Three Effective Top-Down Clustering Algorithms for Location Database Systems

Kwang-Jo Lee and Sung-Bong Yang
Department of Computer Science, Yonsei University, Seoul, Republic of Korea
{kjlee5435, yang}@cs.yonsei.ac.kr

Received 10 May 2010; Revised 24 May 2010; Accepted 27 May 2010

Recent technological advances in mobile communication systems have made explosive growth in the number of mobile device users worldwide. One of the most important issues in designing a mobile computing system is location management of users. The hierarchical systems had been proposed to solve the scalability problem in location management. The scalability problem occurs when there are too many users for a mobile system to handle, as the system is likely to react slow or even get down due to late updates of the location databases. In this paper, we propose a top-down clustering algorithm for hierarchical location database systems in a wireless network. A hierarchical location database system employs a tree structure. The proposed algorithm uses a top-down approach and utilizes the number of visits to each cell made by the users along with the movement information between a pair of adjacent cells. We then present a modified algorithm by incorporating the exhaustive method when there remain a few levels of the tree to be processed. We also propose a capacity constraint top-down clustering algorithm for more realistic environments where a database has a capacity limit. By the capacity of a database we mean the maximum number of mobile device users in the cells that can be handled by the database. This algorithm reduces a number of databases used for the system and improves the update performance. The experimental results show that the proposed, top-down, modified top-down, and capacity constraint top-down clustering algorithms reduce the update cost by 17.0%, 18.0%, 24.1%, the update time by about 43.0%, 39.0%, 42.3%, respectively. The capacity constraint algorithm reduces the average number of databases used for the system by 23.9% over other algorithms.

Categories and Subject Descriptors: System & Architecture

General Terms: Network and Communication

Keyword: Location Database, Top-down Clustering, Location Management

1. INTRODUCTION

The world is experiencing a dramatic increase in the number of mobile device users, owing mainly to the technological advances in mobile devices and wireless data

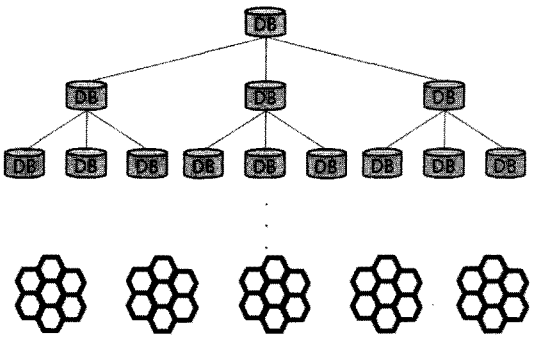


Figure 1. A hierarchical Location Database System.

networking (WIBRO, 3G LTE, and 4G) [Frattasi et al. 2006; Nam et al. 2008; Zheng et al. 2008]. One of the important issues in designing a mobile computing system is location management. It is necessary to manage users' location information as efficiently as possible, since users move around the network and their current locations should be updated in the databases. Especially when there are many users in a network, the mobile system suffers from the scalability problem. By *scalability*, we mean "the ability of a network to adjust or maintain its performance as the size of the network increases (and the demands made upon it increases), yet the performance of a network tends to degrade as the number of mobile users increases" [Rahaman et al. 2007; Şhin and Süral 2007; Pitoura and Samaras 2001; Jixiong et al. 2005; Li et al. 2004].

To address the scalability problem in a mobile computing system, Pitoura and Samaras proposed a hierarchical system with a tree topology [Pitoura and Samaras 2001]. This system relieves the scalability problem by locally updating the databases in the system. In a hierarchical database system, clustering the databases is a very important issue to reduce the update cost. But the optimal clustering can only be obtained exhaustively because the user moving patterns are dynamic in their nature.

Jixiong et al. developed a location database clustering algorithm and later called it *the set-cover algorithm* [Jixiong et al. 2005]. Their algorithm utilizes the "greedy" approximation set-cover algorithm for clustering with a bottom-up approach. However, once some of the databases in cells are grouped into a cluster at the bottommost level, it is difficult to guarantee that the movement information among the cells is used properly for clustering in the upper levels toward the root.

In this paper, we propose a top-down clustering algorithm for the location databases. In our clustering algorithm, we consider the number of visits to each cell by users, called *the visit count* of a cell, as well as the movement information between a pair of adjacent cells; that is, our algorithm takes into account both the node (cell) and edge information, while the set-cover algorithm utilizes only the edge information for clustering. We modified the proposed algorithm by incorporating the exhaustive method when there remain a few levels of the tree to be processed.

Although the proposed top-down algorithms enhance the update performance, these algorithms and the set-cover algorithm always construct full n-ary trees as their

hierarchical structures. But if we consider the capacity of a database in the system, these algorithms suffer from higher update costs. By the *capacity* of a database we mean the maximum number of mobile device users in the cells that can be handled by the database. To improve the update cost, we present a capacity constraint top-down clustering algorithm by modifying the proposed top-down clustering algorithm. The hierarchical structure of a tree constructed by the capacity constraint top-down clustering algorithm is not necessarily a full n-ary tree, since, during the top-down construction, a node may not be split if the database of the node can handle all the mobile device users in the cells managed by the database.

The experimental results show that the proposed, top-down, modified top-down, and capacity constraint top-down clustering algorithms reduce the update cost by 17.0%, 18.0%, 24.1%, the update time by about 43.0%, 39.0%, 42.3% over the set-cover algorithm, respectively. The results also show that the capacity constraint algorithm reduces the average number of databases used for the system by 23.9% over other algorithms.

