Dependent Data Broadcasting for Unordered Queries in a Multiple Channel Mobile Environment

Jiun-Long Huang and Ming-Syan Chen*
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan, ROC
E-mail: jlhuang@arbor.ee.ntu.edu.tw, mschen@cc.ee.ntu.edu.tw*

Abstract

Data broadcast is a promising technique to improve the bandwidth utilization and conserve the power consumption in a mobile computing environment. In many applications, the data items broadcast are dependent upon one another. However, most prior studies on broadcasting dependent data are restricted to a single broadcast channel environment, and as a consequence, the results are of limited applicability to the upcoming mobile environments. In view of this, we relax this restriction and explore in this paper the problem of broadcasting dependent data in multiple broadcast channels. By analyzing the model of dependent data broadcasting, we derive several theoretical properties for the average access time in a multiple channel environment. In light of the theoretical results, we develop a genetic algorithm to generate broadcast programs. Our experimental results show that the theoretical results derived are able to guide the search of the genetic algorithm very effectively, thus leading to broadcast programs of very high quality.

Index Terms: Data broadcast, dependent data, unordered query, mobile information system, mobile computing

*The corresponding author of this paper.

1 Introduction

In a broadcast-based information system [1][5][17], a server *periodically* broadcasts data to mobile users according to a pre-generated broadcast program by a *single* broadcast channel. To retrieve data items of interest, the mobile users need to wait for the appearance of the data items on the broadcast channel instead of explicitly sending data requests to the information system. Due to the above characteristics, data broadcast becomes a promising technique in a mobile computing environment since it has the following advantages:

- *Power conservation*: As mentioned in [16], the power needed for transmission from the mobile client to the mobile support station is proportional to the fourth power of the distance between them. Hence, the power is conserved in broadcast-based systems since mobile clients need not explicitly send data requests to information servers.

- *High scalability*: The high scalability is achieved since the system performance is independent of the number of clients.

- *High bandwidth utilization*: Data items of high interest can be received by multiple mobile clients by one transmission on the broadcast channel.

One objective of designing proper data allocation in the broadcast disks is to reduce the average access time of data items. Several related research issues have attracted a considerable amount of attention, including on-demand broadcast [2][3], data indexing [7][13][17], access frequencies estimation [11][30], and client cache management [25][28]. Note that there do exist situations where multiple low-bandwidth physical channels cannot be combined into a single high-bandwidth physical channel [23][26]. Scenarios that the system can only allocate multiple channels with non-contiguous frequency

ranges have been pointed out in [23][26], and these channels with non-contiguous frequency ranges cannot be merged into a single channel. In addition, several network standards, such as FDMA-based systems, divide the network bandwidth into several physical channels where individual mobile clients are designed to listen to at most one physical channel at a time. As a result, in such network environments, multiple physical channels (even with contiguous frequencies) cannot be merged into one single channel in nature.

Hence, a significant amount of research effort has been elaborated on developing the index mechanisms [20][24], data allocation schemes [22][23] and dynamic data and channel allocation [14][18] in *multiple* broadcast channels. In addition, the bandwidth allocation for multi-cell environments with frequency reuse and inference considered was studied in [29].

Most works mentioned above were under the premise that each user requests only one data item at a time and the requests for all data items are independent. That is, for an arbitrary user, the access probability that the user requests a data item in the i-th request is *predetermined* and is *independent* of what have been requested in the first, second, $\cdots$, $(i-1)$-th requests. However, in many real applications, some data items are *semantically related* and there exists dependency among the requests of these data items. Broadcast program generation algorithms assuming independent requests might not be able to effectively optimize the performance of the broadcast programs. This phenomenon attracts a series of work on solving the problem of dependent data broadcast. A query corresponds to a set of semantically related data items which are likely to be requested successively by a user, and we use a set of queries (named a query profile) to model such dependency of all data items. Hence, if several data items are *dependent* upon one another (i.e., within the same query), they are likely to be successively requested by users. According to the constraint of the retrieval sequence of the dependent data items

within the same query, queries of dependent data can be categorized into the following two types :

Ordered queries: In an ordered query, the required data items should be retrieved in a predetermined order. Consider a Web page with some images as an example. Once the user requests this Web page by a browser, the browser will request these images automatically in a predetermined order after receiving this Web page.

Unordered queries: Similar as an ordered query, an unordered query could be one issued by a mobile user for requesting multiple data items simultaneously. However, unlike in an ordered query, these requested data items may be retrieved in any order. Consider a broadcast system to disseminate the stock information. A mobile user may submit a query like "Show me the stock information of all the LCD companies." As a result, data items in these LCD companies are queried together and displayed without being confined to a specific order.

Prior research works of dependent data broadcast can be categorized by the following two properties: (1) the number of broadcast channels considered (single [6] or multiple channels [15]) and (2) the constraint of the retrieval sequence of the data items (ordered [6][15] or unordered [8] queries) in each query. It is noted that two heuristic algorithms are proposed in [6] to generate broadcast programs in a single channel for ordered queries. In [6], the queries are assumed to be *acyclic*, meaning that if a data item D_i is required to be accessed before D_j in one query, D_j will not be accessed before D_i in all other queries. The authors in [8] proposed a greedy algorithm to generate broadcast programs for unordered queries in one broadcast channel. A randomized algorithm is proposed in [4] to consider the dependency of *two* data items in single broadcast channel. However, there are few works for dependent data broadcasting on multiple channel environments. In [15], two heuristic algorithms were designed to generate broadcast programs in multiple channels for ordered queries. Similarly to [6],

the work in [15] also assumes the queries to be acyclic. In addition, the problem of index and data allocation [10][17][20] can be treated as a special case of broadcasting dependent data with ordered queries. Note that for each ordered query, the required data items of this query should be retrieved according to a predetermined order, and there will be exactly one retrieval order. However, the number of retrieval orders of an unordered query Q_i is $|Q_i|!$ where $|Q_i|$ is the number of required data items of Q_i. This feature makes the broadcasting dependent data for unordered queries more difficult than that for ordered queries.

Consequently, we address in this paper the problem of broadcast program generation for unordered queries with dependent data in multiple broadcast channels. Note that the problem of broadcasting dependent data in a multiple channel environment is intrinsically difficult in that the factor of data dependency and the efficient use of multiple channels, though being dealt with separately before, are in fact entangled, thus making it more complicated to provide an effective solution to this problem. Specifically, several special cases [8][10][20] of the problem of broadcasting dependent data are shown to be NP-hard. In view of this, we shall employ the Genetic Algorithm (abbreviated as GA) in this paper to address the problem of broadcasting dependent data in a multiple channel environment. GA is a widely-used approach in the literature of soft computing and evolutionary computation to solve optimization problems. Basically, GAs are iterative procedures that search the problem solutions by an evolutionary process based on natural selection. GA maintains a population of individual candidate solutions to specific problems. An individual candidate solution can be represented as a list called a *chromosome*. In GA, a *fitness function* has to be designed to evaluate the fitness of each chromosome. The probability that each chromosome will be selected is in proportion to its fitness. The design of the fitness function is key to the effectiveness of the GA algorithms.

Explicitly, in this paper we first model the problem of broadcast program generation for unordered queries in multiple channels. By analyzing the model of dependent data broadcasting, we derive several theoretical properties for the average access time in a multiple channel environment. In light of the theoretical results, we then formulate the fitness functions for the GA to generate broadcast programs. Sensitivity analysis on several parameters is conducted. Our experimental results show that with the analytical results derived, the fitness functions designed are able to guide the search of GA very effectively, thus leading to broadcast programs of very high quality. To the best of our knowledge, there is no prior work on the broadcast program generation for unordered queries in multiple channels. This fact distinguishes this paper from others.

The rest of this paper is organized as follows. Section 2 gives the preliminaries of this study. Analytical models of the problem of dependent data broadcast with unordered queries are given in Section 3. In light of the analysis in Section 3, we devised a GA-based algorithm in Section 4. Performance evaluation on various parameters is conducted in Section 5. Finally, Section 6 concludes this paper.

2 Preliminaries

2.1 Overview of Genetic Algorithms

The flowchart of GAs is presented in Figure 1. Before designing a GA, one must decide a chromosome representation (e.g., tree, list...) policy to transform a solution into a chromosome. In addition, a fitness function is also designed to evaluate how good a chromosome (i.e., a solution) is. Several chromosomes are generated to form initial population. Initial population can be generated by random generation or heuristics. In fitness evaluation phase, the fitness values of all chromosomes are evaluated according

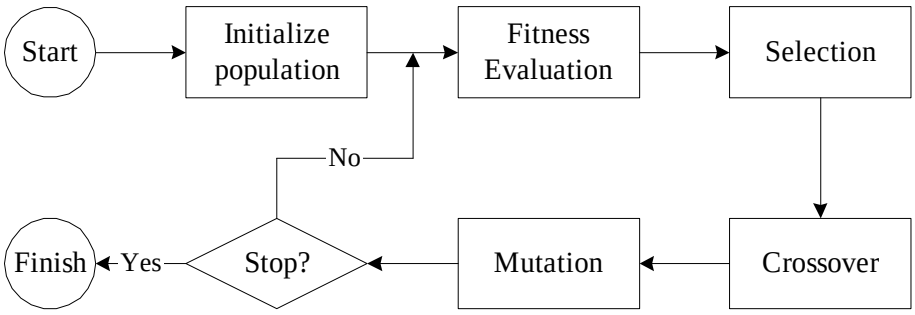

Figure 1: The flowchart of a genetic algorithm

to the fitness function taken. The chromosome with the maximal fitness is then selected as the best chromosome.

