利用 A*演算法於無線環境之資料廣播問題

Using A* Algorithm for Data Broadcast in Wireless Environment

邱美倫研究生
Mai-Lun Chiu
Graduate Student
mellen10980@hotmail.com

吳光閔副教授
Guang-Ming Wu
Associate Professor
gmwu@mail.nhu.edu.tw

南華大學資訊管理研究所
Department of Information Management, NAN HUA University

ABSTRACT

Data Broadcasting is an efficient communication model when clients request data from a server in wireless environment. Data is delivered by a server downstream with a wide bandwidth. All clients keep listening to the broadcast channel and catch the data that interest them. The important issue of designing a proper broadcast schedule is to reduce the clients' *total access time*. Most previous researches focused on a query just only include one data item, but not consider multiple data items are included in a query. In this paper, we propose an A* algorithm to the broadcast problem which consider the complex queries where a query include multiple data items. Experiential results show that our method outperforms the QEM algorithm [14] in access time.

Keywords: Data Broadcasting, Total Access Time, Complex Queries

摘 要

在無線環境中，使用廣播方式傳送資料是一種有效率傳輸模式。在此環境下，伺服端連續且不斷的廣播資料，讓使用者擷取自己所需的資料。因此伺服端如何決定廣播頻道上的資料順序，讓使用者的總存取時間達到最小，是個很重要的議題。過去多數的研究只考慮到一筆查詢(query)只有包含一個資料項(單一資料項)，而忽略現實生活中使用者的要求多半會包含兩個以上的資料項(多重資料項)。故本研究針對無線廣播通道環境下，且使用者多重要求資料項時，提出A*演算法找出資料項的廣播排程，以降低使用者接收資料項所需的時間。由實驗結果證明我們所提出的方法較*QEM*演算法[14]好。

關鍵字：資料廣播、總存取時間、多重資料項

1 INTRODUCTION

In this chapter, we describe the wireless environment of data broadcasting, some problems are discussed and instruction our approach. At last, we list our research framework.

Advance in science and technology, internet and intranet have enabled the development of data-dissemination applications. The mobile computing and the communication technology in wireless are expanded fast in recent years. There are more and more people to utilize the public infrastructures to deliver information to other mobile users who are interested in the information.

Wireless network architectures can be divided into Ad-Hoc and Client-Sever. An ad-hoc network forms a temporary network which consists of mobile devices without pre-established infrastructure [23]. In Ad-Hoc network, each mobile device can be a server to send information to its neighborhood or just be a client to receive the data items, and each mobile device could move free in its communication range. The power control problem of portable devices makes mobile users communicate only within their transmission ranges [28]. In [10] had proposed a broadcast tree method with shorting the longest edge among a spanning tree to save power consuming. There is a research using neighbor caching strategy to put the data items which it will request in its neighborhood for sharing their cache capability, it is an algorithm that can adjust neighbor caching ability and makes all caches flexible according to their idleness of storage [11]. The research in [10], proposed a forwarding set selection scheme to broadcast with transmission power control in two-hop ad hoc network. The two-hop local information included node ID and node’s signal strength which was used to calculate the transmission power.

The wireless communication in client-sever includes a server and many clients. Each mobile client could access data items which they interest in pass the server. The communication capacity from a server to clients (*downstream*) is far greater than clients to a server (*upstream*) in the wireless environment. For example, a server has a high bandwidth broadcast capacity if clients can not sent data with lower bandwidth. That means in the wireless environment, the mobile users are limited in bandwidth and power consuming. Because of this reason, the

mobile

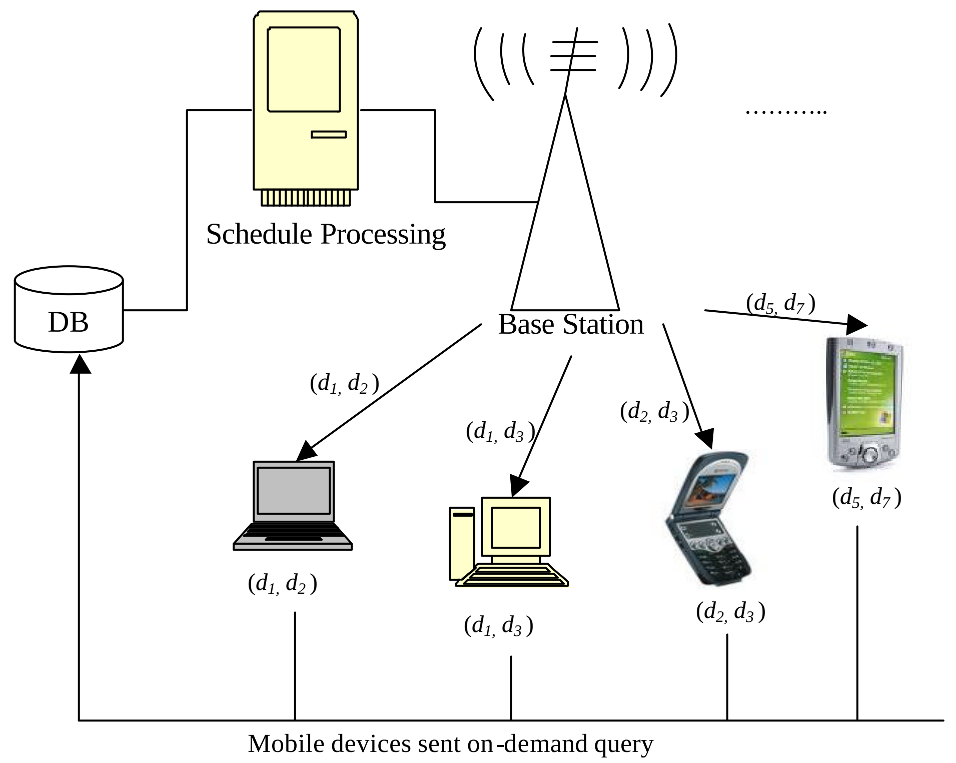


Figure 1- 0：The environment of wireless broadcast for pull-based system.

users just care how long they will receive the data completely which they want to use and how to reduce the power consuming. Therefore, information systems taking *broadcast-based* are proposed in succession. Acharya et al. [1, 2, 5, 34] proposed *Broadcast Disk* for structuring the broadcast way. The client terminals take over the information through the broadcast system. The server analyzes the data items access patterns of all queries of clients and broadcasts the data items in turn.

In wireless broadcast environment, the server will sent all data items repeatedly and continuously. Such the systems can be categorized into two ways to broadcast data items： (1) *pull-based* approach [7, 20, 31, 33, 39, 44]：It considers whether clients sent data queries to the server. A server only broadcasts data items on demand as mobile devices ask them explicitly, so unwanted data items will never be broadcast (shown in Figure 1-1). (2) *push-based* approach [6, 12, 17, 18, 35, 38, 45, 47]：A server broadcasts data items repeatedly and mobile devices listen to the broadcast channel and receive the data items needed. The benefit of this way is the scalability. The broadcast scheduling will be given an indication of data items desired by all clients and the cost for delivering data items is independent of queries. In other words, a push-based broadcast can satisfy multiple queries with the same data items. While the mobile devices entry the broadcast channel and sent their queries, they will listen to the information until they receive all data items which they interested in. We show the environment of wireless broadcast for push-based system in Figure 1-2.

There are two important issues when we discuss wireless data broadcasting which are shorten *tuning time* and reduce *access time*. The tuning time is the amount of time spent by a client listening to the channel [14]. The tuning time is determined by power consuming while mobile

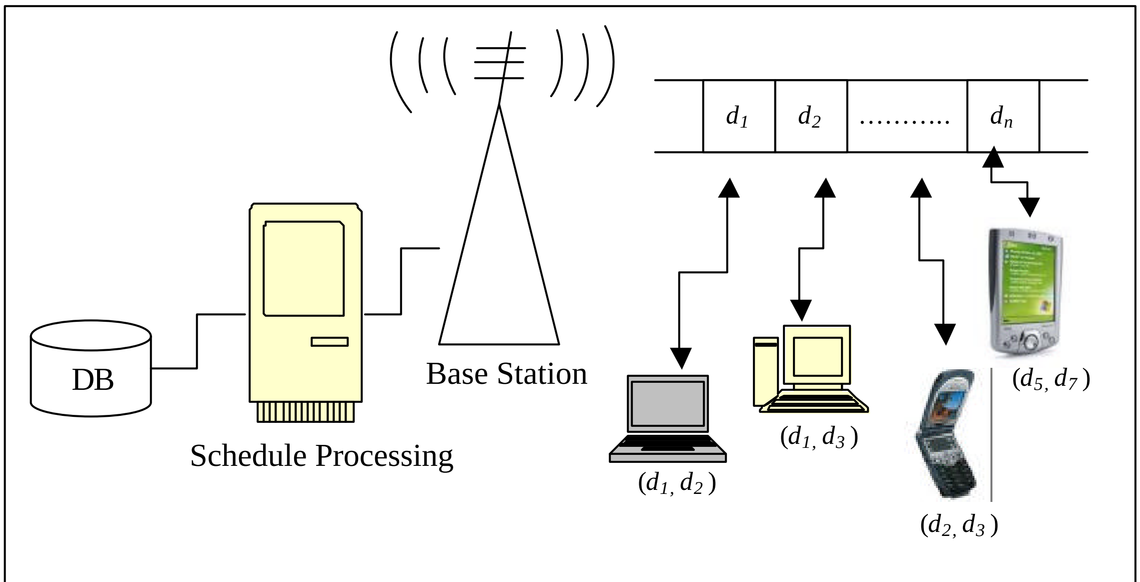


Figure 1- 0：The environment of wireless broadcast for push-based system.

devices receive data items [13]. The *access time* is the amount of time elapse from the moment a client submits a query to the receipt of the data items of his interest on the broadcast channel [14].