The rest of this paper is organized as follows that Section 2 provides the backgrounds on the location database management in a cellular network; the proposed location database clustering algorithms are described in Section 3; in Section 4, the experimental results are given as well as the performance analysis; and finally in Section 5, conclusions are made.

2. BACKGROUNDS

2.1 Hierarchical Location Database Management

A hierarchical location database system has a tree topology as shown in Figure 2. The tree in the figure is a ternary tree in which nodes are databases. A leaf node is associated with a specific cell in the network. Assume that the tree is constructed with the network in Figure 3. Each vertex in the network represents a cell, an edge indicates the link between a pair of adjacent cells, and a weight on an edge denotes an amount of movements of the users between the cells.

In maintaining the databases in the system while users move around the network, we should reduce the update cost as much as possible by properly clustering the databases. In Figure 3, nine cells are grouped into three clusters and Figure 2 illustrates the hierarchical system based on the clustering in Figure 3.

We now define how to compute the update cost of the location databases in the hierarchical system for given users moving patterns as shown in Figure 3. We follow the same definitions as those in [Jixiong et al. 2005; Li and Lam et al. 2004]. Let each of n databases belong to a cell i, where i=1, 2, ..., n. Then the update cost of the movements made by the users between cells i and j can be defined as $UpdateCost(i,j)=(2\times h(i,j)+1)\times Moves(i,j)$, where $h(i,j)$ is the height of the lowest common ancestor between i and j in the tree, and $Moves(i,j)$ is the number of moves between cells i and j made by the users. Note that the length of the path from i to j via the lowest common ancestor is $2*h(i,j)$. We add 1 to the length, since the database in each node along the path from i to j via the lowest common ancestor should be updated.

For example, $UpdateCost(3,6)=(2\times 2+1)\times 4=20$ and $UpdateCost(4,1)=(2\times 1+1)\times 2=6$.

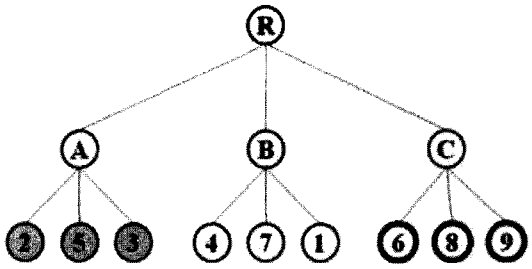


Figure 2. A Ternary Tree in a Hierarchical System.

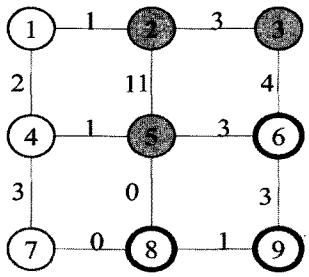


Figure 3. A Network Represented as a Graph.

2.2 The Set-Cover Algorithm

Jixiong et al. presented a location database clustering algorithm which is based on the greedy set-cover approximation algorithm [Jixiong et al. 2005; Chvátal 1979]. The algorithm constructs a tree using a bottom-up approach and is described below.

Algorithm I. The Set-cover Algorithm.

Input: Graph $G=(V, E)$, where $V=\{1, 2, 3, ..., n\}$ and E is the set of weighted edges connecting the cells
Output: $C_1, C_2, \ldots, C_{n/k}$
for i=1 to n/k do
1. Insert the cells incident to the edge with the largest weight in G into cluster C_i
2. Select the cell x that has the largest sum of the weights of the edges between x and each of the cells in C_i and insert x into C_i
3. **while** ($|C_i| < k$)
   select the cells as in Line 2 insert them into C_i
return $C_1, C_2, \ldots, C_{n/k}$

The above algorithm clusters the cells for only one level. We assume that the number n of cells in the input network is k^c such that the hierarchical system is implemented with a full k-ary tree and c is a positive integer. G is supposed to have a weight on each edge connecting a pair of adjacent cells. The weight indicates the number of moves between the cells made by the users.

In the above algorithm, after a cell is inserted into a cluster, the cell is removed

from G and hence all the edges incident to the cell are also removed from G. The ties in Lines 1 and 2 are broken arbitrarily. In each iteration of the for-loop, a cluster is constructed in such a way that the two endpoint cells incident to the edge with the largest weight are included into the cluster, and thereupon each cell with the largest connectivity with the cluster is inserted into the cluster 'greedily' until there are k cells in the cluster.

Let's trace the set-cover algorithm with the network in Figure 3. The tree is ternary, so k=3, V={1, 2, 3, 4, 5, 6, 7, 8, 9}. Since edge (2,5) has the largest weight, its endpoints 2 and 5 are inserted into C_1, i.e., C_1={2,5}, and then cells 3 and 6 have the highest connectivity with C_1; the sum of the weights of the edges (3,2) and (3,5) is 3 and similarly the sum is 3 for cell 6. The ties are broken arbitrarily, and so, say 3 is chosen here. Hence C_1={2, 5, 3}.

Now the algorithm proceeds with the remaining subgraph induced by V={1, 4, 6, 7, 8, 9} to get C_2. The result of the second iteration of the for-loop is C_2={4,7,1}. Finally C_3 has the rest of the cells in G. So, finally the algorithm returns the clusters, C_1={2, 5, 3}, C_2={4, 7, 1}, and C_3={6, 8, 9} as in the bottom-most level of the tree in Figure 2.