During the selection phase, chromosomes in the current population are selected and recombined to produce offspring which will comprise the next generation of population. Chromosomes are selected randomly from the current population using a roulette wheel with slots sized according to fitness. Consider a generation containing three chromosomes and suppose that the fitness of chromosomes 1, 2, and 3 are 4, 2, and 2, respectively. The percentages of chromosomes 1, 2, and 3 being selected will be 50% ($\frac{4}{4+2+2}$), 25% ($\frac{2}{4+2+2}$), and 25% ($\frac{2}{4+2+2}$), respectively. Thus, chromosomes with higher fitness have a higher likelihood to be reproduced.

After selection, crossover is applied in the selected chromosomes with probability P_C. This operator takes two randomly chosen parent chromosomes as input and combines them to generate two children. The crossover operator provides the exploration capacity by exchanging the information from two parents. Crossover may lead to fall into a local optimum of the fitness function because the generated children tend to resemble their parents. In order to reduce this phenomenon, mutation operates with a probability P_M and creates new chromosomes by modifying one or more of the gene values of an existing chromosome. Mutation provides a random search in the solution space and reduces the probability of falling into a local optimum of the fitness function.

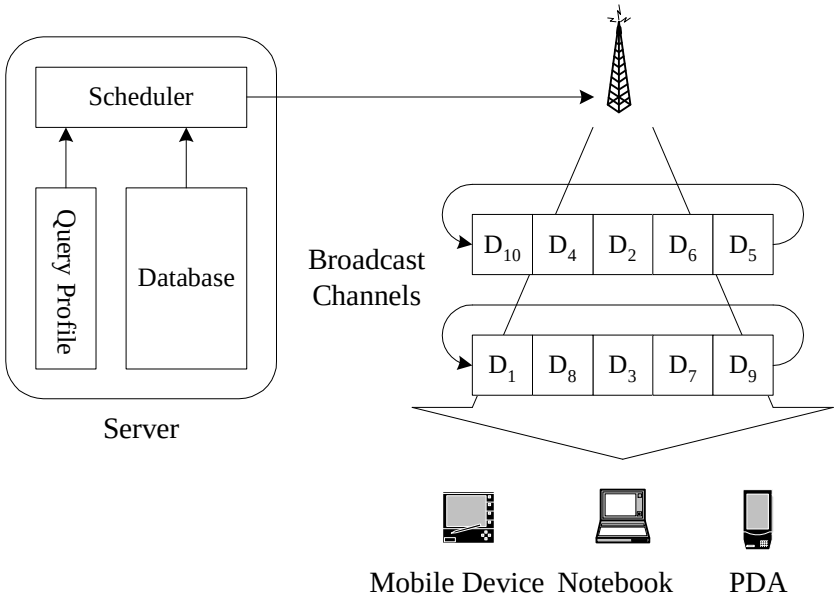

Figure 2: The architecture of a data broadcast system

Finally, a termination criterion is checked to decide whether to terminate the GA. If yes, the best chromosome up to now is then returned as the result of the GA. If not, the GA will enter the fitness evaluation phase to start another iteration. An iteration is called a generation in the GA literature. Interested readers are referred to [9] for the details of GAs.

2.2 System Architecture

Figure 2 shows an example system architecture of a data broadcast system which broadcasts data items periodically according to a broadcast program. We assume that each data item is read-only [19]. In the beginning, since a mobile user will not know the broadcast program of the system, the mobile device should listen on a broadcast channel to wait for the appearance of the broadcast program. The broadcast program contains some auxiliary information such as the data identifiers, attributes for each data item, the data index information and so on. Attributes for each data item are then used to process the query submitted by the user. The broadcast program is kept in the mobile device until it is expired. When

Query(Q_i)	$Pr(Q_i)$
$Q_1 = \{D_1, D_2, D_3, D_4\}$	40%
$Q_2 = \{D_1, D_3, D_4\}$	20%
$Q_3 = \{D_4, D_6, D_1\}$	20%
$Q_4 = \{D_1, D_2, D_5\}$	20%

Figure 3: An example query profile

D_1	D_4	D_2
D_6	D_3	D_5

Figure 4: An example broadcast program

the user submits a query to his or her mobile device, the mobile device will process this query with the aid of stored broadcast program to determine the required data items. The mobile device will then determine the optimal retrieve order to minimize the access time of the required data items and retrieve them from broadcast channels according to the determined retrieve order.

2.3 Problem Description

Suppose that the database D contains $|D|$ data items, $D_1, D_2, \cdots, D_{|D|}$. From the users' point of view, a query is an indivisible request of single or multiple data item(s), and is defined as follows.

Definition 1: An *unordered* query $Q_i = \{D_{q^i(1)}, D_{q^i(2)}, \cdots, D_{q^i(|Q_i|)}\}$ is a non-empty subset of all data items where $|Q_i|$ represents that number of required data items in Q_i and $D_{q^i(x)} \neq D_{q^i(y)}$ when $x \neq y$. Note that $1 \leq q^i(j) \leq |D|$ for all j, $1 \leq j \leq |Q_i|$, and $q^i(j) = k$ represents that the j-th accessed data item in Q_i is D_k.

The query profile is the aggregation of the access behavior of all users. Formally, we have the following definition.

Definition 2: A query profile Q consists of a set of $\langle Q_i, Pr(Q_i)\rangle$ pairs where $|Q|$ indicates the number of queries in Q. $Pr(Q_i)$ represents the probability that a query issued by users is Q_i. It is noted that $\sum_{i=1}^{|Q|} Pr(Q_i) = 1$.

The capture of the query profile is a challenging problem since the data items are dependent upon one another. If the same database can be accessed from Internet, we can assume that the query behavior

of Internet users and mobile users are similar in essence, and hence use the query profile of Internet users to model that of mobile users. Similar to [12], when an uplink channel is provided, the mobile device can store hot queries of its owner and send its query statistics to the server. When the mobile device is about to connect to the server, this statistics can be transferred by piggypacks. Clearly, dependent on applications, different methods to capture the query profile of a broadcast-based information system are conceivable.

Assume that the bandwidth of each channel is divided into slots of equal size s. A data item D_i will occupy $\left\lceil \frac{|D_i|}{s} \right\rceil$ slots where $|D_i|$ is the size of D_i. Let $r = \left\lceil \frac{|D_i|}{s} \right\rceil$ and D_i occupy slots D_i^1, D_i^2, $\cdots$, D_i^r. If a mobile device requests D_i, the system will retrieve $D_i^1, D_i^2, \cdots, D_i^r$ to compose D_i. Therefore, a query containing a multi-slot data items D_i should be expanded from $\{\cdots, D_i, \cdots\}$ into $\{\cdots, D_i^1, D_i^2, \cdots, D_i^r, \cdots\}$. Since the expanded queries are still unordered, we assume that the data items are all of equal size for ease of presentation.

Example 1: Consider a database D containing six data items. Figure 3 shows an example query profile containing four queries. The query Q_3 will read data items D_4, D_6 and D_1. Note that the read of data items need not follow this order. $Pr(Q_3) = 20\%$ shows that 20% of the queries submitted by users is the query Q_3.

Let n represent the number of channels and L be the length of the broadcast program. A broadcast program is defined below.

Definition 3: A broadcast program P is a *placement* of all data items in D into an n by L array where $L = \left\lceil \frac{|D|}{n} \right\rceil$.

Here, we assume that $|D| = L \times n$ without loss of generality. To facilitate the further discussion, we define a function $placement : \{1, 2, \cdots, |D|\} \rightarrow \{1, 2, \cdots, L\}$ to be an onto function which maps

each data item into its placement in the broadcast program. Figure 4 shows an example of broadcast program on two broadcast channels. In this example, the value of $placement(4)$ is equal to 2 which indicates that the broadcast order of the data item D_4 in the broadcast program is 2.

Two metrics, *access time* and *tuning time*, are introduced in [17] to evaluate the performance of broadcast programs. The access time is the time elapsed from the moment a client issues a query to the moment all the relevant data are read. The tuning time is the amount of time spent by the client listening on the broadcast channels, which is a measurement of the power consumption. In this paper, we take the access time as the measurement of the performance of broadcast programs. Hence, broadcast program generation for unordered queries in multiple channels can be formulated as follows.

Definition 4: Given the number of broadcast channels, the database D and a query profile Q, the problem of broadcast program generation for unordered queries in multiple channels is to determine a broadcast program P which minimizes the average access time of the query profile Q.

Denote the average access time of a query Q_i as $T_{Access}(Q_i)$ and the average access time of a query file Q as $T_{Access}(Q)$. The average access time of the query profile Q can be formulated as

$$T_{Access}(Q) = \sum_{i=1}^{|Q|} \Big[T_{Access}(Q_i) \times Pr(Q_i) \Big]. \tag{1}$$

3 Analytical Models

3.1 Decomposition of Average Access Time

The access time of an arbitrary query Q_i can be decomposed into two portions: *startup time* and *retrieval time*. When a mobile user submits a query Q_i, the mobile device should wait until the system starts to broadcast any required data item of Q_i. This time interval is called startup time. Retrieval time

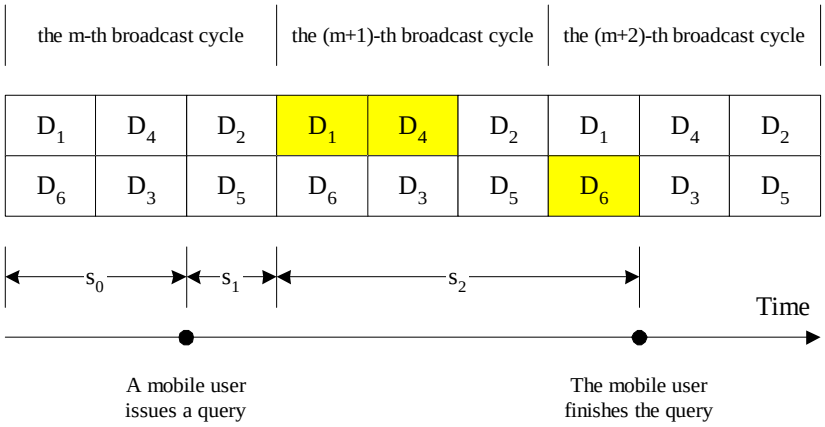

Figure 5: An example scenario of a query

is defined as the time intervals between the moment that the mobile device starts to read data items from broadcast channels and that the mobile device finishes Q_i.