Mobile devices will operate in two modes. One is called *active mode*, while the mobile devices connect the broadcast channel and examine the information from the server to decide if they should receive the data items. In this mode, CPU is operated for investigating the information whether match what they need and it will consume amount of battery power. Another mode means the mobile devices are worked in the *doze mode* to save power consuming as their demanded data objects arrive yet.

For energy saving, there were researches proposed by using index techniques (shown in Figure 1-3) to access data objects on the broadcast channel [13, 23, 24, 32, 43]. Index based organization of data transmitted over broadcast channel, is very important form the power conservation point of view and can result in significant improvement in battery utilization [25]. They added some information in front of all data items and all clients can accord to the addition information to access data objects without listening in the channel continuously, all mobile clients can

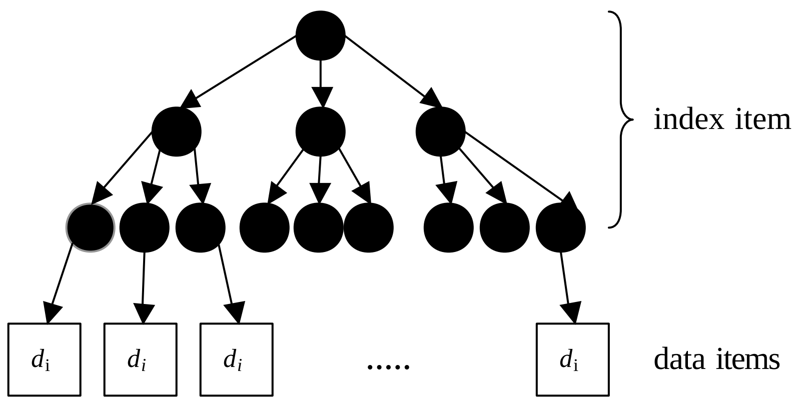


Figure 1- 0： The approach of index tree.

be directed to take over the data items efficiently. In [22, 24, 25], the index data are broadcast *m* times for each broadcast cycle, called (*1,m*). Distributed indexing improves (*1,m*) indexing algorithm by decreasing some partial replication of index. Some researches [42, 43], introduced taxonomy of index dissemination for broadcast channels. They utilize B^+ tree to construct search model.

In [36] devised an algorithm, referred to as algorithm DL, to dynamically adjust the broadcast programs by shuffling data items among different levels in the allocation tree. In [23], proposed two policies to reduce the tuning time. The lower power level index first policy tended to cache the leave index nodes of the index tree while the cut plane first policy cached the cut-plane of index tree. In [37], proposed a novel *on-demand* method, named *NICD* (Normalized Inter Cluster Distance), which eliminates the need for indexing the broadcast schedule by enabling mobile devices compute the require index information themselves. In [17], proposed an on-line algorithm to disseminate events update. It assumed that each channel has the fixed time slots and used the concept of TDM (Time Division Multiplexing) to disseminate data items. Mobile devices monitored the channel with the same interval time to save energy consuming and avoid missing update data items. In [43], presented a global indexing scheme for location dependent queries, which was designed to serve queries in which the query result is relevant to client' s location.

Many schemes proposed to broadcast data items efficiently to a large number of mobile devices. They tried to minimize the total access time for the data items needed. Some algorithms consider the property of *real-time* data items and *non-real-time* data items. In the real-time system [8, 27, 31, 33], the data items must be transferred to clients within the deadline. In [27] introduced the concept of absolute validity interval (AVI) to capture the temporal constraint of the data items. It was applied in many applications such as stock trading system, traffic system…, and so on. For example, the stock price changes at any time, and if the users can not receive correct price information, they will not handle the stocks on time. In intelligent vehicle highway system (IVHS) [31], sent present traffic information to drivers on time. If the information is not sent to the drivers on time, it will be useless information.

In non-real-time system, many broadcasting schedules are studied to reduce the waiting time of clients for asked data items on the air. For transmitting data items efficiently, we must look for suitable broadcasting schedule of a set of data objects. Some researches were proposed in [14, 30], which utilize the characteristic of data frequently to decide the scheme. So the more popular data items must be broadcasted many times or be placed in front of the not popular data items in the same cycle. The schedule methods [12, 19, 29, 45, 46], established the broadcast schedule by using *caching strategy* which put the hot data in local host. The advantage of this way is decreasing the times for clients to ask their desired data items pass a server. However, it is limited in the cache size of capacity with each mobile user. In [18], proposed the method is called First Come First Served (*FCFS*), which ordered the data items by their request time. The advantage is the access request will get responded in a finite time. But it does not consider the difference of access frequency with data items. In the later, Most Request First (*MRF*) scheduling method broadcasts the data items which bases on the largest number of request was proposed. If most-frequent data items in a broadcast cycle, they will have higher response ratio. But its shortcoming is the lower-frequency data items will always put behind the most-frequent data items. So the request on those will not be satisfied in a short period. In addition, [4] combined the benefits of MRF and FCFS in order to provide good performance for both hot and cold data items. It considers the data items of access frequency and waiting time to calculate the proper data scheduling, declared as $R \times W$ method. In [40, 47], proposed a non-greedy, low polynomial time cost optimization method to place data over a wireless broadcast channel for multi-dimensional range query processing.

Previous works have focused on retrieving a single data item from a broadcast channel. But in real word, mobile devices may access multiple data items. Few works [15, 19, 27], had been done on complex queries where a query includes multiple data items. In [15], addressed the clustering of data items for multipoint requests, that was, a query access more than one data item recorded on the broadcast stream. It defined two affinity measures：data affinity and segment affinity.

The method clustered data items based on the two measures.

Some researches [35, 38], using data mining techniques decide a broadcast scheme. They based on analyzing the broadcast history (i.e., the chronological sequence of data items that had been requested by mobile devices) to find associations and sequences in individual data items. In [14], they construct the data scheduling by appending the data items of each query to minimal total access time with greedy method. It considers the frequency of each query to find the relationship between all data items. If the data items have high relationship, they will be put on together. But in this way, they just only account of the data items in one query, and ignore the situation of all data items which were accessed in whole queries.

In this paper, our goal is to find a good broadcast scheduling that can be reduced clients' access times. Our system environment is assumed as follows：

- The server broadcast data items on a push-based system.
- There is only one broadcast channel.
- A query can be included multiple data items.
- The size of data items is equally.
- A data item will be disseminated once in the same broadcast cycle.

The data placement problem can be formulated as a path search problem, hence we propose an A^* algorithm which is a graph search algorithm to decide a broadcast scheduling. A^* algorithm is quite famous in artificial intelligence domain. In the A^* algorithm, we consider the requested relationships among data items in whole queries. We design a cost function and combine a heuristic Breadth-First-Search algorithm to find a good solution. In order to arrive at the aim in the reasonable time, we also design a window size to make the data items within the range are performed A^* algorithm for searching optimal data placement in a window size. Experiential results show that our method outperforms the *QEM* algorithm in access time.

The rest of this paper is organized as follows. Chapter 2, we define the problem of the data broadcasting problem in the wireless environment and address some assumption conditions. The A^* algorithm for data allocation with some illustrative examples is proposed in Chapter 3. Performance study results are discussed in Chapter 4. Final, conclusions are given in Chapter 5.

2 BROADCAST SCHEDULING PROBLEM

In this chapter, we define the data placement problem and introduce how the issue is produced. This problem was showed that it is a NP-complete problem [14]. We will use *QD* method [14] to measure the total access time of a query which is a mobile client needed.

