication Technology in Tourism, pp.22–25, 2002.

[14] J. Fawcett and P. Robinson, "Adaptive routing for road traffic," IEEE Comput. Graph. Appl., vol.20, no.3, pp.46–53, 2000.

[15] H. Hu, J. Xu, W.S. Wong, B. Zheng, D.L. Lee, and W.-C. Lee, "Proactive caching for spatial queries in mobile environments," IEEE Intl. Conf. on Data Engineering, pp.403–414, 2005.

[16] THINKWARE Systems Corp. http://www.thinkwaresys.com/

[17] C.W. Jenks, "Applications of intelligent transportation systems to public transportation in europe," TCRP Research Results Digest, 1998.

[18] S.D. Maclean and D.J. Dailey, "Real-time bus information on mobile devices," IEEE Intl. Conf. on Intelligent Transportation Systems, pp.25–29, 2001.

[19] SK Telecom NATE Service. http://www.nate.com/

[20] W.-C. Peng and M.-S. Chen, Design and performance studies of an adaptive cache retrieval scheme in a mobile computing environment, IEEE Trans. Mobile Computing, vol.4, no.1, pp.29–40, 2005.

[21] S.Q. Shi, X. Gong, and H. Mo, "Study on real time traffic flow routing algorithm based on case based reasoning," Intl. Conf. on Intelligent Transportation Systems, pp.163–167, 2003.

[22] R. Strahan, C. Muldoon, G.M.P. O'Hare, M. Bertolotto, and R.W. Collier, "An agent-based architecture for wireless bus travel assistants," Wireless Information Systems, pp.54–62, 2003.

[23] The TomTom Website. http://www.TomTom.com/

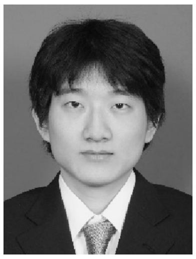
[24] S. Tseng and C. Tsui, "Mining multilevel and location-aware service patterns in mobile web environments," IEEE Trans. Syst. Man Cybern., B, Cybern., vol.34, no.6, pp.2480–2485, 2004.

[25] R.E. Walpole, R.H. Myers, S.L. Myers, and K. Ye, Probability and Statistics for Engineers and Scientists, 7th ed., Prentice Hall, 2002.

[26] F.-Y. Wang, S. Tang, Y. Sui, and X. Wang, "Toward intelligent transportation systems for the 2008 olympics," IEEE Intell. Syst., vol.18, no.6, pp.8–11, 2003.

[27] Q. Wu and J. Hartley, "Accommodating user preferences in the optimization of public transport travel," Int. J. Simulation Systems, Science & Technology: Applied Modelling & Simulation, vol.5, no.3-4, pp.12–25, 2004.

[28] L. Yin, G. Cao, C. Das, and A. Ashraf, "Power-aware prefetch in mobile environments," in Distributed Computing Systems, pp.571–578, 2002.



Jongchan Lee received B.S. and M.S. degrees in Computer Science from Yonsei University, Korea in 2004 and 2006, respectively. His research interests include location based system (LBS), bioinformatics, data mining and multimedia database.

Sanghyun Park received B.S. and M.S. degrees in Computer Engineering from Seoul National University, Korea in 1989 and 1991, respectively, and earned the Ph.D. degree in Computer Science from UCLA. His research interests include bioinformatics, secure data and content management, stream data mining and embedded database.

Minkoo Seo received B.S. degree in Computer Science from Yonsei University, Korea in 2004 and M.S. in Computer Science from Korea Advanced Institute of Science and Technology (KAIST) in 2007. His interests lies in database systems, information retrievals, and computer systems for storing and disseminating information.

Sang-Wook Kim received the B.S. degree in Computer Engineering from Seoul National University, Korea in 1989, and earned the M.S. and Ph.D. degrees in Computer Science from Korea Advanced Institute of Science and Technology (KAIST) in 1991 and 1994, respectively. His research interests include storage systems, transaction management, main-memory DBMSs, embedded DBMSs, data mining/data warehousing, multimedia information retrieval, and geographic information systems.