For the next level clustering, the set-cover algorithm creates a new network by treating each cluster as a node and by connecting an edge between two nodes if a cell in one node is adjacent to a cell in the other node in the previous level. The weight of an edge in the new network is the summation of the weights of the edges, each of which is connecting the two cells in different nodes. And then the set-cover algorithm is applied to this new network to cluster the nodes, and keeps doing so until the nodes are grouped into one cluster.

Although the set-cover algorithm is very simple, it constructs 'very good' clusters and is almost unbeatable by any bottom-up approaches, for example, the ones based on genetic algorithms and simulated annealing techniques. Nevertheless the set-cover algorithm is not an optimal algorithm; hence there is still some room for improvement. Notice that in a bottom-up clustering algorithm, when some clusters are not 'good' in a level, the clustered results may be propagated to the upper levels. To alleviate such problems, we suggest a top-down clustering with the visit count information along with the edge connectivity.

3. THE TOP-DOWN CLUSTERING ALGORITHMS

We now describe the proposed clustering algorithms in detail. First we describe a top-down clustering algorithm and then present a modified version to reduce the update cost further. Finally the capacity constraint top-down clustering algorithm is presented.

3.1 The Top-down Clustering Algorithm

The proposed algorithm first calculates the visit count of each cell (node) in the network. It then finds the node that has the largest visit count. We call the node *the seed node.* It then inserts it into a cluster which is initially empty. Now starting from the seed node, the algorithm selects the node with the largest movements to the cell(s) in the cluster and inserts it to the cluster. It keeps doing this until the cluster has the proper number of nodes. And then it checks the size of a cluster. If the cluster has more than k nodes, then the cluster should be split further by calling the algorithm

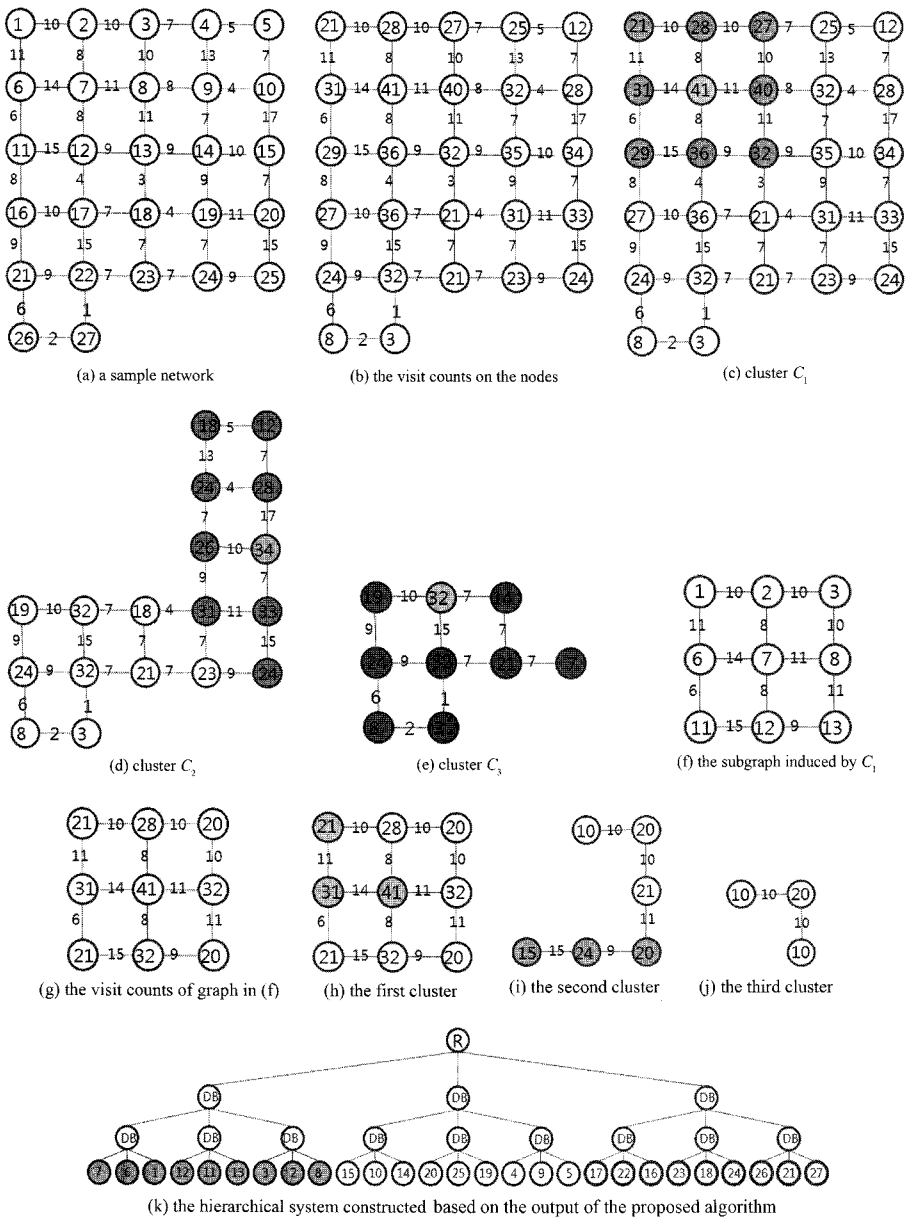
recursively. We again assume that the number n of cells in the input network is k^c such that the hierarchical system is implemented with a full k-ary tree and c is a positive integer. Note that for a recursive call, the graph H is a subgraph of the input graph G induced by V.