2.1 Symbol definitions

Table 2-1 shows some notations for problem definition [14]. A server will place the data items on the broadcast channel to minimize the *total access time* (*TAT*), denoted by [14]：

Table2- 1：Symbol definitions [14]

Notation	Meaning
d_i	a data item to be broadcast
D	a set of data items d_i;{ d_1, d_2 ,..., d_n }
B	the size of a broadcast stream i.e.,$\sum \lvert d_i \rvert, \forall d_i \in D$
q_i	a query that is issued on broadcast data stream
$QDS(q_i)$	the set of data items that q_i accesses
$ferq(q_i)$	the frequency of q_i
Q	the set of queries of;{ $q_1, q_2, ..., q_M$ }
s	the broadcast schedule of D

$$TAT(\sigma)=\sum_{q_i \in Q} AT^{avg}(q_i,\sigma)\times freq(q_i), \quad (1)$$

where $\underset{q_i \in Q}{AT}^{avg}(q_i, s)$ is the average access time of a query q_i based on s. Because clients tune into broadcast channel on different time, $\underset{q_i \in Q}{AT}^{avg}(q_i, s)$ is hard to calculate. However, a measure manner *Query Distance* (*QD*) is proposed to evaluate $\underset{q_i \in Q}{AT}^{avg}(q_i, s)$, that indicates the degree of coherence of the data items in a query [14]. The measure is interpreted as follows：

Definition 1 [14]：

Suppose $QDS(q_i)$ *is* $\{d_1, d_2, \ldots, d_n\}$ *and* d_j *is the interval between* d_j *and* d_{j+1} *in a schedule* s. *Then the QD of* q_i *in* s *is defined as* ：

$$QD(q_i, \sigma) = B - MAX(\delta_k), \quad k = 1 \sim n$$

The example in Figure 2-1, we assume $s = < d_1, d_2, d_3, d_4, d_5 >$, the B is equal to 5 and the $QDS(q_i) = \{d_2, d_4\}$. Hence the d_1 is equal to 1 and d_2 is equal to 2, and $QD(q_i, \sigma) = B - MAX(\delta_k) = 5 - 2 = 3$.

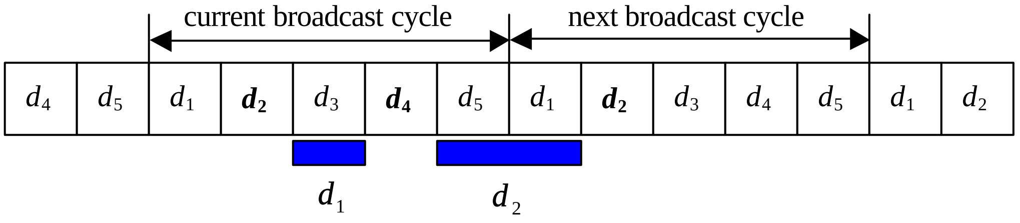

current broadcast cycle
next broadcast cycle
d_4 d_5 d_1 d_2 d_3 d_4 d_5 d_1 d_2 d_3 d_4 d_5 d_1 d_2
d_1
d_2

Figure 2- 0：The *QD* of a query.

In [14] proposed the lemma as follow：

Given a query q_i *and two schedule* s_1 *and* s_2

if $QD(q_i, \sigma_1) \geq QD(q_i, \sigma_2)$
then $AT^{avg}(q_i, \sigma_1) \geq AT^{avg}(q_i, \sigma_2)$

The total query distance is presented as $TQD(s)$ that is defined as $QD(q_i, s) * freq(q_i)$, where $q_i \in Q$. The broadcast scheduling problem redefine to minimize the $TQD(s)$.

The definition was proposed in [14]：

Given a set of queries Q and a set of data items D, the wireless data placement problem is to find a broadcast schedule s_i *such that*

TQD(s_i*) is minimum among all possible* s_i, $i=1,\ldots$

2.2 Effect of different broadcast schedule

A broadcast scheduling of a server determines an ordering of data items which through the server. We use s as a broadcast cycle which presents a data ordering. In this paper, we assume that there is a server and some clients in the wireless environment. Server will analyze the clients' request patterns to find a data schedule. The sizes of data items are equally and they will be broadcasted once in the same cycle. Besides, we allow each query can consist of more than one data items. A data item is denoted as d_i in this paper.

An ordering of data items affects the access time of all clients directly [14]. For example, in Figure 2-2 (a), we assume the broadcast cycle $s = <d_1, d_2, d_3, d_4, d_5>$. There is a client ($C_i$) which requests the data items d_3 and d_5 ($q_i = \{d_3, d_5\}$). C_i listens to the broadcast channel when the server broadcasts d_3 in part. In order to access d_3 completely, it will wait for next broadcast cycle, but d_5 will be received in current broadcast cycle. In other words, the client must wait for d_3 until next broadcast cycle to access d_3 and d_5 completely. We present the AT (q_i) as the time from a client tunes in a broadcast channel until it receives all data items which are the client wanted. So while the broadcast cycle $s = <d_1, d_2, d_3, d_4, d_5>$, the AT (q_i) is equal to 5.5 (Figure 2-2(a)). But if the broadcast cycle is changed as $s' = <d_4, d_1, d_2, d_3, d_5>$, the AT (q_i) will be 2.5 (Figure 2-2(b)).

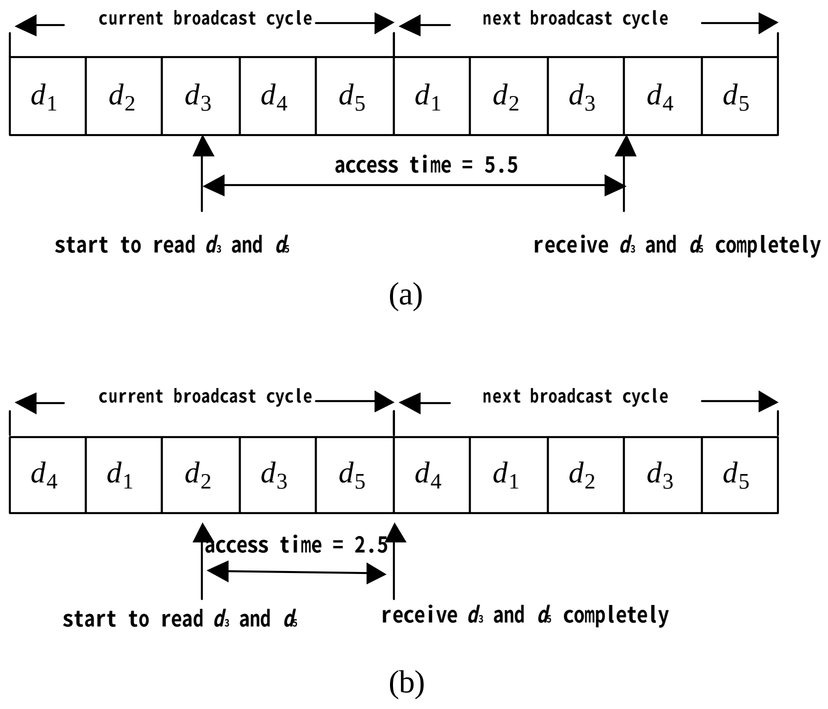

current broadcast cycle
next broadcast cycle
d_1 d_2 d_3 d_4 d_5 d_1 d_2 d_3 d_4 d_5
access time = 5.5
start to read d_3 and d_5
receive d_3 and d_5 completely
(a)
current broadcast cycle
next broadcast cycle
d_4 d_1 d_2 d_3 d_5 d_4 d_1 d_2 d_3 d_5
access time = 2.5
start to read d_3 and d_5
receive d_3 and d_5 completely
(b)

Figure 2- 0：The example of the *Access Time*.

However, in real world, there are many clients to request different queries. If we change the broadcast cycle, it maybe increases the access time of some clients. For example, we assume two clients sent their queries, $q_1 = \{d_1, d_2\}$ and $q_2 = \{d_2, d_3\}$. If $s = <d_1, d_2, d_3, d_4, d_5>$, the $AT(q_1) = 2$ and $AT(q_2) = 3$ (shown in Figure 2-3(a)). If $s' = <d_2, d_3, d_5, d_1, d_4>$, then $AT(q_1)$ will be increase from 2 to 4, and $AT(q_2)$ will be decrease from 3 to 2 (shown in Figure 2-3(b)). In this case, we know the benefit among queries is a complex and hard work, and all data items have different frequency with accessing times. Our purpose is to decide a data schedule to make the *Total Access Time* as smaller as possible.

3 A* search algorithm Approach to Wireless Data Placement

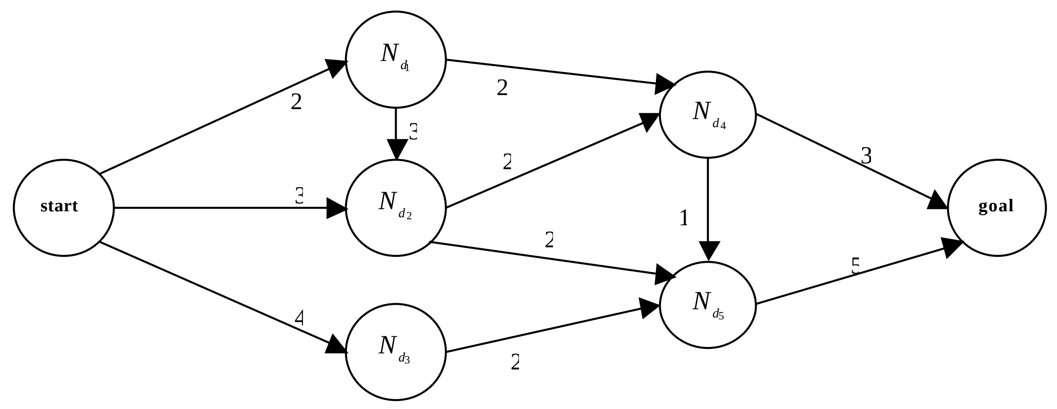


Figure 3- 0：A state space graph.