Denote the size of each data item as s and the bandwidth of each broadcast channel as B. We have the following example.

Example 2: Consider the scenario shown in Figure 5. This example assumes that the mobile user submits Q_3 shown in Figure 3. After the user submits Q_3, the mobile device will manipulate the stored broadcast program and obtains the optimal request order to be $D_1 \rightarrow D_4 \rightarrow D_6$. It is noted that $D_6 \rightarrow D_4 \rightarrow D_1$ is also an optimal request order. And the choice of the mobile device will not affect the access time of Q_i. In this example, the startup time of Q_3 is $s_1 = 1 \times \frac{s}{B}$ and the retrieval time is $s_2 = 4 \times \frac{s}{B}$. Finally, the access time of Q_3 is equal to the summation of startup time and retrieval time (i.e., $5 \times \frac{s}{B}$).

Denote the average startup time and the average retrieval time of a query Q_i as $T_{Startup}(Q_i)$ and $T_{Retr.}(Q_i)$, respectively. By the above definitions, the average access time of a query is equal to the summation of the average startup time and the average retrieval time of the query. Hence, we have the following equation.

$$T_{Access}(Q_i) = T_{Startup}(Q_i) + T_{Retr.}(Q_i) \tag{2}$$

3.2 Derivation of Average Access Time

By Equation (1), the average access time of a query profile is a weighted summation of the average access of all queries. Then, the average access time of each query should be calculated first in order to obtain the average access time of a query profile. However, it is in general difficult to directly derive the average access time of a query since some data items of the query may have the same placement (i.e., with the same value of function $placement$) in a broadcast program.

To address this difficulty, we propose the procedure query refinement to refine each query Q_i before the calculation of $T_{Access}(Q_i)$ on a broadcast program. For a query Q_i, the refined query $Q_i^{'}$ and an integer $cycleNo$ will be obtained after query refinement. In the rest of this subsection, the details of query refinement will be described first, and the procedure to calculate $T_{Access}(Q_i)$ from $Q^{'}$ and $cycleNo$ will then be given.

Given a broadcast program, all required data items of Q_i are hashed into a hash table H according to their placements. That is, the function $placement$ is taken as the hash function of H. Due to the domain of the function $placement$, H consists of L buckets. Let $H[j]$ represent the contents of the j-th bucket of H, and $|H[j]|$ be the number of the data items in $H[j]$. The set $Q_i^{'}$ is initialized to be an empty set. Let max be "the maximum among all $|H[j]|$". Then, for each bucket $H[k]$ where $|H[k]|$ is equal to max, we randomly select one data item from $H[j]$ and insert the selected data item into $Q_i^{'}$. Moreover, the value of $cycleNo$ is set to be $max - 1$. Finally, $Q_i^{'}$ and $cycleNo$ are returned as the results of the query refinement of Q_i. The algorithmic form of query refinement is as below.

Function QueryRefinement(Q_i, P)
Input: A query Q_i and a broadcast program P.
Output: The refined query $Q_i^{'}$ and the number of necessarily complete cycles $cycleNo$.
1: Hash all required data items of Q_i into H according to their placements.
2: $max \leftarrow \max_{1 \leq j \leq L} \{|H[j]|\}$;
3: $cycleNo = max - 1$;

4: $Q_i^{'} \leftarrow \phi$;
5: **for** ($k = 1$ to L) **do**
6: **if** ($|H[k]| = max$) **then**
7: Randomly select one data item from $H[k]$, and insert the selected data item into $Q_i^{'}$;
8: **end if**
9: **end for**
10: return ($Q_i^{'}$, $cycleNo$);

Example 3: Consider the broadcast program in Figure 4. Suppose that $Q_i = \{D_4, D_6, D_1\}$. We first hash the placements of D_4, D_6 and D_1 into H. $H[1] = \{D_1, D_6\}$ and $|H[1]| = 2$ since $placement(1) = placement(6) = 1$. Similarly, $|H[2]| = 1$ and $|H[3]| = 0$. The value of max is 2 since $|H[1]| = \max\Big\{|H[1]|, |H[2]|, |H[3]|\Big\}$, and therefore $cycleNo = max - 1 = 1$. By the loop in line 5-9, we have $Q_i^{'} = \{D_1\}$ or $Q_i^{'} = \{D_6\}$.

By the definitions and procedure query refinement mentioned above, we have the following lemmas. Interested readers are referred to the Appendix for the proofs of all lemmas. Note that $T_{Access}(Q_i^{'})$ is independent of the selection of data items from $H[j]$ in line 7 of the procedure QueryRefinement.

Lemma 1: Given a query Q_i, a broadcast program and the results of the refinement of Q_i (i.e., $Q_i^{'}$ and $cycleNo$), the average access time of a query Q_i (i.e., $T_{Access}(Q_i)$) can be formulated as

$$T_{Access}(Q_i) = T_{Access}(Q_i^{'}) + cycleNo \times L \times \frac{s}{B}.$$

In Lemma 1, s and L are system parameters which can be obtained easily. As a consequence, the average access time of a query Q_i can be obtained by the results of query refinement of Q_i and Lemma 1. That is to say, after query refinement, $T_{Access}(Q_i)$ can be obtained by $T_{Access}(Q_i^{'})$ and $cycleNo$.

In addition, the refined query $Q_i^{'}$ has the following property.

Lemma 2: Consider a broadcast program, a query Q_i, and the refined query $Q_i^{'}$ of Q_i. Let D_x and D_y represent two randomly selected data items from $Q_i^{'}$. Then, we have $placement(x) \neq placement(y)$.

According to Lemma 2, the placements of all data items of the refined query $Q_i^{'}$ are distinct. This property makes the derivation of $T_{Access}(Q_i^{'})$ possible. The derivation of $T_{Access}(Q_i^{'})$ is as follows.

Without loss of generality, we assume that the user submits a query in the m-th broadcast cycle. We also assume that the offset between the start point of m-th broadcast cycle and the moment of the mobile user submits a query (i.e., s_0) follows a uniform distribution over $(0, L)$. According to Equation (2), we have the following equation.

$$T_{Access}(Q_i^{'}) = T_{Startup}(Q_i^{'}) + T_{Retr.}(Q_i^{'}) \tag{3}$$

Let a series $a(j)$, $1 \leq j \leq |Q_i^{'}|$, represent the *sorted* placements of all required data items of $Q_i^{'}$, and b be $|Q_i^{'}|$. Since the placements of all data items of the refined query $Q_i^{'}$ are distinct (by Lemma 2), $T_{Startup}(Q_i^{'})$ and $T_{Retr.}(Q_i^{'})$ can be formulated by the following lemmas.

Lemma 3: $T_{Startup}(Q_i^{'})$ can be formulated as

$$T_{Startup}(Q_i^{'}) = \frac{s}{B \times L} \times \left\{ \sum_{j=1}^{b-1} \frac{\Big[a(j+1) - a(j)\Big]^2}{2} + \frac{\Big[L - a(b) + a(1)\Big]^2}{2} \right\}.$$

Lemma 4: $T_{Retr.}(Q_i^{'})$ can be formulated as

$$\begin{aligned} T_{Retr.}(Q_i^{'}) = \frac{s}{B \times L} \Big\{ & (L - a(b) + a(1)) \times (a(b) - a(1) + 1) \\ & + \sum_{j=2}^{b} \Big[(a(j) - a(j-1)) \times (L - a(j) + a(j-1) + 1) \Big] \Big\}. \end{aligned}$$

Then, the average access time of $Q_i^{'}$ can be obtained by Lemmas 3, 4 and Equation (3).

We now summarize the derivation of the average access time of a query profile. According to Equation (1), the average access time of a query profile (i.e., $T_{Access}(Q)$) is a weighted summation of

the average access time of each query Q_i (i.e., $T_{Access}(Q_i)$). To calculate $T_{Access}(Q)$, we first refine Q_i by query refinement. The results of refining Q_i are a refined query $Q_i^{'}$ and an integer $cycleNo$. According to Equation (3), the average access time of $Q^{'}$ (i.e., $T_{Access}(Q_i^{'})$) is equal to the summation of the startup time and the retrieve time of $Q_i^{'}$ which can be derived by Lemmas 3 and 4. With $T_{Access}(Q^{'})$ and $cycleNo$, $T_{Access}(Q_i)$ can be obtained by Lemma 1. Finally, after obtaining $T_{Access}(Q_i)$ for each query Q_i, $T_{Access}(Q)$ can be calculated by Equation (1). The algorithmic form of the procedure to obtain $T_{Access}(Q)$ is given in Section 4.1.

4 Design of Genetic Algorithm

4.1 Chromosome Representation and Fitness Evaluation

By the definitions in Section 2, a broadcast program can be transformed into a list as shown in Figure 6. Partially mapped crossover (abbreviated as PMX) [9] is a widely-used crossover method when the chromosomes are encoded as lists, and is taken as the crossover operator. Here, we use swap operator as the mutation operator. Fitness is the measurement of the quality of the chromosomes, and the GA is designed to search the chromosome with the highest fitness (i.e, maximize the fitness). Since the goal of dependent data broadcast is to minimize the average access time of the given query profile, the fitness function is defined as $Fitness(P) = \frac{1}{T_{Access}(Q)}$.