Algorithm II. The Top-down Algorithm.

Top-down Approach(H, N)
//**Input**: Graph $H=(V,E)$, where $V=\{1, 2, 3, \ldots, N\}$ and E is the set of weighted edges connecting the cells
//**Output**: $C_1, C_2, \ldots, C_k$
for i=1 to k do
1. Calculate the visit count of each cell in H
2. Select the cell x with the largest visit count
3. Insert x into C_i and remove x from H
4. **while** (the number of cells in C_i N/k)
5. Select the cell y that has the largest sum of the weights of the edges between the cell and each of the cells in C_i
6. Insert y into C_i and remove y from H
for j=1 to k do // splitting a cluster if its size > k
7. **if** ($|C_j| > k$) then C_j=***Top-down Approach***$(C_j, N/k)$
return $C_1, C_2, \ldots, C_k$;

Initially, we call the algorithm with *Top-down Approach*(G,n), where G is the input network and n is the number of cells in the network. The above algorithm has two for-loops. The algorithm constructs k clusters with the first for-loop and splits the clusters by recursive calls in the second for-loop. The recursive calls are made until each cluster has k cells.

We now trace the proposed algorithm with an input network in Figure 4(a). We assume that k=3, n=3^3=27. In Line 1 the visit count of each cell is obtained as in Figure 4(b). In this example, 7 becomes the seed node for cluster C_1 as shown in Figure 4(c). In Lines 4~6, we find the cell x that has the largest sum of the weights of the edges between x and the cells in C_1 one by one until there are 27/3=9 cells in C_1. Figure 4(c) shows the result, C_1={7, 6, 1, 2, 8, 3, 13, 12, 11}, with the order of the 'greedy' selections by the algorithm. During the selection, when the tie occurs, the algorithm selects the one with the smallest index. Similarly, C_2={15, 10, 14, 9, 4, 5, 19, 20, 25} and C_3={17, 22, 16, 21, 18, 23, 24, 26, 27} are obtained after the while-loop is terminated and are shown in Figure 4(d) and (e), respectively. In Line 7, since the size of each cluster is greater than k=3, we call it *Top-down Approach*(C_1, 27/3) recursively. Figure 4(f) shows the subgraph of the input graph G induced by C_1. Figure 4(g) shows the visit counts for the network in Figure 4(f). We now get C'_1={7, 6, 1}, C'_2={12, 11, 13}, and C'_3={3, 2, 8} in turn, as shown in Figure 4(h)~(j). Upon returning from the call *Top-down Approach*(C_1, 27/3), we get C_1 = {7 6 1; 12 11 13; 3 2 8}. We can get C_2 = {15 10 14; 20 25 19; 4 9 5} and C_3={17 22 16; 23 18 24; 26 21 27} in the same manner. So the final output is {7, 6, 1}, {12, 11, 13}, {3, 2, 8}, {15, 10, 14}, {20, 25, 19}, {4, 9, 5}, {17, 22, 16}, {23, 18, 24}, and {26, 21, 27}. The hierarchical system constructed based on the output of the proposed algorithm is shown in Figure 4(k).



(a) a sample network (b) the visit counts on the nodes (c) cluster C_1

(d) cluster C_2 (e) cluster C_3 (f) the subgraph induced by C_1

(g) the visit counts of graph in (f) (h) the first cluster (i) the second cluster (j) the third cluster

(k) the hierarchical system constructed based on the output of the proposed algorithm

Figure 4. A Trace of the Proposed Algorithm.

Note that the update cost of the databases on the system in Figure 4(k) is 973, while the cost on the system in Figure 5 constructed with the set-cover algorithm on the same input is 990.

3.2 The Modified Clustering Algorithm

In this section we propose a modified top-down algorithm. In the top-down approach,

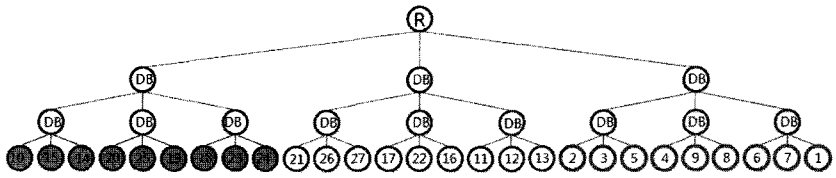


Figure 5. The Hierarchical System Constructed with the Set-cover Algorithm for the Sample Network in Figure 4(a).

the number of nodes that participate into clustering becomes smaller since the clustering is proceeded from top to bottom. In the modified algorithm, we stop calling *Top-down Approach* recursively at a particular level l, and find the optimal clustering for the networks at level l with the exhausted method. This technique is exactly the same idea used when sorting a huge file, quick sort is used recursively until the file size gets smaller; from then on, an elementary sort such as the insertion sort is used.

It was found out that when the number of nodes is not greater than 9, it is reasonable to stop the recursive calls for node splitting. If it is greater than 9, it takes longer time to get the optimal clustering. We obtained this value after various experiments such as 9, 27, 81, and 729.

3.3 The Capacity Constraint Top-Down Clustering Algorithm

The set-cover algorithm and the above proposed algorithms do not care about the capacity of a database. But in reality a database cannot handle unlimited number of mobile device users in the cells; hence, we should not ignore such a factor for a location database system. An algorithm that does not take into account the capacity factor constructs a full n-ary tree in which some nodes should have not been split. Such unnecessary node splits result in using more databases for the system.