3.1 Basic idea

A* algorithm [16, 41] is a graph search algorithm that finds a path from a given initial node to a given goal node. It utilizes a "heuristic estimate" that orders each node by estimating the best route that goes through that node. It visits the nodes in order of this *heuristic* estimate. The A* algorithm is therefore an example of best-first search. The Best-First-Search (BFS) algorithm [41] uses *heuristic* function to estimate how far from the goal. Instead of choosing the node closest to the start point, it selects the node closest to the goal node. Because of using a heuristic function guides the way towards the goal node very quickly.

Let we consider the follow example. If we are standing at place X, and we want to go to place Y. The X place is a node of the graph and a road is an edge. The data placement problem can be formulated as a path search problem in an acyclic directed graph, called state space graph (shown in Figure 3-1). If we do a breadth-first search which is like Dijkstra's Algorithm [41], we will search all nodes within a state space graph, gradually expanding paths to search places farther and farther away from our starting node. However, a better strategy is to explore the node directly to the goal node first. Then, the roads permitting, we will continue to explore intersections closer and closer to the goal.

3.2 Description

A^* algorithm begins at a selected node. Applied to this node is the "cost" of this node (usually zero for the initial node). A^* algorithm estimates the cost to the goal node from the current node. The cost is assigned to the path leading to this node. Then, the node is added to a priority queue, usually denoted as "*open*". The algorithm after removes the next node from the priority queue. If the queue is empty, there is no path from the initial node to the goal node and the algorithm can be stopped. If the node is the goal node, A^* algorithm will output the successful path.

If the node is not the goal node, new nodes are created for other admissible adjoining nodes. For any successive node, A^* algorithm calculates the "cost" of the node and saves it with the node. This cost is calculated from the cumulative sum of costs which are stored with their ancestors, plus the cost of the process which reached this new node.

The algorithm maintains a "*closed*" list of nodes which have been checked. If a generated node newly has been located in this list with an equal or lower cost, no further processing is done on that node. If a node in the *closed* list mates a new node, but had been stored with a *higher* cost, it is removed from the closed list, and processing continues on the new node. Next, an estimate of the new node's cost to the goal is increased to the cost with forming the heuristic for that node. Then it is added to the "open" priority queue, unless an identical node with lesser or equal heuristic is found there.

As soon as the above steps have been repeated for each new adjoining node, the original node taken from the priority queue is added to the "closed" list. The next node is then popped from the priority queue and the process is repeated. The A^* algorithm procedure is a branch and bound search algorithm, with an estimate of remaining path, which is combined with the dynamical programming principle. It is also an important work to design a proper cost function in a heuristic search algorithm. An estimate of the new node's cost directly affects the final solution. If the estimate of remaining path forever is a lower-bound on the actual path, it is the optimal solution. We will describe our cost function in 3.3.1. To conduct A* algorithm search below [16]：

- ✓ From a one-element queue consisting of a zero-length path that contains only the root node.
- ✓ Until the first path in the queue terminates at the goal node or the queue is empty,
  - ■ Remove the first path from the queue; create new paths by extending the first path to all the neighbor of the terminal node.
  - ■ Reject all new paths with loops.
  - ■ If two or more paths reach a common node, delete all those paths except the one that reaches the common node with the minimum cost.
  - ■ Sort the entire queue by the sum of the path length and a lower-bound estimate of the cost remaining, with least-cost paths in front.
- ✓ If the goal node is found, announce success; otherwise, announce failure.

3.3 Using A* algorithm for data broadcast

Using A* to decide a path with minimum costs in a state space graph is effective [5, 9]. It is a branch and bound algorithm that starts at a vertex and branches at the vertex i with the lowest cost that has been visited up now. Note that only the visited nodes are created dynamically.

A correct estimate will cause only expansions on the optimum path. Moreover the search is accelerated by the use of a monotonically increasing cost function, because not any vertex will be expanded twice. Next, we introduce our cost function.

3.3.1 Cost function

There are different results with different cost functions, so how to estimate the cost of each node is an important work. In this session, we illustrate our cost function with a simple example.

Assume there is a set of data items to be placed, denoted as D= $\{d_1, d_2,\ldots,d_n\}$. A query q_k accesses a set of data items is represented as $QDS(q_k)$. We introduce the relationship of data items for the queries q_k in Figure 3-2. A vertex is denoted as V_i included a set of data items that has been placed (shown in Figure 3-3). The V_i's parent node is $P(V_i)$. A cost of the vertex V_i is denoted as $C(V_i)$ and $C(P(V_i))$ is the cost of its parent node. The

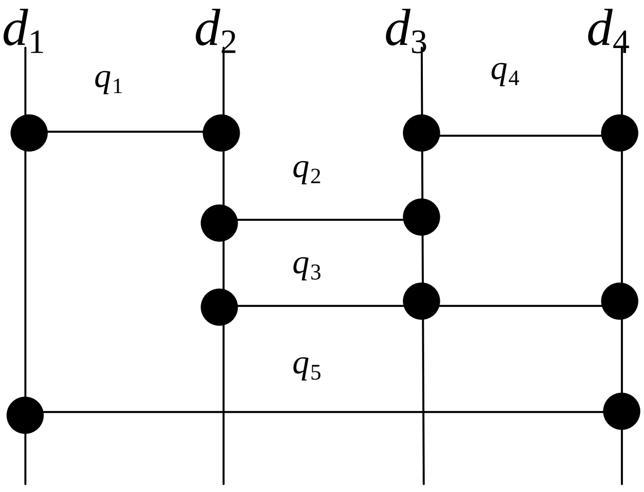


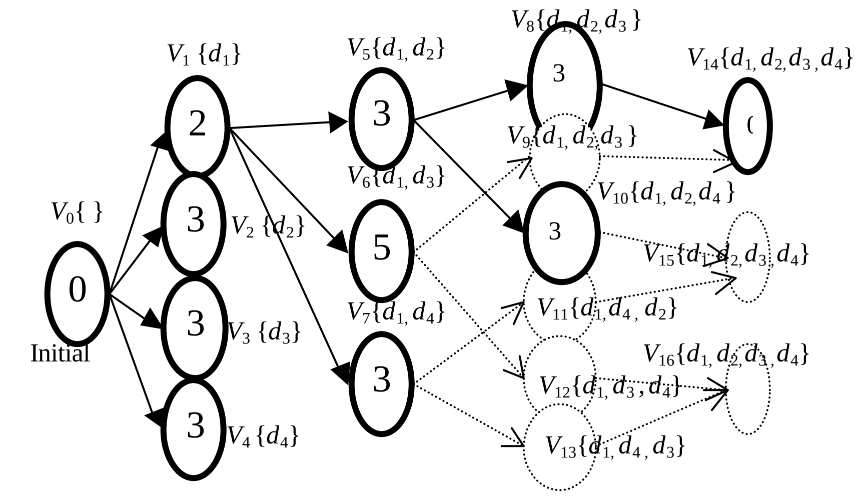
Figure 3- 0：The relationships of data items with different queries.

$M(V_i) \subset D$ means a set of data items which has been placed and $\overline{M}(V_i) \subset D$ is a set of data items which not have been placed. The $\hat{C}(V_i)$ is the number of queries that links $M(V_i)$ and $\overline{M}(V_i)$. If there are n queries between the two set, $\hat{C}(V_i)$ is equal to n. The number within a vertex in Figure 3-3 is $\hat{C}(V_i)$. We illustrate how to order the data items with a simple case.

For example, if there are five queries $QDS(q_1) = \{d_1, d_2\}$、 $QDS(q_2) = \{d_2, d_3\}$、 $QDS(q_3) = \{d_2, d_3, d_4\}$、 $QDS(q_4) = \{d_3, d_4\}$、 $QDS(q_5) = \{d_1, d_4\}$, the data items' relationships are shown in Figure 3-3. The cost of vertex V_1、V_2、V_3、V_4 are equal to 2、3、3、3, denoted as $\hat{C}(V_1)=2$、$\hat{C}(V_2)=3$、$\hat{C}(V_3)=3$、$\hat{C}(V_4)=3$, the computational processes are presented in Figure 3-4. In other words, if exists two data items, d_i and d_j belonged to $QDS(q_k)$, and $d_i \in M(V_i)$、$d_j \in \overline{M}(V_i)$, then we add 1 to

$\hat{C}(V_i)$. Thus the cost function in our research is presented as follows：

$$C(V_i) = C(P(V_i)) + \hat{C}(V_i) \qquad (2)$$



$V_0\{\ \}$
0
Initial
$V_1\ \{d_1\}$
2
$V_2\ \{d_2\}$
3
$V_3\ \{d_3\}$
3
$V_4\ \{d_4\}$
3
$V_5\{d_1, d_2\}$
3
$V_6\{d_1, d_3\}$
5
$V_7\{d_1, d_4\}$
3
$V_8\{d_1, d_2, d_3\}$
3
$V_9\{d_1, d_2, d_3\}$
$V_{10}\{d_1, d_2, d_4\}$
3
$V_{11}\{d_1, d_4, d_2\}$
$V_{12}\{d_1, d_3, d_4\}$
$V_{13}\{d_1, d_4, d_3\}$
$V_{14}\{d_1, d_2, d_3, d_4\}$
$V_{15}\{d_1, d_2, d_3, d_4\}$
$V_{16}\{d_1, d_2, d_3, d_4\}$

Figure 3- 0：State space graph for the example in Figure 3-2. Sub graph searched during Branch and Bound (solid), and optimal path (bold).