According to Equation (1), $T_{Access}(Q)$ is a weighted summation of all $T_{Access}(Q_i)$. In addition, $T_{Access}(Q_i)$ for each query Q_i can be calculated on the basis of the theoretical results derived in Section 3. Hence, $T_{Access}(Q)$ can be obtained by Equation (1) after the calculation of $T_{Access}(Q_i)$ for each query Q_i. The algorithmic forms of the calculations of $T_{Access}(Q_i)$ and $T_{Access}(Q)$ are as below. After

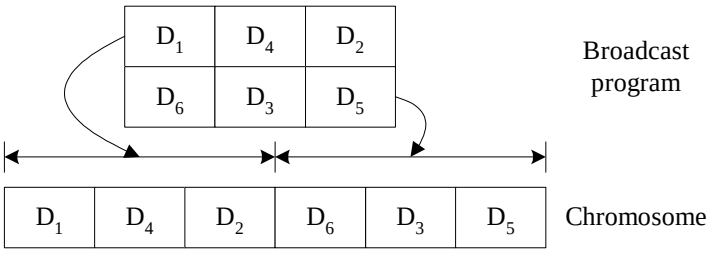

Figure 6: The mapping of a broadcast program and its corresponding chromosome

$T_{Access}(Q)$ is calculated, the fitness value of each chromosome can be obtained based on the definition of fitness function.

Procedure CalAccessTime(Q, P)
Input: A query profile Q and a broadcast program P.
Output: $T_{Access}(Q)$ over the broadcast program P.

1: $T_{Access}(Q) \leftarrow 0$
2: **for** $i = 1$ to $|Q|$ **do**
3: $\quad T_{Access}(Q) \leftarrow T_{Access}(Q) + CalAccessTimeOfQuery(Q_i, P) \times Pr(Q_i)$
4: **end for**
5: return $T_{Access}(Q)$

Function CalAccessTimeOfQuery(Q_i, P)
Input: A query Q_i and a broadcast program P.
Output: $T_{Access}(Q_i)$ on the broadcast program P.

1: ($Q_i^{'}$, $cycleNo$)←QueryRefinement(Q_i, P) /* Refine the query Q_i */
2: Calculate $T_{Startup}(Q_i^{'})$ and $T_{Retr.}(Q_i^{'})$ in accordance with the results of Lemma 3 and 4
3: $T_{Access}(Q_i^{'}) \leftarrow T_{Startup}(Q_i^{'}) + T_{Retr.}(Q_i^{'})$
4: return $T_{Access}(Q_i^{'}) + cycleNo \times L \times \frac{s}{B}$ /* According to Lemma 1 */

4.2 Complexity Analysis

In this paper, we terminate the GA after a specified number of generations (n_{Gen}). It is intuitive that the optimal solution can be obtained by exhaustive search. However, when exhaustive search is employed, the time complexity is in proportion to the size of search space (i.e., $O(|D|!)$) which makes exhaustive search infeasible. On the other hand, for a given population size n_{Pop} after n_{Gen} generations, the size of the search space of the proposed GA is reduced to $O(n_{Pop} \times n_{Gen})$. It is noted that the time complexity is dominated by the number of objective function evaluations. Since the time complexity of the objective

Parameters	Values
The size of population (n_{Pop})	4
The probability of crossover (P_C)	0.5
The probability of mutation (P_M)	0.5
The size of each data item (s)	8K bytes
The bandwidth of each broadcast channel (B)	80K bytes/sec.

Table 1: System Parameters in the example

function is $O(|Q| \times |D|)$, the overall time complexity of the proposed GA is $O(n_{Pop} \times n_{Gen} \times |Q| \times |D|)$.

Next, we analyze the space complexity of the GA. There are two main data structures in the GA. One is used to store the whole query profile, and therefore, the size of this data structure is $O(|Q| \times AvgQLen)$, where $AvgQLen$ represents the average query length. The other data structure is used to store the chromosomes. Since there are n_{Pop} chromosomes and the length of one chromosome is equal to $L \times n = |D|$, the size of this data structure is $O(n_{Pop} \times |D|)$. As a result, the overall space complexity of the proposed GA is $O(|Q| \times AvgQLen + n_{Pop} \times |D|)$.

4.3 An Illustrative Example

Figure 7 shows an example to illustrate the execution of the proposed algorithm. The query profile is shown in Figure 3 and the system parameters of this example are given in Table 1. Note that only the process of the first generation of the proposed GA is described in Figure 7, since GA is iterative and the process of each iteration (i.e., generation) is similar to one another.

In the beginning, since the initial population is empty, four chromosomes (i.e., C_1, C_2, C_3 and C_4) are randomly generated as shown in Figure 7a. Then, the fitness values of all chromosomes are calculated in fitness evaluation phase. Consider the fitness evaluation of Q_1 on C_1. After query refinement, we have $Q_1^{'} = \{D_1, D_2\}$ and $cycleNo = 1$. In accordance with Lemma 1 and Equation (3), $T_{Startup}(Q_1^{'})$ and $T_{Retr.}(Q_1^{'})$ should be calculated before the calculation of $T_{Access}(Q_1)$. By Lemma

ID	Representation	Fitness	Selection Prob.
C_1	$\{D_1, D_2, D_6, D_4, D_3, D_5\}$	1.86	25.41%
C_2	$\{D_4, D_2, D_6, D_1, D_5, D_3\}$	1.82	24.863%
C_3	$\{D_2, D_3, D_1, D_5, D_6, D_4\}$	1.82	24.863%
C_4	$\{D_4, D_1, D_5, D_3, D_6, D_2\}$	1.82	24.863%

(a) Initial population

ID	Representation	Fitness	Selection Prob.
C_1	$\{D_1, D_2, D_6, D_4, D_3, D_5\}$	1.86	25.272%
C_2	$\{D_4, D_2, D_6, D_1, D_5, D_3\}$	1.82	24.728%
C_1	$\{D_1, D_2, D_6, D_4, D_3, D_5\}$	1.86	25.272%
C_3	$\{D_2, D_3, D_1, D_5, D_6, D_4\}$	1.82	24.728%

(b) The result of selection

ID	Representation	Fitness	Selection Prob.
C_5	$\{D_4, D_2, D_6, D_1, D_3, D_5\}$	1.86	25.272%
C_6	$\{D_1, D_2, D_6, D_4, D_5, D_3\}$	1.82	24.728%
C_1	$\{D_1, D_2, D_6, D_4, D_3, D_5\}$	1.86	25.272%
C_3	$\{D_2, D_3, D_1, D_5, D_6, D_4\}$	1.82	24.728%

(c) The result of crossover

ID	Representation	Fitness	Selection Prob.
C_5	$\{D_4, D_2, D_6, D_1, D_3, D_5\}$	1.86	24.668%
C_7	$\{D_6, D_1, D_2, D_5, D_4, D_3\}$	1.86	24.668%
C_1	$\{D_1, D_2, D_6, D_4, D_3, D_5\}$	1.86	24.668%
C_8	$\{D_2, D_3, D_6, D_1, D_5, D_4\}$	1.96	25.996%

(d) The result of mutation

Figure 7: The first generation of the proposed GA

D_1	D_5	D_2
D_6	D_4	D_3

Figure 8: The result broadcast program of the example

3, we can obtain $T_{Startup}(Q_1^{'}) = \frac{8}{80\times3} \times \left[\frac{(2-1)^2}{2} + \frac{(3-2+1)^2}{2}\right] = \frac{1}{12}$. Similarly, we have $T_{Retr.}(Q_1^{'}) = \frac{8}{80\times3} \times \Big[(3-2+1) \times (2-1+1) + (2-1) \times (3-2+1+1)\Big] = \frac{7}{30}$, and $T_{Access}(Q_1^{'}) = T_{Startup}(Q_1^{'}) + T_{Retr.}(Q_1^{'}) = \frac{1}{12} + \frac{7}{30} = \frac{19}{60}$. Finally, we have $T_{Access}(Q_1) = \frac{19}{60} + 1 \times 3 \times \frac{8}{80} = \frac{37}{60} = 0.6167$.

By similar approach, we have $T_{Access}(Q_2) = T_{Access}(Q_3) = \frac{11}{20} = 0.55$ and $T_{Access}(Q_4) = \frac{7}{20} = 0.35$. On the basis of Lemma 1, the average access time of the query profile on C_1 is $T_{Access}(Q) = 0.4 \times 0.6167 + 0.2 \times 0.55 + 0.2 \times 0.55 + 0.2 \times 0.35 = 0.53668$. The fitness value of C_1 is equal to $\frac{1}{0.53668} = 1.86$. Similarly, the fitness values of C_2, C_3 and C_4 are as shown in Figure 7a.

In selection phase, the selection probability of C_1 is equal to $\frac{1.86}{1.86+1.82+1.82+1.82} = 25.41\%$. Similarly, the selection probabilities of other chromosomes are as shown in Figure 7a. Then the chromosome selection executes four (i.e., the value of n_{Pop}) times, and in each execution of chromosome selection, C_1, C_2,C_3 and C_4 are selected with probabilities 25.41%, 24.863%, 24.863% and 24.863%, respectively. Note that one chromosome can be selected multiple times. The result of selection in this example is shown in Figure 7b.

In crossover phase, on the average, 50% of chromosomes are selected to crossover since $P_C = 0.5$. We assume that C_1 and C_2 are selected to crossover. After the execution of crossover operator (PMX in this example), two offspring of C_1 and C_2 (i.e., C_5 and C_6) are generated and the result of crossover is described in Figure 7c. Interested readers are referred to [9] for the details of PMX. In mutation phase, we assume that C_3 and C_6 are selected to mutate, and the result of mutation is shown in Figure 7d. Finally, the chromosome with largest fitness value is recorded as the best chromosome. In this iteration, the best chromosome is $C_8 = \{D_2, D_3, D_6, D_1, D_5, D_4\}$. A better chromosome $\{D_1, D_5, D_2, D_6, D_4, D_3\}$ can be obtained by successive iterations. This chromosome is then transformed into the broadcast program as shown in Figure 8.