By modifying the top-down clustering algorithm we propose a capacity constraint top-down clustering algorithm. During a recursive call for node splitting, we check if the capacity of a node (cluster) is enough for handling the users, that is, it is not greater than the summation of movements in the cells of the cluster. If so, we do not split the node. Otherwise, we spilt it recursively. The following code replaces Line 7 of the top-down clustering algorithm for the capacity constraint clustering algorithm.

```
7. if ( |Cj|'s movement < the capacity of a database)
     then return; // do not split any further
   else if (|Cj| > k)
     then Cj=Top-down Approach(Cj, N/k)
```

Since a tree generated with this algorithm is not always a full n-ary tree, the update cost is expected to be smaller than those of other algorithms and the number of databases deployed should not be greater than those of other algorithms. In the next section we analyze the performances of the proposed clustering algorithms and compared them with that of the set-cover algorithm.

4. EXPERIMENTAL RESULTS

We tested the performances of the proposed clustering algorithms and the set-cover algorithm under various experimental environments. The experiments were performed on a PC with Core2Quad Q6600 2.4 Ghz processor, 8 GBytes RAM, and Vista 64 bits. The experimental parameters are given in Table I. We assumed that a tree for the hierarchical system is a full ternary tree. The numbers of cells in the networks are assumed to be 3^4, 3^6, and 3^8. The average number of users in a cell of the network is assumed to be 15 by analyzing the experiments done in [Şhin and Süral 2007]. Hence, for example, when the network size is 3^8, there are $3^8 \times 15 = 98{,}415$ users in the network on the average.

For the users' movements in the networks, we define *the number of boundary crossings* as the number of movements made by all the users between two cells in the network; that is, the average number of boundary crossings is the sum of the weights of all the edges in the network divided by the number of edges. We made five input networks for each network size and each network has ten variants by changing the average number of boundary crossings from 2 to 20 with an increment of 5. We tested four different capacities for a database; they are 500, 1000, 1500, and 2000. Note that we assumed that all the databases in the system have the same capacity for the experiments.

The following three figures show the update costs, update times, and numbers of databases used for the cell size 3^4, varying the number of boundary crossings from 2 to 20 with an increment of 2.

When there are 3^4 cells, the proposed algorithms have reduced the update costs with respect to the set-cover algorithm, while the modified top-down was better than the top-down algorithm and the capacity constraint algorithms showed the best performance. We can acknowledge that the capacity constraint algorithm constructs a tree with a smaller height as the capacity increases. The update cost of the capacity constraint algorithm becomes almost the same as that of the top-down algorithm as the number of boundary crossings increases, because if the movements are increased we need a larger number of databases.

Regarding the update time, the top-down algorithm exhibits the best performance because the set-cover algorithm always calculates the movements of between a pair of all the cells during each recursive call while the top-down algorithm calculates the movements between a pair of the cells involved only with the current recursive call. Observe that the modified top-down algorithm shows longer update times, since it

Table I. Experiment Parameters.

Parameters	Values
the number of children of each node in a tree	3
the number of cells	3^4, 3^6, 3^8
the average number of users in a cell	15
the average number of boundary crossings	2-20
the capacity of a database	500, 1,000, 1,500, 2,000

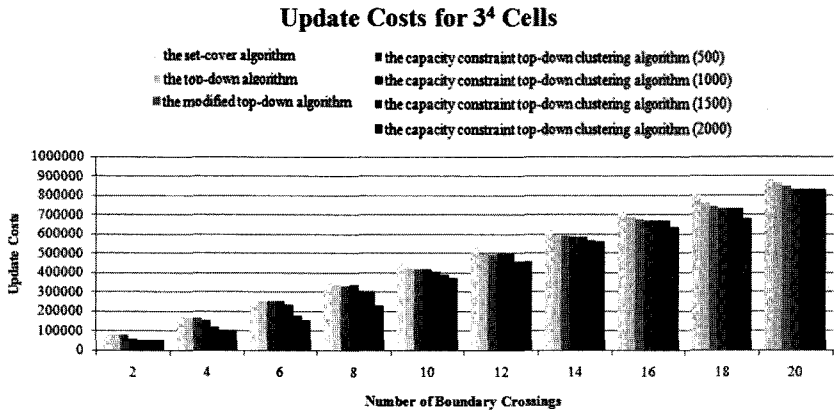


Figure 6. Update Costs for 3^4 Cells.

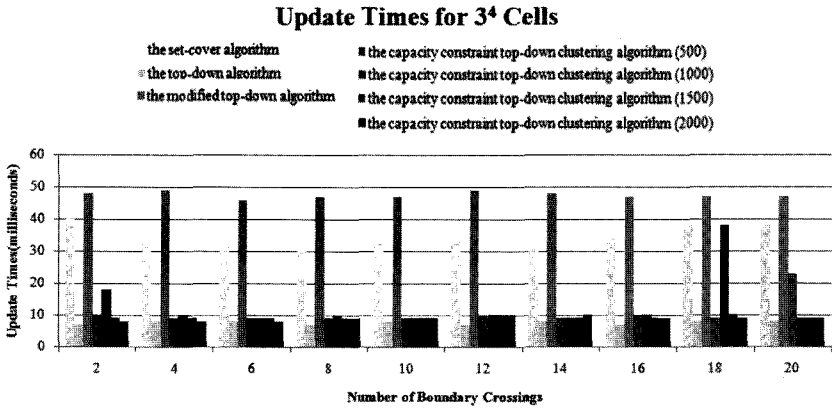


Figure 7. Update Times for 3^4 Cells.