Our goal is to find a data schedule that let $TAT(s)$ is smaller as possible. According to the cost function, we calculate the vertex as follows to find an optimum solution. Figure 3-3 is a state space graph for the example in Figure 3-2. The calculation of a node of V_i's $C(V_i)$ are explained as follows：

$\hat{C}(V_1) = 2$、$C(P(V_1)) = 0 = C(V_0)$ so

$C(V_1) = C(P(V_1)) + \hat{C}(V_1) = 0 + 2 = 2$

$\hat{C}(V_2) = 3$、$C(P(V_2)) = 0 = C(V_0)$ so

$C(V_2) = C(P(V_2)) + \hat{C}(V_2) = 0 + 3 = 3$

$\hat{C}(V_3) = 3$、$C(P(V_3)) = 0 = C(V_0)$ so

$C(V_3) = C(P(V_3)) + \hat{C}(V_3) = 0 + 3 = 3$

$\hat{C}(V_4) = 3$、$C(P(V_4)) = 0 = C(V_0)$ so

$C(V_4) = C(P(V_4)) + \hat{C}(V_4) = 0 + 3 = 3$

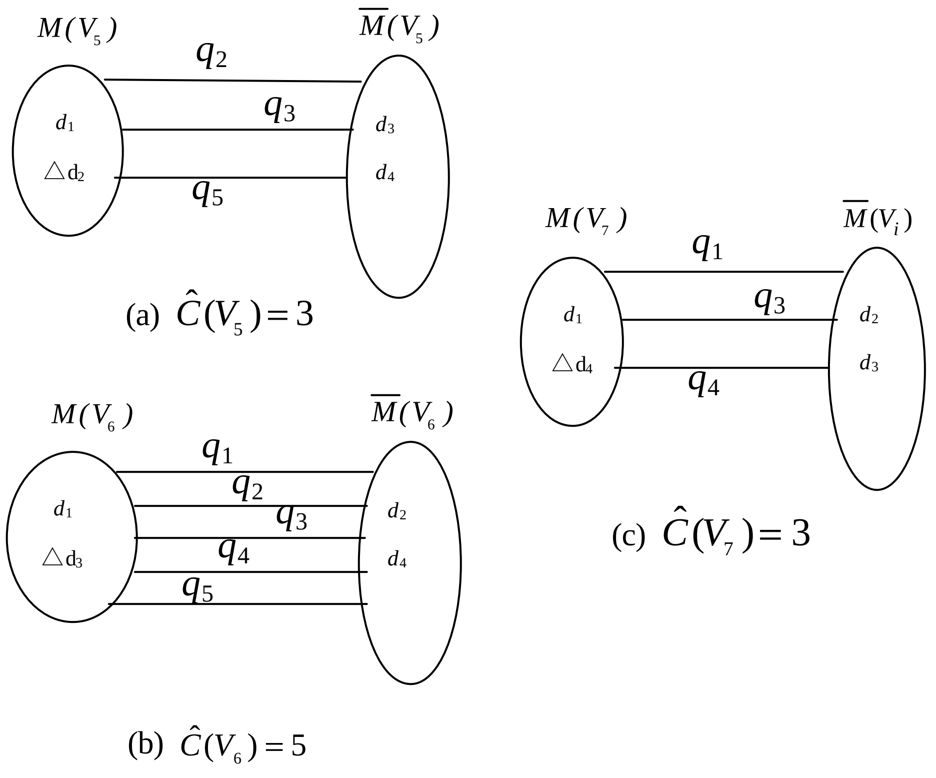


Figure 3- 0：The calculation of a node of V_i's $\hat{C}(V_i)$.

We choose the minimum cost of node ($C(V_1)$) to expand. Now, d_1 presents the data item that has been located ($M(V_1) = d_1$), and the set of data items, $d_{2,}$ $d_{3,}$ d_4, presents those *not* have been located ($\overline{M}(V_1) = d_{2,}d_{3,}d_4$). A* algorithm always choices the minimum cost of V_i to expand. We calculate the cost of data sequence of $d_1 \rightarrow d_2$ ($C(V_5)$)、$d_1 \rightarrow d_3$ ($C(V_6)$) and $d_1 \rightarrow d_4$ ($C(V_7)$),the result is presented in Figure 3-5, then it expands V_5 (Figure 3-6) and V_8 (Figure 3-7). In the instance, the optimal path is $d_1 \rightarrow d_2 \rightarrow d_3 \rightarrow d_4$, and the cost is equal to 8.

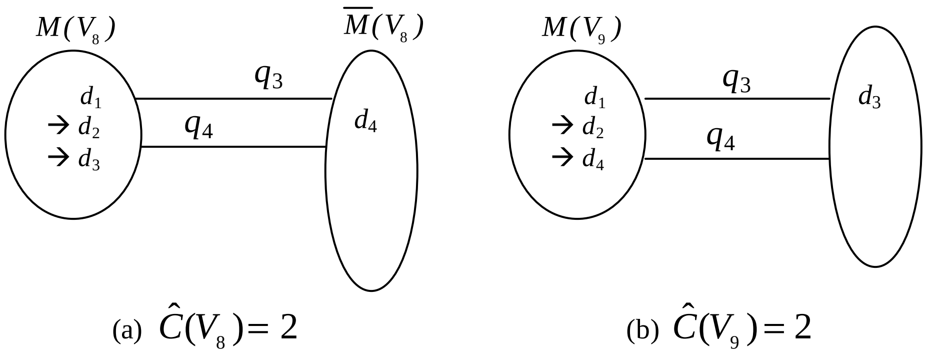


Figure 3- 6：Expand V_5 to calculate the V_i's $\hat{C}(V_i)$.

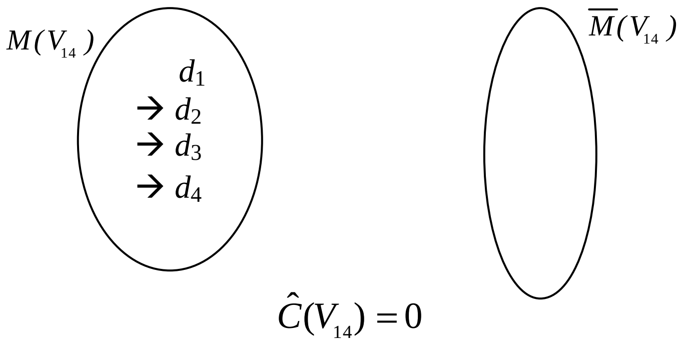


Figure 3- 7：Expand V_8 to calculate the V_i's $\hat{C}(V_i)$

3.3.2 Our algorithm

```
Algorithm：A$^*$ Algorithm ($s'$).
Input：$s'$ - the ordering of data items in the window.
Output：An optimal data schedule($s''$).
1. initial root $r$;
2. $Q \leftarrow f$; /* $Q$=Queue.
3. $M(N_{d_r})=\{f\}$; $C(N_{d_r})=0$;
4. add all $N_{d_r}$ into Q;
5. each time delete $N_{d_i}$ from Queue with Min $C(N_{d_i})$;
6. for each $d_j$ in $s'$
7.     if ($d_j \notin M(N_{d_i})$) then
8.         new node $N_{d_j}$;
9.         if $N_{d_j}$ exist data items belonged to $QDS(q_k)$)
10.            if some data items in $q_k \in M(N_{d_j})$ and some data items in $q_k \in \overline{M}(N_{d_j})$ then
11.                $\hat{C}(N_{d_j}) = \hat{C}(N_{d_j}) + 1$;
12.                $M(N_{d_j}) = M(N_{d_i}) \cup \{d_j\}$;
13.                $C(N_{d_j}) = C(P(N_{d_j})) + \hat{C}(N_{d_j})$;
14.        T=true;
15.        for each $N_{d_k}$ in $Q$;
16.            if ($M(N_{d_k}) \supseteq M(N_{d_j})$ and $C(N_{d_k}) \le C(N_{d_j})$) then
17.                T=false;
18.                abort the for loop;
19.            if ($M(N_{d_k}) \subseteq M(N_{d_j})$ and $C(N_{d_k}) \ge C(N_{d_j})$) then
20.                delete $N_{d_k}$ from $Q$;
21.    if (T) then
```

First, we initial the root r (N_{d_r}), and let the queue, denoted as Q, is empty. The root node is not placed any data item yet, that is $M(N_{d_r})=f$, and its cost is equal to 0, presented as $C(N_{d_r})=0$. And then N_{d_r} is added into the Q.