Parameters	Values
The number of generations (n_{Gen})	200
The size of population (n_{Pop})	20
The probability of crossover (P_C)	0.5
The probability of mutation (P_M)	0.5
The size of each data item (s)	8K bytes
The bandwidth of each broadcast channel (B)	80K bytes/sec.
The number of data items ($\|D\|$)	1000
The number of queries ($\|Q\|$)	500
The average query length	15
The Zipf parameter	0.4

Table 2: System Parameters

5 Performance Evaluation

5.1 The System Model

We implement our GA-based algorithm with GAlib [27]. In addition, we also implement a query profile generator based on the approach mentioned in [21]. The query profile generator contains the following adjustable parameters: (1) the size of database ($|D|$), (2) the number of queries ($|Q|$), (3) the average length of queries and (4) the fanout of each data item. The dependency among data items is modeled as a dependency graph. The fanout of each data item represents the degree of the corresponding vertex in the dependency graph. The probability of the query Q_i issued by users is assumed to be $Pr(Q_i) = \frac{(\frac{1}{i})^\theta}{\sum_{j=1}^{|Q|}(\frac{1}{j})^\theta}$ where θ is the parameter of the Zipf [31] distribution. Table 2 shows the system parameters in our experiments. The simulator and query profile generator are coded in C++.

In addition to the proposed GA, named as scheme GA, we also implement scheme VF^K[22], GREEDY and OPT for comparison purposes. Scheme VF^K is an effective algorithm which can efficiently generate a broadcast program in multiple channels. As shown in [22], the results of VF^K are very close to the optimal ones if there is no dependency among data items. Scheme OPT is an exhaustive search algorithm which is able to obtain the optimal broadcast programs.

Scheme GREEDY is a greedy algorithm which generates a broadcast program for unordered queries. The rationale of scheme GREEDY is to place the data items within the same query closely since these data items are likely to be requested together. Scheme GREEDY takes the chromosome proposed in Section 4.1 as the representation of a broadcast program. The details of scheme GREEDY are as follows. First, all queries in the query profile are sorted according to their access probabilities in descending order. In addition, the resulting chromosome is initialized as empty and all data items are marked as UNPLACED states. Scheme GREEDY then considers each query according to the sorted order. In the consideration of a query, say Q_i, data items in PLACED states are removed from Q_i since their placements are determined (i.e., they have been placed). Then, the remaining data items in Q_i are appended to the chromosome. In addition, these data items are marked as PLACED states. Scheme GREEDY considers the queries in the query profile in turn until all data items are in PLACED states. Finally, scheme GREEDY transforms the resulting chromosome into the corresponding broadcast program and takes this broadcast program as the result.

We use the query profile shown in Figure 3 to illustrate the execution of scheme GREEDY. Initially, all data items are in UNPLACED states and the resulting chromosome is set to be empty. The order that scheme GREEDY considers these queries is Q_1, Q_2, Q_3 and Q_4. Note that the order of Q_2, Q_3 and Q_4 are randomly determined since $Pr(Q_2) = Pr(Q_3) = Pr(Q_4)$. When considering Q_1, since D_1, D_2, D_3 and D_4 are in UNPLACED states, scheme GREEDY appends them into the resulting chromosome and marks them as PLACED states. Hence, the resulting chromosome after the consideration of Q_1 is $\{D_1, D_2, D_3, D_4\}$. Then, scheme GREEDY considers Q_2. Nothing happens since all data items in Q_2 are in PLACED states. When considering Q_3, scheme GREEDY appends D_6 to the resulting chromosome. Then, the resulting chromosome becomes $\{D_1, D_2, D_3, D_4, D_6\}$. Similarly, the resulting

chromosome is $\{D_1, D_2, D_3, D_4, D_6, D_5\}$ after the consideration of Q_4. Finally, the resulting broadcast program can be obtained by transforming the resulting chromosome by the steps shown in Figure 6.

For performance comparison, the performance gain of scheme A over scheme B is defined as

$$\frac{\text{Avg. Access Time of scheme B} - \text{Avg. Access Time of scheme A}}{\text{Avg. Access Time of scheme B}}.$$

In addition, we use performance degradation as the performance metric when comparing scheme A and scheme OPT. The performance degradation of scheme A over scheme OPT is defined as

$$\frac{\text{Avg. Access Time of scheme A} - \text{Avg. Access Time of scheme OPT}}{\text{Avg. Access Time of scheme OPT}}.$$

5.2 Experimental Results

5.2.1 The Evolution of Scheme GA

As mentioned in Section 4, scheme GA is an iterative process. The quality of obtained result and execution time is dependent on the value of n_{Gen}. Obviously, the obtained result is of better quality when n_{Gen} is larger. However, a large n_{Gen} implies long execution time. Hence, it is important to determine a proper value of n_{Gen} to strike a balance between the execution time and the quality of result. In this experiment, we investigate the evolution process of scheme GA by varying the value of n_{Gen}. Figure 9 shows the effect of different values of n_{Gen} ranging from 0 to 500 on the average access times of the resulting broadcast programs. The corresponding execution times of all schemes are presented in Figure 10. Note that $n_{Gen} = 0$ represents the case of randomly generating n_{Pop} solutions.

As shown in Figure 9, since scheme OPT, GREEDY and VF^K are independent of n_{Gen}, the average access times of the result broadcast programs of these schemes are not affected by n_{Gen}. We observe

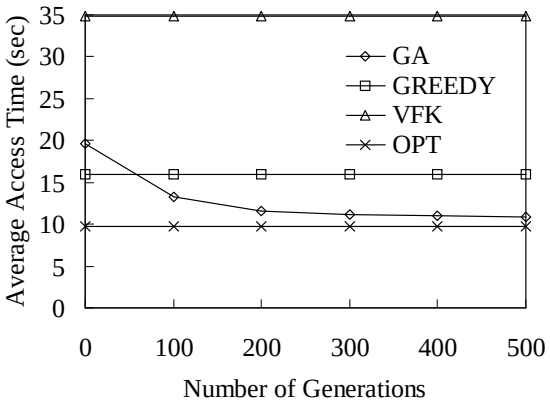

Figure 9: The average access time with n_{Gen} varied

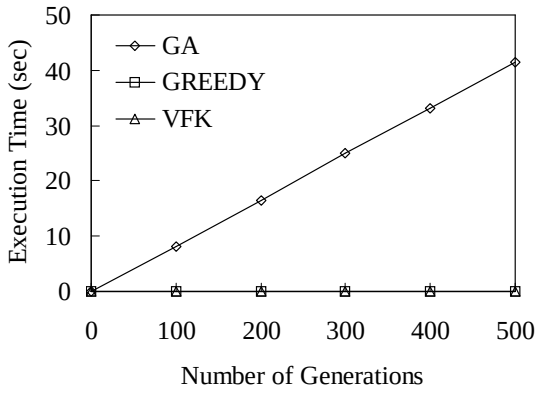

Figure 10: The execution time with n_{Gen} varied

that the results of VF^K are much worse than that of other schemes, and hence, the results of VF^K are omitted henceforth. Although the results of VF^K are close to the optimal ones when the data items are independent of one another, scheme VF^K does not perform well due to the lack of the consideration of data dependency. This result shows the necessity of the consideration of data dependency. We also observe that scheme GREEDY outperforms scheme VF^K. This can be explained by the reason that scheme GREEDY considers the data dependency by placing the required data items of the queries with higher access probabilities as closely as possible.

The average access times of the broadcast programs of scheme GA decrease as the value of n_{Gen} increases. In addition, the results obtained by scheme GA become closer to the optimal ones when the value of n_{Gen} is larger. However, the speed of convergence becomes slow when n_{Gen} is larger than 200. Therefore, we set n_{Gen} to be 200 in the following experiments. We observe that when n_{Gen} is small, scheme GREEDY outperforms scheme GA. However, when n_{Gen} is large enough (larger than 80 in this experiment), the results of scheme GA are better than that of scheme GREEDY. The performance gain of scheme GA over GREEDY increases from 17% to 31.4% when n_{Gen} increases from 100 to 500.

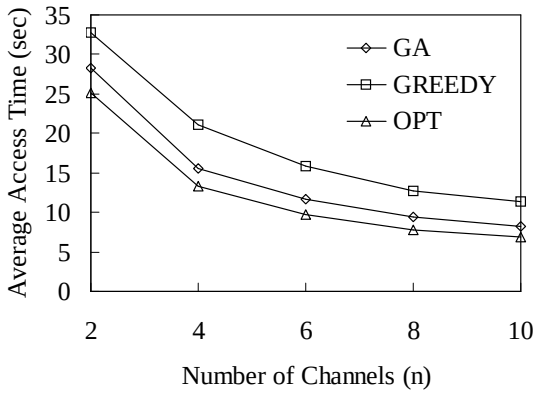

Figure 11: The average access time with n varied

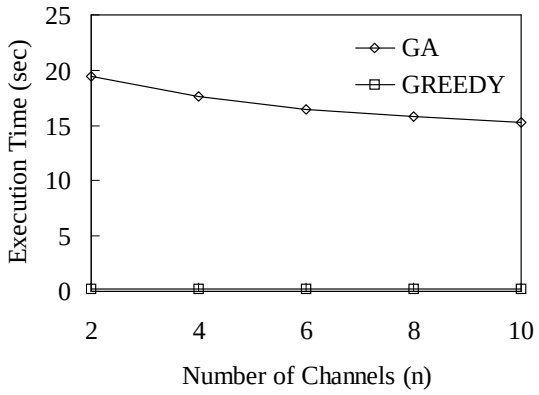

Figure 12: The execution time with n varied

As shown in Figure 10, the execution time of scheme GA increases linearly as the value of n_{Gen} increases, which fully agrees with the time complexity analysis in Section 4.2. Since scheme VF^K and GREEDY are simple heuristics, they execute much faster than scheme GA. In addition, since the execution time of scheme OPT is much longer than that of other schemes, the execution time of scheme OPT is omitted in this and the following experiments.