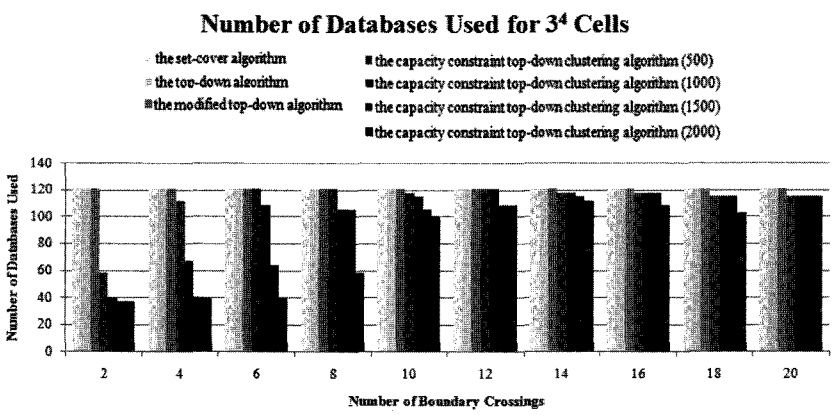


Figure 8. Numbers of Databases Used for 3^4 Cells.

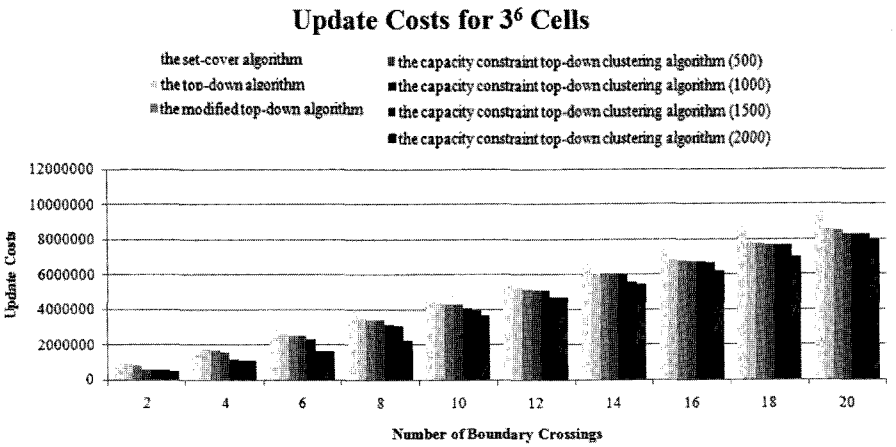


Figure 9. Update Costs for 3^6 Cells.

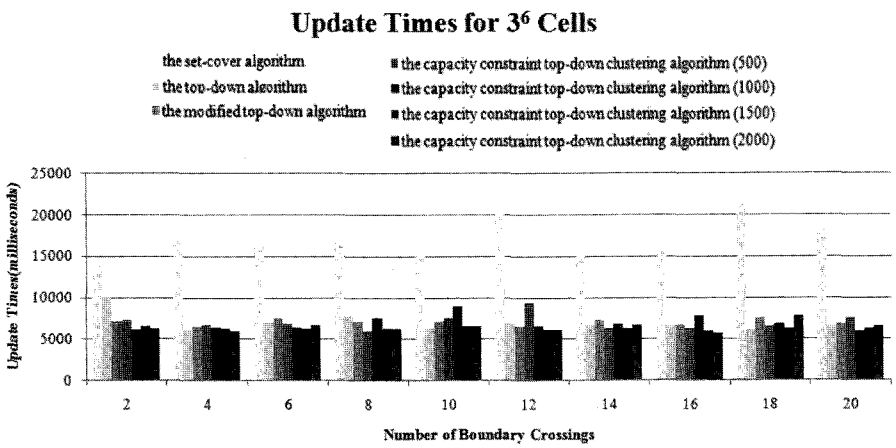


Figure 10. Update Times for 3^6 Cells.

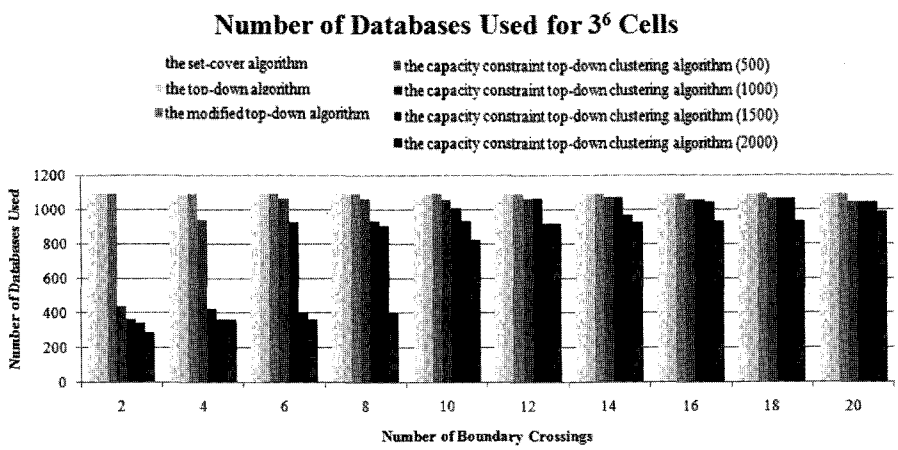


Figure 11. Numbers of Databases Used for 3^6 Cells.