Line 5, removes the node N_{d_i} with minimal cost from the Q. Lines 6 to 13 explain how to create nodes according to $M(N_{d_i})$. Lines 9 to 13 calculate the cost of a new N_{d_j} (the blow-by-blow step is shown in section 3.2.2), and we copy the parent node d_i's $M(N_{d_i})$ into $M(N_{d_j})$ and then append d_j into $M(N_{d_j})$.

Lines 15 to 20 adjudge whether have N_{d_i} must be deleted from the Q or N_{d_j} has not added into the Q. Lines 16 to 18 show if $M(N_{d_i})\supseteq M(N_{d_j})$ and the cost of N_{d_i} is less than the cost of N_{d_j} ($C(N_{d_i})\le C(N_{d_j})$). Then we will reject N_{d_j} into the Q. For example, we assume a node N_{d_i} in the Q and its $M(N_{d_i})=\{d_{1,}d_{2,}d_3\}$ and its cost is 5. The new node N_{d_j} and its $M(N_{d_j})=\{d_{3,}d_2\}$ and its cost is 8. We will reject N_{d_j} into the Q. Lines 19 to 20 explain how to delete the nodes that can not find optimal solution.

If $M(N_{d_i})\subseteq M(N_{d_j})$ and its cost is greater than the cost of N_{d_j} ($C(N_{d_i})\ge C(N_{d_j})$). We will delete N_{d_i} from the Q and then add N_{d_j} into the Q. For example, if a node N_{d_i} is in the Q, and its $M(N_{d_i})=\{d_{1,}d_{3,}d_2\}$ and its cost is 9. The new node N_{d_j} has $M(N_{d_j})=\{d_{1,}d_{2,}d_3\}$) and its cost is 5. We will delete N_{d_i} from the Q and then add N_{d_j} into the Q.

3.3.3 Set a range to implement A* algorithm

When a server collects clients' query patterns, it will produce a broadcast schedule (shown in Figure 3-8). As description in section 3.1, A* algorithm uses the branch and bound method to

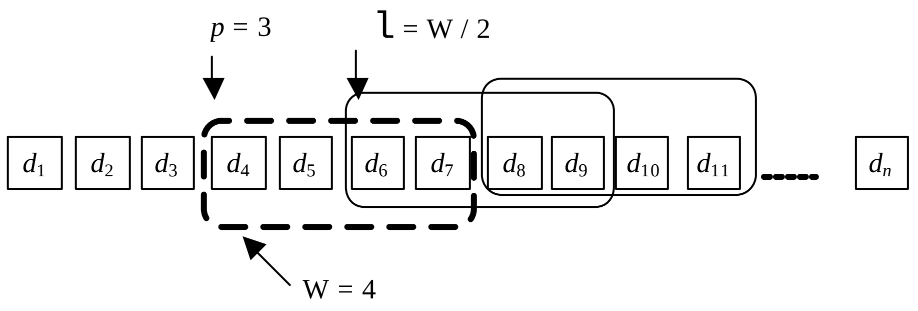


Figure 3- 0：Set a window size.

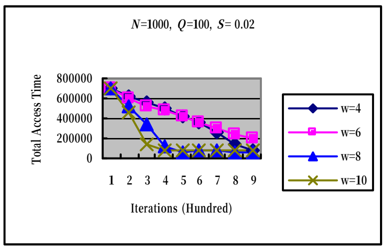
reach its work. So we can predict that if the number of data items becomes greater, it also expends more time to estimate all possible paths in a state space. For this reason, we set a search range called *window size*, denoted as *W* (shown in Figure 3-8), and let the data items in the scope are implemented by A^* algorithm until the window size includes the last node. Notice that in order to cover data items in front *W*, the shift scope, denoted as l is between 1 to W. Therefore A^* algorithm gets a broadcast schedule in a reasonable time. In order to move data items in a proper position, we reset continuously the start point of the window denoted as *offset*, *p*. we determine the *p value* between 0 to *W* in a random way. On the left of *p* is the first group to perform A^* and the other data items are enforced A^* which are according to the window size. We summarize our method as follows：

```
1.    Input：W, l ;   /* W is the range which
      includes data items to run A*.
2.    /* l  is the window shift scope.
3.    initial a data schedule;
4.    repeat
5.          random choose a number p;
                /* 0 <= p <W.
```

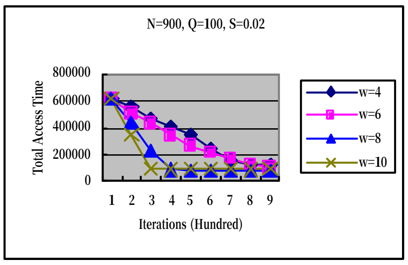
```
6.  set the first window covers the first $p$
      data items and use A* algorithm
7.          to schedule the data items;
8.        repeat
9.        shift right the window by $l$ ;
             /* 0 <= $l$ <$W$.
10.       call A* ($s$ ') algorithm to schedule
             the data items in the window;
11.          /*$s$ ' is the ordering of data
             items in the window.
12.       until the window includes the last
            data item;
13. until the cost converge;
```

4 Performance Evaluation

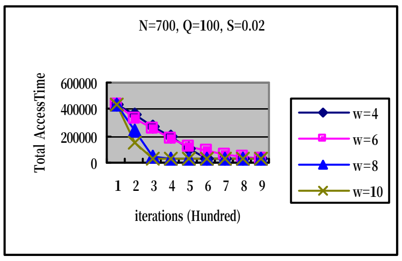
In this chapter, we evaluate the performance of the A* algorithm and compare our method with the QEM algorithm [14]. We implement the algorithm A* and QEM using Java language. The performance metric is considered in experiments with the total access time of all queries. (Please see Chapter 2). We run these programs on a PC with P4 2.0 GHz micro-processor, 256MB RAM and 30GB hard disk. In our experiments, the selectivity is denoted as S. Each query can include S data items of N (the number of data items) at most. For example, if N = 100 and S = 2%, each query can access 100*2% data items at most. The query patterns' access frequencies are with two distributions：(1) Normal distribution (2) Uniform distribution.



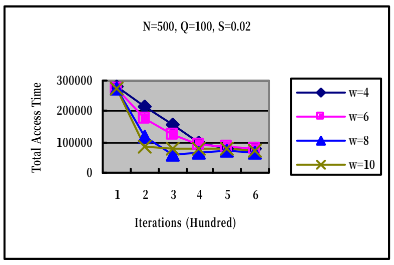
(a) N=1000



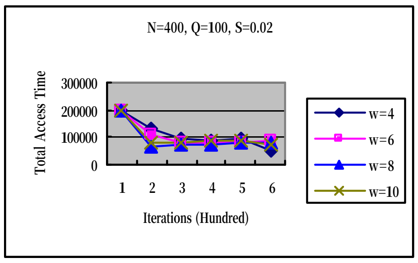
(b) N=900



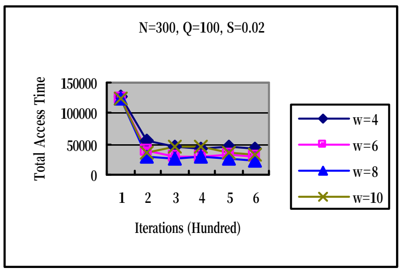
(c) N=700



(d) N=500



(e) N=400



(f) N=300

Figure 4-1：Total Access Time of various numbers of data items with different window sizes and iterations. The query patterns' access frequencies are uniform distribution.

In our algorithm, we just calculate all possible paths which can find the optimal solution. The cost function is described in Section 3.2.2.

4.1 Efficiency of the Various Window Sizes and Iterations

In Figure 4-1, we use 100 query patterns and S is equal to 2%. The query patterns' access frequencies are uniform distribution. When N are equal to 300、400、500、700、900、1000.

We observe the variation of window sizes and iterations in the total access time.

As shown in the result, while W = 10 or W = 8, the total access time will be converged in 200 iterations approximately. But while W = 4 or W = 6, the total access time is converged with more iterations. In other words, if the number of data items becomes greater and window size is smaller, the A* algorithm needs more iterations to find a good broadcast schedule.

4.2 Efficiency of the Number of Data Items

We assume that there are 100 query patterns and the selectivity is 2%. We set the W = 4 and l = W/2. The query patterns' access frequencies are with a normal distribution. We observe the variation of total access time by changing the total number of data items (N). The results are shown in Figure 4-2.

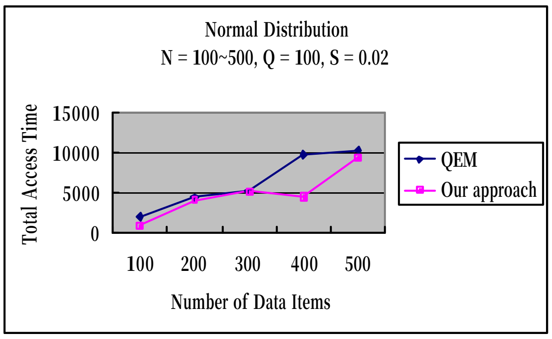


Figure 4- 2：Efficiency of the number of data items with normal distribution.