5.2.2 The Effect of the Number of Channels

This experiment investigates the effect of the number of channels in the average access times and the execution times of all schemes. Figures 11 and 12 show the average access times and execution times of scheme GA, GREEDY and OPT with the number of broadcast channels (i.e., n) varied. The value of n ranges from 2 to 10.

Consider the results shown in Figure 11. The average access times of all schemes decrease as the number of channels increases. This result agrees with our intuition in that the increase of the network bandwidth causes the average access time to decrease. However, the improvement on the average access time decreases as the number of channels increases. As a result, the determination of the number of

broadcast channels should consider the balance between performance improvement and the number of channels used. The number of broadcast channels suggested by our experiment is around 6. The performance gain of scheme GA over scheme GREEDY increases from 13.33% to 25.67% when the number of channels increases from 2 to 10. This result shows that scheme GA outperforms scheme GREEDY especially when the number of channels is large.

As shown in Figure 12, the execution time of scheme GREEDY is not affected by the number of channels. However, the execution time of scheme GA decreases as the number of channels increases. This result is caused by query refinement. By Definition 3, a large value of n implies a short broadcast program (i.e., a small value of L). According to function QueryRefinement, the number of the required data items of each refined query $Q_i^{'}$ is always smaller than or equal to L. The decrease of L will incur short refined queries and therefore decrease the time to calculate $T_{Startup}(Q_i^{'})$ and $T_{Retr.}(Q_i^{'})$.

5.2.3 Comparison of Single and Multiple Channel Environments

We next investigate the effect of the number of channels by fixing the total available bandwidth. We set the total bandwidth to be 800K bytes/sec, and the number of channels is set to be from 1 to 10. The average access times of all schemes with the number of channels varied are given in Figure 13.

As observed, the average access times of all schemes increase as the number of channels increases. This result agrees with the intuition that in a single environment, all bandwidth can be best utilized to minimize the average access time. The utilization of bandwidth degrades in a multiple channel environment since a mobile client can only listen on one channel at a time. Multiple channels also increase the difficulty to minimize the average access time. Hence, the performance degradation of scheme GA over scheme OPT increases from 0.96% to 8.02% as the number of channels increases

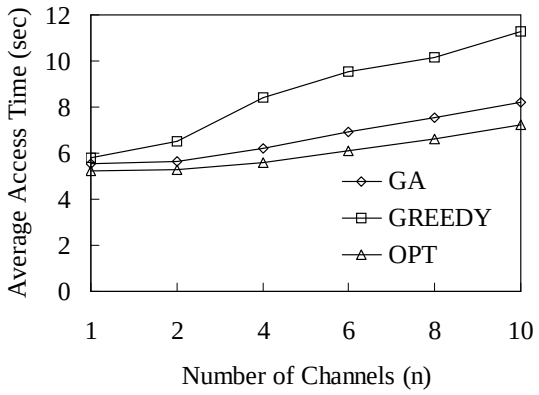

Figure 13: The average access time of single and multiple channel(s) with fixed total bandwidth

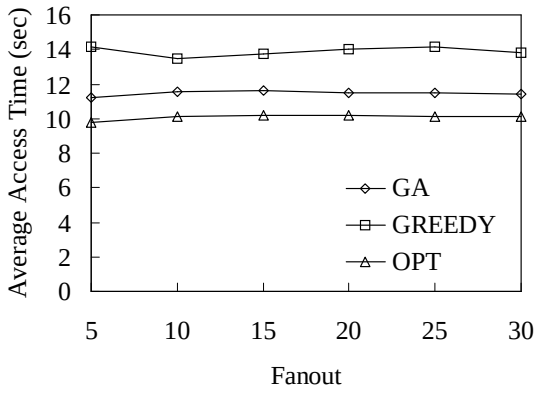

Figure 14: The average access time with the value of fanout varied

from 1 to 10. Under the same condition, it is interesting to see that the performance degradation of scheme GREEDY over scheme OPT increases from 5.64% to 48.88%. This is explained by the reason that scheme GREEDY only tries to place data items with the same query as closely as possible and does not consider the effect of multiple channels. As a result, the performance degradation of scheme GREEDY over scheme OPT becomes more severe when the number of channels increases.

5.2.4 The Effect of Average Fanout of Data Items

This experiment evaluates the effect of the average fanout of data items. Figure 14 shows the average access times of all schemes with the value of average fanout varied. The value of average fanout is set to be from 5 to 30. As shown in Figure 14, the average fanout of data items only slightly affects the results of all schemes. The average access time of scheme OPT ranges from 9.8 to 10.2. In addition, scheme GA outperforms scheme GREEDY in this experiment. The performance gain of scheme GA over scheme GREEDY ranges from 16.62% to 26.41%.

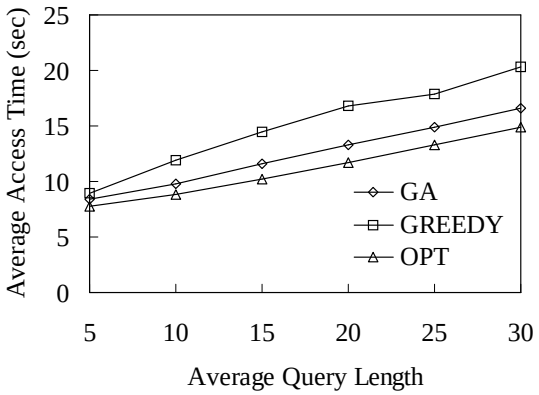

Figure 15: The average access time with average query length varied

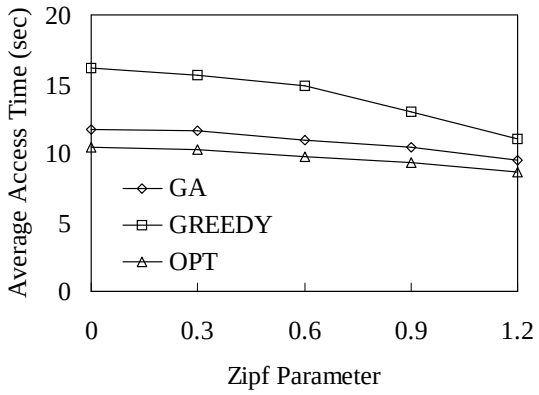

Figure 16: The average access time with θ varied

5.2.5 The Effect of Average Query Length

In this experiment, we investigate the effect of the average query length. Figure 15 shows the average access times of all schemes with the value of average query length varied. The value of average query length is set from 5 to 30. It is intuitive that the average access time increases as the value of average query length increases. We observe that the performance degradation of scheme GA over scheme OPT increases from 8.24% to 13.67% when the value of average query length increases from 5 to 15. In addition, the performance degradation of scheme GA over scheme OPT keeps in the range between 11.5% and 13.5% when the average query length is larger than 15. On the other hand, the performance degradation of scheme GREEDY over scheme OPT increases from 15.11% to 41.95%. In addition, the performance degradation of scheme GREEDY over scheme OPT keeps in the range between 34% and 43.5% when the average query length is larger than 15. It can be explained that the optimization constraints are relaxed when the value of query length is small. Hence, all schemes perform well with a small value of average query length. In addition, scheme GA outperforms scheme GREEDY especially when the value of average query length is large. Since scheme GREEDY is a simple heuristic, the

performance of scheme GREEDY degrades severer than that of scheme GA when the optimization constraints are strict (i.e., the value of average query length is large).

5.2.6 The Effect of the Skewness of Queries

In this experiment, we consider the effect of the skewness of the access probabilities of queries. The skewness is controlled by the value of θ. The larger the value of θ, the more skewed the access probabilities of the queries are. The value of θ is set to be from 0 to 1.2. Note that $\theta = 0$ indicates that the access probabilities of all queries are uniform (i.e., $Pr(Q_i) = Pr(Q_j)$ for all i and j).

Figure 16 shows the average access times of all schemes with θ varied. We observe that the average access times of all schemes decrease as the value of θ increases. It is because that when the access probabilities are not skewed, minimizing the average access time is not effective since more queries are involved. When the access probabilities of queries are skewed, optimizing hot queries is able to minimize the average access time well. Hence, scheme GREEDY performs better when the access probabilities are highly skewed since scheme GREEDY favors queries with high access probabilities. As a result, the performance gain of scheme GA over scheme GREEDY decreases from 27.79% to 13.73% as the value of θ increases from 0 to 1.2.

5.2.7 Summary

Based on the above experimental results, the characteristics of scheme GA and GREEDY are summarized in this subsection according to the following respects.

- *Result effectiveness:* The relative performance of schemes depends on the value of n_{Gen}. If n_{Gen} is large enough (i.e., given sufficient time to execute), scheme GA is more effective than scheme

GREEDY. However, if the time to execute is not sufficient (i.e., n_{Gen} is too small), scheme GREEDY outperforms scheme GA.

- *Execution speed:* Scheme GREEDY is faster than scheme GA due to the simplicity of scheme GREEDY.