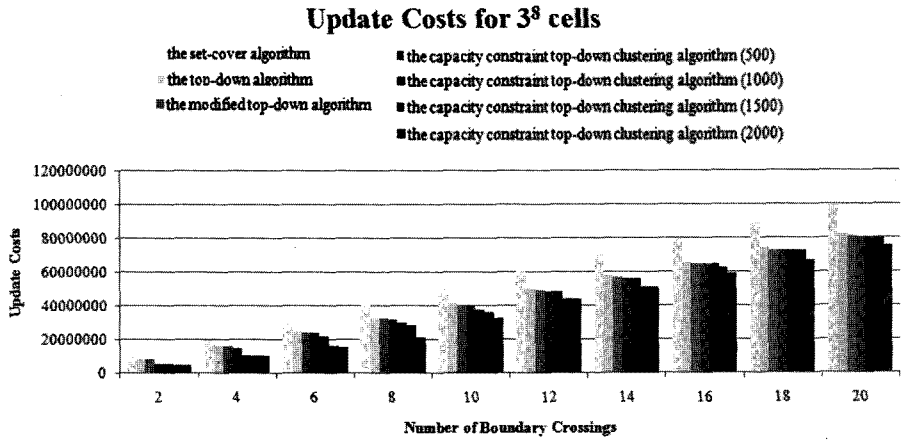


Figure 12. Update Costs for 3^8 Cells.

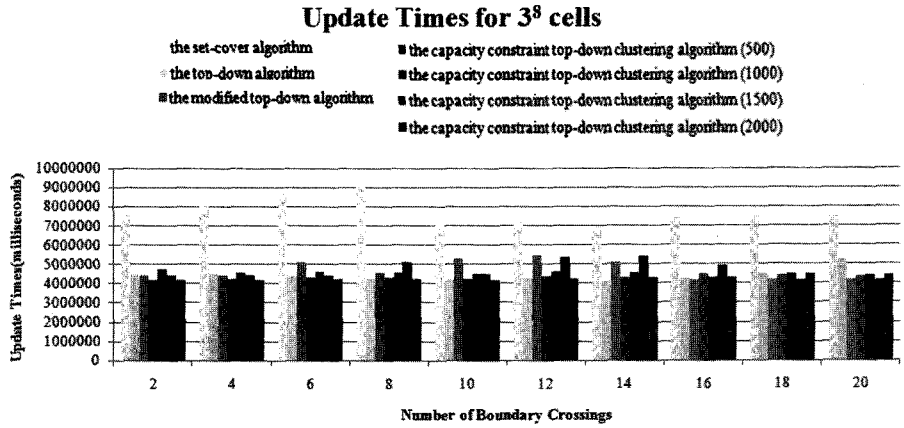


Figure 13. Update Times for 3^8 Cells.

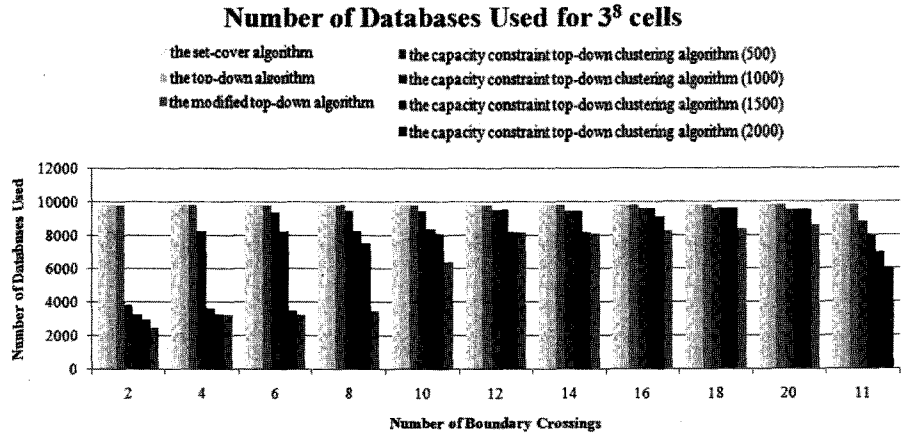


Figure 14. Numbers of Databases Used for 3^8 Cells.

Table II. Average Experiment Results.

parameters			update cost		update time		no. of databases used	
			x 1,000	%	seconds	%		%
set-cover algorithm			20,255	-	2,573	-	3684.7	-
top-down algorithm			16,803	17.0	1,468	43.0		
modified top-down algorithm			16,607	18.0	1,569	39.0		
capacity constraint top-down clustering algorithm	capacity	500	16,300	19.5	1,436	44.2	3314.8	10.0
		1,000	15,876	21.6	1,512	41.3	2991.0	18.8
		1,500	15,158	25.1	1,565	39.2	2636.6	28.4
		2,000	14,202	29.9	1,430	44.4	2276.5	38.2

tries to find the optimal clustering when the cell size is 9. For the number of databases required for the system, it is obvious that the set-cover, top-down, and modified top-down algorithms require the same number of databases, since they create full n-ary trees. But for the capacity constraint algorithm, the number of databases increases as the number of boundary crossings increases. The following six figures show the update costs, update times, and numbers of databases used for the cell sizes 3^6 and 3^8, varying the number of boundary crossings from 2 to 20 with an increment of 2.