The improvement ratio with different data items are presented in Table 4-1. The results of A* are superior to QEM in access time. On average, our approach yields improvement of 47.29% over QEM.

Table 4- 1：Improvement ratio with different data items with normal distribution

Normal Distribution			
# of data items	QEM	A*	Improve (%)
100	2192	1052	108.365019
200	4462	4119	8.327263899
300	5198	5134	1.246591352
400	9740	4666	108.7441063
500	10397	9468	9.81199831
Average			47.29899577

We set the W = 4 and l = W/2. The query patterns' access frequencies are with a uniform distribution. There are 100 query patterns and the S is 2%. We observe the total access time variation with the numbers of data items from 100 to 500. The results are shown in Figure 4-3. A* still outperforms QEM approach.

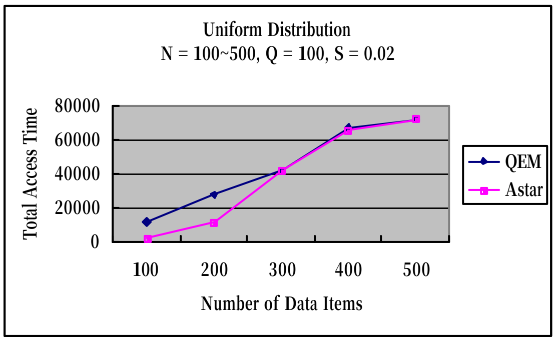


Figure 4- 3：Efficiency of the number of data items with uniform distribution.

The improvement ratio about the change of data items are shown in Table 4-2. The results of A* are better than QEM in access time. On average, our approach yields improvement of 137.17% over QEM.

Table 4- 2：Improvement ratio with different data items with uniform distribution

Uniform Distribution			
# of data items	QEM	A*	Improve (%)
100	11820	1830	545.9016
200	27800	11760	136.3946
300	42200	42080	0.285171
400	67230	65450	2.719633
500	72820	72420	0.552334
Average			137.1707

4.3 Efficiency of the Number of Query Patterns

Figure 4-4 is shown the results with various numbers of query patterns. The numbers of query patterns are among 100 to 900. We set the W = 4 and l = $W/2$. The query patterns' access frequencies are with a normal distribution. The number of data items is 100, and the S is 2%. A* outperforms QEM approach.

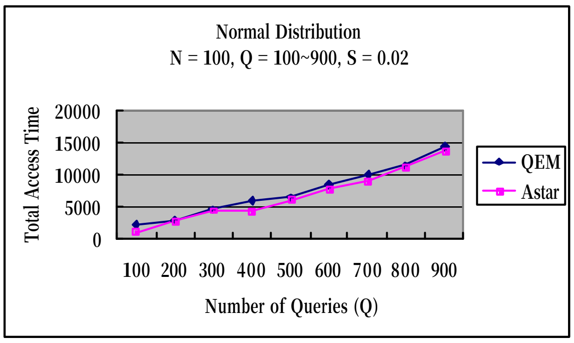


Figure 4- 4：Efficiency of the number of query patterns with normal distribution.

The improvement ratio about the change of data items are shown in Table 4-3. The results of A* are better than QEM in access time. On average, our approach yields improvement of 21.67% over QEM.

Table 4- 3：Improvement ratio with different number of query patterns with normal distribution.

Normal Distribution			
# of queries	QEM	A*	Improve (%)
100	2192	1021	114.6915
200	2966	2798	6.004289
300	4642	4503	3.086831
400	5989	4256	40.71898
500	6418	6110	5.040917
600	8512	7769	9.56365
700	10041	9154	9.689753
800	11571	11381	1.669449

900	14366	13734	4.601718
Average			21.67412

Figure 4-5 is shown the results with various numbers of query patterns. The numbers of query patterns are among 100 to 900. We set the *W* = 4 and l = *W*/2. The query patterns' access frequencies are with a uniform distribution. The number of data items is 100, and the *S* is 2%. Our proposed approach gives better performance than QEM algorithm.

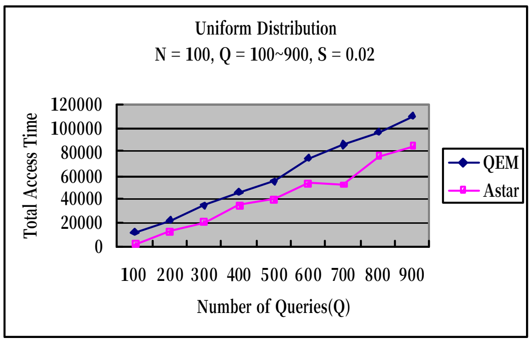


Figure 4- 5：Efficiency of the number of query patterns with uniform distribution.

The improvement ratio with different data items are presented in Table 4-4. The results of A* are superior to QEM in access time. On average, our approach yields improvement of 103.19% over QEM.

Table 4- 4：Improvement ratio with different number of query patterns with uniform distribution

Uniform Distribution			
# of queries	QEM	A*	Improve (%)
100	11820	1790	560.3352
200	21870	13020	67.97235
300	34220	20140	69.91063
400	45510	33960	34.0106
500	55010	40080	37.2505
600	73990	52960	39.70921
700	85360	52360	63.02521
800	96800	75850	27.6203
900	109590	85030	28.88392
Average			103.1909

4.4 Efficiency of Selectivity Parameter *S*

In this section, we observe the variation of selectivity in the total access time. The results are shown in Figure 4-6.We set the *W* = 4 and l = *W*/2. The query patterns' access frequencies are with a uniform distribution. We use 100 data items and 500 query patterns in this experiment. The performance of A* is better than QEM. As a query accesses more data items, our approach still can get a good solution. Particularly, the selectivity is smaller than 3%.

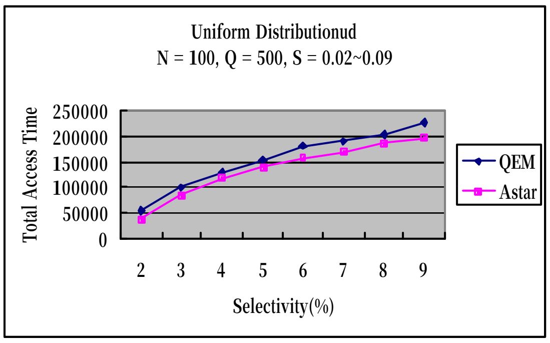


Figure 4-6：Efficiency of different selectivity.

The improvement ratio with different data items are presented in Table 4-5. The results of A* are superior to QEM in access time. On average, our approach yields improvement of 15.95% over QEM.

Table 4-5：Improvement ratio with different selectivity with uniform distribution

Uniform Distribution			
Selectivity	QEM	A*	Improve (%)
2	55010	39670	38.66901941
3	100400	85470	17.46811747
4	131290	118820	10.49486618
5	153570	139380	10.18080069
6	180210	158240	13.88397371
7	191210	169470	12.82822919
8	203810	185640	9.787761258
9	226040	197750	14.30594185
Average			15.95233872

5 Conclusion

In this paper, we have proposed an A* method for the data broadcast problem which mobile clients' access more than one data items. A* uses the branch and bound method to reach its work. We also set a search range and let the data items in the scope are implemented A* in a reasonable time. We compare our method with QEM [14]. The proposed A* strategy is shown to generally outperform QEM.

In the future we will expend this work on multi channels environment and data items with non-uniform lengths.

6 References

[1] Acharya, S., Alonso, R., Franklin, M., and Zdonik, S., "Broadcast Disks: Data Management for Asymmetric Communication Environments," *Proceedings of ACM SIGMOD International Conference*, 1995, pp. 199 – 210.

[2] Acharya, S., Franklin, M., and Zdonik, S., “Disseminating Updates on Broadcast Disks,” *Proceedings of Very Large Data Bases Conference*, 1996, pp. 354 – 365.

[3] Acharya, S., Franklin, M., and Zdonik. S., “Dissemination-based Data Delivery Using Broadcast Disks,” *Proceedings of IEEE Personal Communications*, volume 2, issue 6, 1995, pp. 50-60.

[4] Aksoy, D., and Franklin, M., “R x W: A Scheduling Approach for Large-Scale On-Demand Data Broadcast,” *Proceedings of IEEE Transactions on Networking*, volume 7, issue 6, 1999, pp. 846-860.

[5] Asano, T., “An Optimal Gate Placement Algorithm for MOS One-Dissensional Arrays,” *Journal Digital System*, 1982

[6] Bar-Noy, A. and Shilo, Y., “Optimal broadcasting of two files over an asymmetric channel,” *Proceedings of IEEE INFOCOM Conference*, volume 1, 1999, pp. 267-274.

[7] Baruah, S., Bestavros, A., “Real-Time Mutable Broadcast Disks,” *Proceedings of RTDB*, 1997, pp. 3-21.