- *Performance stability:* If n_{Gen} is large enough (i.e., given sufficient time to execute), the results of scheme GA are always close to the optimal ones. On the other hand, the performance of the results of scheme GREEDY depends on several factors. For example, as pointed out above, scheme GREEDY does not perform well when the access probabilities of queries are not skewed. Hence, the performance of scheme GA is more stable than that of scheme GREEDY.

- *Convergence:* Since genetic algorithms are convergent processes, the results of scheme GA become better when the allowed time to execute is longer. On the contrary, the results of scheme GREEDY are not affected by the allowed time to execute.

5.3 Adaptation Ability of Scheme GA

We next investigate the adaptation ability of scheme GA against the change of the query profile. To adapt the change of the query profile, an adaptive version of scheme GA (referred to as scheme Adaptive-GA) is proposed. In scheme Adaptive-GA, scheme GA is activated periodically, and we call each activation a run. In each run, scheme GA is executed for a predetermined time interval. The resulting population of the current run is taken as the initial population of the next run.

According to the observations in Section 5.2.7, the relative merits of scheme GREEDY and GA are in fact complementary rather than competitive. Therefore, we propose a scheme named Augmented-

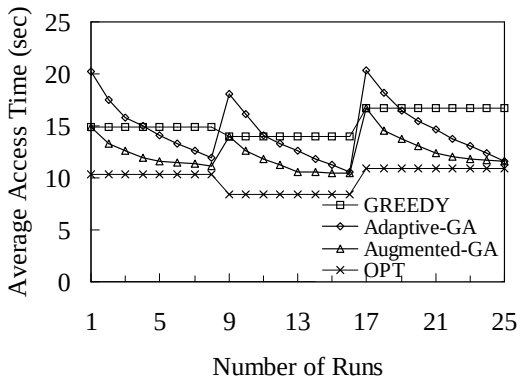

Figure 17: The average access time with time varied

GA to take advantage of the respective merits of both schemes. In essence, scheme Augmented-GA is a revised version of scheme Adaptive-GA. In each run of scheme Augmented-GA, scheme GREEDY is executed and the result of scheme GREEDY is inserted into the population of the current run.

To compare the adaptation ability of scheme Adaptive-GA and Augmented-GA on the change of the query profile, we execute scheme GREEDY and scheme OPT in each run. We assume the time interval between runs to be one hour. The allowed time to execute scheme GA for each run is set to six seconds. In the first run, the value of θ is set to be 0.6. The value of θ is changed to 1.2 after a period which is, without loss of generality, eight hours. The order of the access probabilities of all queries is also changed. The value of θ is changed to 0.2 after more eight hours. The experiment executes for 24 hours. The average access times of scheme GREEDY, OPT, Adaptive-GA and Augmented-GA with time varied are shown in Figure 17.

In the first run, since the allowed time to execute scheme GA is short, scheme GREEDY outperforms scheme Adaptive-GA. Since the result of scheme GREEDY is inserted into the population of scheme Augmented-GA, the result of scheme Augmented-GA is at least as good as that of scheme GREEDY. In the successive runs (from the second run to the eighth run), due to the convergence prop-

erty of GAs, the results of schemes Adaptive-GA and Augmented-GA become better and better. We observe that scheme Adaptive-GA outperforms scheme GREEDY after the fifth run. Due the insertion of the result of scheme GREEDY, the convergence speed of scheme Augmented-GA is faster than that of scheme Adaptive-GA. Hence, scheme Augmented-GA outperforms scheme Adaptive-GA. The results of schemes GREEDY and OPT remain stable in the first eight runs since the query profile is not changed.

The first change of the query profile occurs in the ninth run. As the results shown in Section 5.2.6, the increase of skewness will increase the average access time. Hence, the average access times of schemes GREEDY and OPT decrease after the ninth run. Scheme Adaptive-GA is able to adapt the change of the query profile. However, since the allowed time to execute scheme GA is short, scheme Adaptive-GA in the ninth run is outperformed by scheme GREEDY. The results of schemes Adaptive-GA and Augmented-GA improve in the subsequent runs until the second change of the query profile. Since the second change of the query profile occurs in the 17th run, the scenario between the 17th run and the 25th run is similar to that between the ninth run and the 16th run.

This experiment result shows that scheme Augmented-GA can successfully take advantages of the respective merits of scheme GREEDY and scheme GA to achieve better performance than pure GA-based schemes. In fact, with scheme Augmented-GA, one can either decrease the period of each run or increase the execution time to speed up adaptation and convergence.

6 Conclusion

Data broadcast is an important data dissemination technique on mobile computing environments. Generating broadcast programs to effectively reduce the average waiting time is an important issue of data

broadcast. We explored in this paper the problem of broadcasting dependent data in multiple broadcast channels for unordered queries. By analyzing the model of dependent data broadcasting, we derived several theoretical properties for the average access time in a multiple channel environment. In light of the theoretical results, we developed scheme GA to generate broadcast programs for unordered queries. To evaluate the effectiveness of scheme GA, we developed scheme OPT which is able to generate the optimal broadcast programs by exhaustive search. In addition, we also designed scheme GREEDY to efficiently generate broadcast programs for unordered queries in a greedy manner.

To measure the performance of scheme GA, several experiments were then conducted. Our experimental results showed that the theoretical results derived are able to guide the search of the genetic algorithm very effectively, thus leading to broadcast programs of very high quality. Specifically, the performance of the broadcast programs generated by scheme GA is close to that generated by scheme OPT (i.e., the performance of the optimal broadcast programs). After summarizing the experimental results, we observed that the relative merits of scheme GA and scheme GREEDY are in fact complementary rather than competitive. As a result, we developed scheme Augmented-GA to take advantage of the respective merits of both schemes. Experimental results showed that scheme Augmented-GA can achieve better performance than pure GA-based schemes.

References

[1] S. Acharya, M. J. Franklin, and S. Zdonik. Dissemination-based Data Delivery Using Broadcast Disks. *IEEE Personal Communications*, 2(6), December 1995.

[2] S. Acharya and S. Muthukrishnan. Scheduling on-demand Broadcasts: New Metrics and Algorithms. In *Proceedings of the 4th ACM/IEEE International Conference on Mobile Computing and Networking*, pages 43–94, October 1998.

[3] D. Aksoy and M. J. Franklin. Scheduling for Large-Scale On-Demand Data Broadcasting. In *Proceedings of IEEE INFOCOM Conference*, pages 651–659, March 1998.

[4] A. Bar-Noy, J. Naor, and B. Schieber. Pushing Dependent Data in Clients-Providers-Servers Systems. In *Proceedings of 6th ACM/IEEE International Conference on Mobile Computing and Networking*, pages 222–230, August 2000.

[5] A. Bar-Noy and Y. Shilo. Optimal Broadcasting of Two Files over an Asymmetric Channel. In *Proceedings of the IEEE INFOCOM Conference*, pages 267–274, March 1999.

[6] Y. C. Chehadeh, A. R. Hurson, and M. Kavehrad. Object Organization on a Single Broadcast Channel in the Mobile Computing Environment. *Multimedia Tools and Applications*, 9(1):69–94, July 1999.

[7] M.-S. Chen, K.-L. Wu, and P. S. Yu. Optimizing Index Allocation for Sequential Data Broadcasting in Wireless Mobile Computing. *IEEE Transactions on Knowledge and Data Engineering*, 15(1), February 2003.

[8] Y. D. Chung and M. H. Kim. Effective Data Placement for Wireless Broadcast. *Distributed and Parallel Databases*, 9(2):133–150, March 2001.

[9] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, 1989.

[10] V. Gondhalekar, R. Jain, and J. Werth. Scheduling on Airdisks: Efficient Access to Personalized Information Services via Periodic Wireless Data Broadcast. In *Proceedings of the IEEE ICC Conference*, June 1997.

[11] C.-L. Hu and M.-S. Chen. Dynamic Data Broadcasting with Traffic Awareness. In *Proceedings of the 22th IEEE International Conference on Distributed Computing and Systems*, July 2002.

[12] Q. L. Hu, D. L. Lee, and W.-C. Lee. Dynamic Data Delivery in Wireless Communication Environments. In *Proceedings of International Workshop on Mobile Data Access*, pages 218–229, November 1998.

[13] Q. L. Hu, W.-C. Lee, and D. L. Lee. Indexing Techniques for Wireless Data Broadcast under Data Clustering and Scheduling. In *Proceedings of the 8th ACM International Conference on Information and Knowledge Management*, pages 351–718, November 1999.

[14] J.-L. Huang, W.-C. Peng, and M.-S. Chen. Binary Interpolation Search for Solution Mapping on Broadcast and On-demand Channels in a Mobile Computing Environment. In *Proceedings of the 10th ACM International Conference on Information and Knowledge Management*, November 2001.

[15] A. R. Hurson, Y. C. Chehadeh, and J. Hannan. Object Organization on Parallel Broadcast Channels in a Global Information Sharing Environment. In *Proceedings of the 19th IEEE International Performance, Computing, and Communications Conference*, pages 347–353, February 2000.

[16] T. Imielinski and S. Viswanathan. Adaptive Wireless Information Systems. In *Proceedings of SIGDBS (Special Interest Group in DataBase Systems) Conference*, October 1994.

[17] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Data on Air: Organization and Access. *IEEE Transactions on Knowledge and Data Engineering*, 9(9):353–372, June 1997.

[18] W.-C. Lee, Q. L. Hu, and D. L. Lee. A Study on Channel Allocation for Data Dissemination in Mobile Computing Environments. *ACM/Baltzer Mobile Networks and Applications*, 4(5):117–129, May 1999.

[19] C.-W. Lin and D. L. Lee. Adaptive Data Delivery in Wireless Communication Environments. In *Proceedings of the 20th IEEE International Conference on Distributed Computing Systems*, pages 444–456, April 2000.