When the numbers of cells are 3^6 and 3^8, we obtained similar results to those for 3^4. For the update cost, our proposed algorithms perform well as the cell size becomes larger. Note that the capacity constraint top-down clustering algorithm with 2,000 as the database capacity has the shortest update time among all the results. Such a result is possible since the tree constructed with the algorithm has the smallest number of databases and has a lower height.

The following table compares the performances of the algorithms for the average update cost, update time, and number of databases used. The 'percent' column of each parameter indicates the improvement of an algorithm in a percentage with respect to the set-cover algorithm.

The proposed, top-down, modified top-down, and capacity constraint top-down clustering algorithms reduce the update cost by 17.0%, 18.0%, 24.1%, the update time by about 43.0%, 39.0%, 42.3% over the set-cover algorithm, respectively. The results also show that the capacity constraint algorithm reduces the average number of databases used for the system by 23.9% over other algorithms.

5. CONCLUSIONS

In a wireless environment, the location database management needs to be done as efficiently as possible. When the size of a network increases, updating the location databases may degrade the performance of the system. In this paper, we have proposed a clustering algorithm to reduce both the update cost of the location databases and the update time in the hierarchical system. The proposed algorithm exploits the visit counts of the cells in finding the seeds of clusters. Afterwards, each

cluster gathers the cells greedily using the movement information with a top-down approach. We also proposed a modified version of the top-down clustering algorithm that incorporates the exhausted method for finding the optimal clustering at a lower level of the tree.

The set-cover algorithms and the two proposed algorithms do not consider the capacity of a database. But the capacity factor allows us to construct a tree for a hierarchical system with less number of databases as well as a possibly lower height, as we have a larger capacity of a database. The capacity constraint top-down clustering algorithm we proposed is able to suggest a proper size of the capacity in practice.

We tested and compared the proposed algorithms against the set-cover algorithm with various inputs. The experimental results showed that the top-down clustering algorithm performed quite well especially when the network size is large and improved the update cost by 17.0% over the set-cover algorithm. It can be seen that the seed in building up a cluster played a pivotal role and the top-down way of splitting preserved the cohesiveness of the cells in a cluster. The results also show that the modified top-down algorithm reduced the update cost by 18.0% over the set-cover algorithm. The partial optimal values could reduce the update cost further. The results also show that the capacity constraint top-down clustering algorithm reduced the update cost by 24.1% over the set-cover algorithm. It uses a smaller number of databases than other algorithms. The proposed, top-down, modified top-down, and capacity constraint top-down clustering algorithms reduce the update time by about 43.0%, 39.0%, 42.3% over the set-cover algorithm, respectively. The results also show that the capacity constraint algorithm reduces the average number of databases used for the system by 23.9% over other algorithms. All the results show that the scalability problem of the location database management in the wireless network had been resolved to some extent.

ACKNOWLEDGMENTS

This work was supported by the Korea Science and Engineering Foundation (KOSEF) for the research (2010-0015846).

REFERENCES

CHVÁTAL, V. 1979. A greedy-heuristic for the set covering problem. *Math. Oper. Res. 4*, 233–235.

FRATTASI, S., FATHI, H., FITZEK, F., PRASAD, R., AND KATZ, M. 2006. Defining 4G technology from the users perspective. *IEEE Network.*

JIXIONG, C., GUOHUI, L., HUAJIE, X., XIA, C., AND BING, Y. 2005. Location database clustering to achieve location management time cost reduction in a mobile computing system. *Wireless Communications, Networking and Mobile Computing 2*, 23-26, 1328–1332.

LI, G., LAM, K., KUO, T., AND WU, S. 2004. Location management in cellular mobile computing systems with dynamic hierarchical location databases. *Journal of Systems and Software 69*, 1–2, 159–171.

NAM, C., KIM, S., AND LEE, H. 2008. The role of wibro: filling the gaps in mobile broadband technologies. *Vehicular Technology Magazine.*

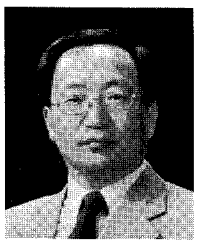
PITOURA, E. AND SAMARAS, G. 2001. Locating objects in mobile computing. *IEEE Transactions on Knowledge and Data Engineering 13*, 571–592.

RAHAMAN, A., ABAWAJY, J., AND HOBBS, M. 2007. Taxonomy and survey of location management systems. *IEEE International Workshop on Component-Based Software Engineering* 369–374.

ŞHIN G. AND SÜRAL, H. 2007. A review of hierarchical facility location models. *Elsevier Science Ltd.*

ZHENG, K., HUANG, L., LI, G., CAO, H., WANG, W., AND Dohler, M. 2008. Beyond 3G evolution. *Vehicular Technology Magazine.*

Kwang-Jo Lee received the B.S. in Computer Engineering from Sejong University Seoul, Korea, in 2007 and M.S. degrees in Computer Science from Yonsei University. He is currently a Ph.D. Candidate in Computer Science at Yonsei University. His research interests include Mobile Computing, Mobile Network, and 3D Graphics.



Sung-Bong Yang received Ph.D. degree in Computer Science from the University of Oklahoma in 1992. He has been a faculty member at the Department of Computer Science, Yonsei University, Seoul, Korea, since 1994. His research interests include Mobile Systems, Peer-to-Peer Computing, and 3D Graphics.