[8] Buttazzo, G. and Sensini, F., “Optimal deadline assignment for scheduling soft aperiodic tasks in hard real-time environments,” *IEEE Transactions on Computers*, volume 48, issue 10, 1999, pp. 1035-1052.

[9] Cai, H., “A Data Path Layout Assembler for High Performance DSP Chips,” *DAC*, 1990, pp. 306-311.

[10] Cheng, M., Sun, J., Min, M., and Du, Lee, “Energy-efficient Broadcast and Multicast Routing in Ad Hoc Wireless Networks,” *Proceedings of IEEE International Conference on Performance, Computing, and Communications*,, 2003, pp. 87-94.

[11] Cho, J., Oh, S., Kim, J., and Lee, J., “Neighbor Caching in Multi-Hop Wireless Ad Hoc Networks,” *IEEE Transactions on Communications Letters*, volume 7, issue 11, 2003, pp. 525-527.

[12] Chow, C. Y., Leong, H.V., and Chan, A., “Cache signatures for peer-to-peer cooperative caching in mobile environments,” *Proceedings of IEEE 18th International Conference on Advanced Information Networking and Applications*, volume 1, 2004, pp. 96-101.

[13] Chung, Y. D. and Kim, M. H., “An Index Replication Scheme for

Wireless Data Broadcasting," *Journal of ACM Systems and Software*, volume 51, issue 3, 2000, pp. 191-199.

[14] Chung, Y. D. and Kin M. H., "Effective Data Placement for Wireless Broadcast," *Proceedings of ACM Distributed and Parallel Database*, volume 9, issue 2, 2001, pp. 133-150.

[15] Chung, Y. D., Bang, S. H., and Kim, M. H., "An efficient broadcast data clustering method for multipoint queries in wireless in formation systems," *The Journal of ACM Systems and Software*, volume 64, issue 3, 2002, pp. 173-181.

[16] Date, C. J., "An Introduction to Database Systems, 7th Edition," *Addison Wesley*, 2000.

[17] Demir, O.E., and Aksoy, D., "Energy-efficient broadcast-based event update dissemination," *Proceedings of IEEE International Conference on Performance, Computing, and Communications*, 2004, pp. 477-482.

[18] Dykeman, H. D., Ammar, M. H., and Wong, J. W., "Scheduling algorithms for videotext systems

[19] Etsuko, Y., Takahiro, H., Masahiko T., and Shojiro N., "Scheduling and caching strategies for broadcasting correlated data," *Proceedings of ACM symposium on Applied computing*, 2001, pp. 504-510.

[20] Fang, Q., Vrbsky V., Dang Y., and Ni, W., "A Pull-Based Broadcast Algorithm that Considers Timing Constraints," *Proceedings of IEEE International Conference on Parallel Processing Workshops*, 2004, pp. 46-53.

[21] Gunes, M., "Routing Algorithms for Mobile Multi-Hop Ad-Hoc Networks," *Citeseer of International Workshop NGNT*, 2004, pp. 10-24.

[22] Hsu, C. H., Lee, G., and Chen, A.L.P., "Index and Data Allocation on Multiple Broadcast Channels Considering Data Access Frequencies," *Proceedings of IEEE the Third International Conference on Mobile Data Management*, 2002, pp. 87-93.

[23] Hung, J. J., and Leu, Y., "Efficient

index caching schemes for data broadcasting in mobile computing environments," *Proceedings of IEEE 14th International Workshop on Database and Expert Systems Applications*, 2003, pp. 139-143.

[24] Imielinski, T., Viswanathan, S., and Badrinath, B. R., " Data on Air: Organization and Access, " *IEEE Transaction on Knowledge and Data Engineering*, volume 9, issue 3, 1997, pp. 353-372.

[25] Imielinski, T., Viswanathan, S., and Badrinath, B. R., " Energy Efficient Indexing on Air, " *Proceedings of ACM SIGMOD International Conference*, volume 23, 1994, pp. 25-36.

[26] Jin, Seung., and Jae, Woo., and Joo, Young., "Efficient Broadcast Schemes with Transmission Power Control in Mobile Ad Hoc Networks," *Proceedings of IEEE Communications Society*, volume 7, 2004, pp. 3859 – 3863.

[27] Lam, K. Y., Chan, E., and Yuen, C., "Approaches for broadcasting temporal data in mobile computing systems," *The journal of ACM Systems and Software*, volume 51, issue 3, 2000, pp. 175-189.

[28] Lee, G., Lo, S. C. and Chen, A. L. P., "Data Allocation on Wireless Broadcast Channels for Efficient Query Processing," *IEEE Transaction on Computers*, volume 51, issue 10, 2002, pp. 1237-1252.

[29] Lee, S. J., Kitsuregawa, M., and Hwang, C. S., " Efficient processing of wireless read-only transactions in data broadcast, " *Proceedings of IEEE Research Issues in Data Engineering: Engineering E-Commerce/E-Business Systems*, 2002, pp. 101-111.

[30] Lee, G., Yeh, M. S., Lo, S. C., and Chen, A. L. P., " A Strategy for Efficient Access of Multiple Data Items in Mobile Environments, " *Proceedings of IEEE Third International Conference on Mobile Data Management*, 2002, pp. 71-78.

[31] Lim, S. H., and Kim, J.H., " Real-time broadcast algorithm for mobile computing, " *The Journal of ACM Systems and Software*, volume 69, issue 1-2, 2004, pp. 173-181.

[32] Lo, S. C., and Chen, A.L.P., " Optimal index and data allocation in multiple broadcast channels, " *Proceedings of IEEE 16th International Conference on Data*

Engineering, 2000, pp. 293-302.

[33] Ni,W., Fang, Q., and Vrbsky, V., "A Lazy Data Request Approach for On-demand Data Broadcasting," *Proceedings of IEEE International Conference on Distributed Computing Systems Workshops*, 2003, pp. 790-796.

[34] Peng, W. C. and Chen, M. S., "Dynamic Generation of Data Broadcast Programs for a Broadcast Disk Array in a Mobile Computing Environment," *Proceedings of ACM International Conference on Information and Knowledge Management*, 2000, pp. 38-45.

[35] Peng, W. C., and Chen, M. S,, "Developing Data Allocation Schemes by Incremental Mining of User Moving Patterns in a Mobile Computing System," *IEEE Transactions on Knowledge and Data Engineering*, volume 15, issue 1, 2003, pp. 70-85.

[36] Peng, W. C., Huang, J. L., and Chen M. S., "Dynamic Leveling: Adaptive Data Broadcasting in Mobile Computing Environment," *ACM/Kluwer Mobile Networks and Applications*, 2003, pp. 355-364.

[37] Ramanaiah, O.B.V., and Mohanty, H., "NICD: a novel indexless wireless

[38] Saygin, Y., and Ulusoy, O., "Exploiting data mining techniques for broadcasting data in mobile computing environments," *IEEE Transactions on Knowledge and Data Engineering*, volume 14, issue 6, 2002, pp. 1387-1399.

[39] Sun, W., Shi W., Shi, B., and Yu, Y., "A Cost-Efficient Scheduling Algorithm of On-Demand Broadcasts," *Proceedings of ACM Wireless Networks*, volume 9, issue 3, 2003, pp. 239-247.

[40] Tan, K. and Yu, J. X., "Generating Broadcast Programs that Support Range Queries," *IEEE Transactions on Knowledge and Data Engineering*, volume 10, issue 4, 1998, pp. 668-672.

[41] Thomas H. C., Charles E. L., and Ronald L. R., "Introduction to algorithms," *McGraw-Hill*, 2001.

[42] Waluyo, A.B., Srinivasan, B., and Taniar, D., "A Taxonomy of Broadcast Indexing Schemes for Multi Channel Data Dissemination in

Mobile Database," *Proceedings of IEEE the 18th International Conference on Advanced Information Networking and Application*, volume 1, 2004, pp. 213-218.

[43] Waluyo, A.B., Srinivasan, B., and Taniar, D., "Global Indexing Scheme for Location-Dependent Queries in Multi Channels Mobile Broadcast Environment," *Proceedings of IEEE 19th International Conference on Advanced Information Networking and Applications*, volume 1, 2005, pp. 1011-1016.

[44] Wu, Y., and Cao, G., "Stretch-Optimal Scheduling for On-Demand Data Broadcasts," *Proceedings of IEEE International Conference on Computer Communications and Networks*, 2001, pp. 500-504.

[45] Xu, Ji., Xu, Jianliang., and Li, Bo., "A Cooperative Caching Algorithm for Multi-Cell Data Broadcasting," *Proceedings of IEEE Communications Society*, volume 7, 2004, pp. 4072-4076.

[46] Yajima, E., Hara, T., Tsukamoto, M., and Nishio, S., "Scheduling and Caching Strategies for broadcasting Correlated Data," *Proceedings of ACM Symposium on Applied*

[47] Zhang, J. and Gruenwald, L., "Optimizing Data Placement Over Wireless Broadcast Channel For Multi-Dimensional Range Query Processing," *Proceedings of the 2004 IEEE International Conference on Mobile Data Management*, 2004, pp. 256-265.