[20] S.-C. Lo and A. L. P. Chen. Optimal Index and Data Allocation in Multiple Broadcast Channels. In *Proceedings of the 16th International Conference on Data Engineering*, pages 293–702, March 2000.

[21] A. Nanopoulos, D. Katsaros, and Y. Manolopoulos. Effective Prediction of Web-user Accesses: A Data Mining Approach. In *Proceedings of the WEBKDD Workshop,* August 2001.

[22] W.-C. Peng and M.-S. Chen. Efficient Channel Allocation Tree Generation for Data Broadcasting in a Mobile Computing Environment. *ACM Wireless Networks*, 9(2):117–129, 2003.

[23] K. Prabhakara, K. A. Hua, and J. H. Oh. Multi-Level Multi-Channel Air Cache Designs for Broadcasting in a Mobile Environment. In *Proceedings of the 16th International Conference on Data Engineering*, pages 167–186, February-March 2000.

[24] N. Shivakumar and S. Venkatasubramanian. Efficient Indexing for Broadcast Based Wireless Systems. *ACM/Baltzer Mobile Networks and Applications*, 4(6):433–446, January 1996.

[25] C.-J. Su and L. Tassiulas. Joint Broadcast Scheduling and User's Cache Management for Efficient Information Delivery. In *Proceedings of the 4th ACM/IEEE International Conference on Mobile Computing and Networking*, pages 33–42, October 1998.

[26] D. A. Tran, K. Hua, and K. Prabhakaran. On The Efficient Use of Multiple Physical Channel Air Cache. In *Proceedings of the IEEE Wireless Communications and Networking Conference*, 17-21 2002.

[27] M. Wall. GAlib: A C++ Library of Genetic Algorithm Components. *http://lancet.mit.edu/ga*, August 1996.

[28] J. L. Xu, Q. L. Hu, D. L. Lee, and W.-C. Lee. SAIU: An Efficient Cache Replacement Policy for Wireless On-demand Broadcasts. In *Proceedings of the 9th ACM International Conference on Information and Knowledge Management*, pages 46–53, November 2000.

[29] J. L. Xu, D. L. Lee, and B. Li. On Bandwidth Allocation for Data Dissemination in Celluar Mobile Networks. *to appear in ACM Wireless Networks*.

[30] J. X. Yu, T. Sakata, and K. L. Tan. Statistical Estimation of Access Frequencies in Data Broadcasting Environments. *ACM/Baltzer Wireless Networks*, 6(2):89–98, March 2000.

[31] G. K. Zipf. *Human Behaviour and the Principle of Least Effort*. Addison Wesley, 1949.

Appendix: Proofs of All Lemmas

Proof of Lemma 1: Consider the procedure QueryRefinement in Section 3.2. Since $cycleNo$ is defined as "the maximum of all $|H[j]|$"-1, the maximal number of data items in all $H[j]$ for $1 \leq j \leq L$ is $cycleNo + 1$. Since the mobile device can only listen to at most one broadcast channel at a time, the mobile device can retrieve at most one data item in $H[j]$ in one *full* broadcast cycle (i.e., the mobile device listens on the broadcast channels for $L \times \frac{s}{B}$). Therefore, to read all required data items of Q_i, the mobile device should listen on the broadcast channels at least $cycleNo$ *full* broadcast cycles.

Suppose that the mobile user submits Q_i in the m-th broadcast cycle, and the distance between the starting point of m-th broadcast cycle and the moment that the mobile user submits Q_i to be x. Assume that x follows a uniform distribution over $(0, L)$. After listening on the broadcast channels for $cycleNo$ full broadcast cycles, the mobile device should then retrieve data items in $Q_i^{'}$ in the $(m + cycleNo + 1)$-st broadcast cycle. Denote the access time when the mobile user submits Q_i in offset x as $t(Q, x)$. $T_{Access}(Q_i)$ can be formulated as

$$\begin{aligned} T_{Access}(Q_i) &= \frac{1}{L} \times \left[\int_0^L t(Q_i, x) dx \right] \times \frac{s}{B} = \frac{1}{L} \times \left[\int_0^L \left(cycleNo \times L + t(Q_i^{'}, x) \right) dx \right] \times \frac{s}{B} \\ &= T_{Access}(Q_i^{'}) + cycleNo \times L \times \frac{s}{B}. \end{aligned}$$

Q.E.D.

Proof of Lemma 2: According to the procedure QueryRefinement in Section 3.2, all data items of equal placements are hashed into the same bucket. Due to the loop in line 5-9, for each bucket in H, at most one data item will be selected and inserted into $Q_i^{'}$. Therefore, for two randomly selected data items from $Q_i^{'}$, say D_x and D_y, we have $placement(x) \neq placement(y)$. **Q.E.D.**

Proof of Lemma 3: Consider the proof of Lemma 2. Let the series $a(j)$, $1 \leq j \leq |Q_i^{'}|$ represent the *sorted* placements of all data items required by $Q_i^{'}$ in ascending order and b represent $|Q_i^{'}|$. Without loss of generality, we reorder all required data items of $Q_i^{'}$ to be $\{D_{k(1)}, D_{k(1)}, \cdots, D_{k(b)}\}$ according to the placements of all data items in ascending order. Note that the placement of $D_{k(j)}$ is $a(j)$. Due to the result of Lemma 2, all elements in the series $a(j)$ are distinct with one another.

As shown in Figure 18, the broadcast program can be divided into $b + 1$ parts by the start time of $D_{k(j)}$, $k = 1, 2, \cdots, b$. Consider the same scenario in the proof of Lemma 1. To minimize the access time, the mobile device will read $D_{k(p)}$ as the first retrieved data item when x lies on part p, $1 \leq p \leq b$. In addition, the mobile device will read $D_{k(1)}$ in $(m + 1)$-st broadcast cycle when x lies on part $(b + 1)$.

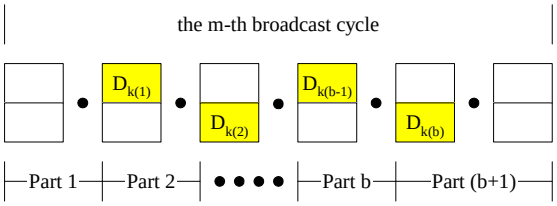

Figure 18: The illustration of the proof of Lemma 3

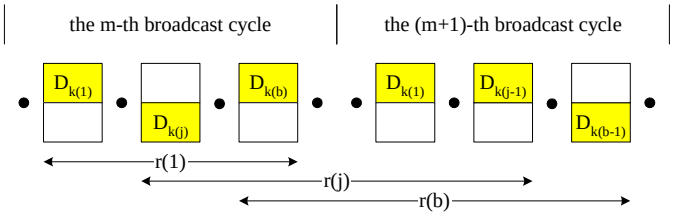

Figure 19: The illustration of the proof of Lemma 4

Recall that the time interval between the start time of the m-th broadcast cycle and the moment of the appearance of D_j is $(placement(j)-1)\times\frac{s}{B}=(a(j)-1))\times\frac{s}{B}$. Since x follows a uniform distribution over $(0, L)$, the average startup time of Q_i on the broadcast program can be formulated as below.

$$\begin{aligned} T_{Startup}(Q_i^{'}) &= \frac{1}{L}\times\left\{\int_{x=0}^{a(1)-1}\Big[a(1)-1-x\Big]dx+\int_{x=a(1)-1}^{a(2)-1}\Big[a(2)-1-x\Big]dx\right. \\ &\quad \left.+\cdots+\int_{x=a(b-1)-1}^{a(b)-1}\Big[a(b)-1-x\Big]dx+\int_{x=a(b)-1}^{L}\Big[L-x+a(1)-1\Big]dx\right\}\times\frac{s}{B} \\ &= \frac{s}{L\times B}\times\left\{\sum_{i=1}^{b-1}\frac{\Big[a(i+1)-a(i)\Big]^2}{2}+\frac{\Big[L-a(b)+a(1)\Big]^2}{2}\right\} \end{aligned}$$

Q.E.D.

Proof of Lemma 4: Consider the same scenario in the proof of Lemma 3. Let $p(j)$ represent the probability that the mobile device reads $D_{k(j)}$ as the first retrieved data item. For $2\le j\le b$, $p(j)$ is the probability that x lies on part j. In addition, $p(1)$ is the probability that x lies on part 1 and $b+1$. $p(j)$ can be formulated as $p(j)=\frac{L-a(b)+a(1)}{L}$ when $j=1$ and $p(j)=\frac{a(j)-a(j-1)}{L}$ when $j\neq 1$.

Now consider Figure 19. Since the placements of $D_{k(j)}$ $j=1,2,\cdots,b$ are distinct and in an ascending order, the mobile device will sequentially retrieve $D_{k(j)}, D_{k(j+1)},\cdots,D_{k(b)}$ in the m-th broadcast cycle and then retrieve $D_{k(1)}, D_{k(2)},\cdots,D_{k(j-1)}$ in the $(m+1)$-th broadcast cycle. Let $r(j)$ represent the retrieval time of $Q^{'}$ when the mobile device reads $D_{k(j)}$ as the first retrieved data item. $r(j)$ can be formulated as $r(j)=(a(b)-a(1)+1)\times\frac{s}{B}$ when $j=1$ and $r(j)=(L-a(j)+a(j-1)+1)\times\frac{s}{B}$ when $j\neq 1$. Then, $T_{Retr.}(Q^{'})$ can be formulated as $T_{Retr.}(Q^{'})=\sum_{j=1}^{b}p(j)\times r(j)$

$$=\frac{s}{B\times L}\Big\{(L-a(b)+a(1))\times(a(b)-a(1)+1)+\sum_{j=2}^{b}\Big[(a(j)-a(j-1))\times(L-a(j)+a(j-1)+1)\Big]\Big\}.$$

Q.E.D.