

Scheduling on airdisks: Efficient access to personalized information services via periodic wireless data broadcast

Veena Gondhalekar
Dept. of Electrical & Comp. Eng.
Univ. of Texas at Austin

*Ravi Jain**
Applied Research
Bellcore

John Werth
Dept of Computer Sciences
Univ. of Texas at Austin

Abstract. Recently there has been considerable interest in delivering information to distributed mobile clients via wireless broadcast. Information transmitted periodically over wireless media can be regarded as a virtual disk, which we call an *airdisk*, analogous to a standard magnetic disk. Airdisks offer an efficient mechanism for delivering personalized information services to mobile clients with portable or laptop computers by broadcasting data and allowing clients to filter out the items of interest to them.

We study the problem of scheduling the order in which data items are broadcast so as to minimize the access time of the clients, focusing on the case where the server inserts an index at the start of the broadcast period. We observe that the problem is analogous to that of determining how data should be laid out on the disk, and show that the problem is in general NP-complete. We develop a branch-and-bound procedure for solving the problem optimally, and then develop a fast, simple heuristic. The results of our simulation experiments show that the heuristic runs substantially faster than the branch-and-bound procedure, and yet produces schedules that are only slightly longer.

1 Introduction

The delivery of information over wireless media is rapidly becoming an important and expanding application area. We consider the delivery of personal information services via periodic wireless broadcast described as follows. An information services provider generates data. A fixed, (logically) centralized information server periodically broadcasts the data to a large number of mobile clients via a wireless medium. As an example, the server receives updates of current stock prices from the stock exchange, updates its database, and periodically broadcasts the updated information to the clients. The information may also be road traffic information, e.g., similar to the personalized information provided by the SCOUT traffic information system [14]. The clients receive the broadcasts and filter out the information which is not desired. (Note that a client is a device, while the ultimate human recipient of the data is called the user.) This approach is particularly well-suited for applications, such as traffic information [14], where it is expected that there will be

substantial commonality in the items of interest to users, and the wireless communication bandwidth is at a premium. Note that a mobile client device need not necessarily be continuously “tuned in” at full power while waiting for items of interest; by slipping into a *doze mode* while waiting, substantial savings in power consumption can be achieved.

We can model the periodic broadcast of data as a virtual disk, which we call an *airdisk* [9, 10]. This is similar to the notion of broadcast disks developed by Acharya et al [1]. (In the rest of this paper, the term “disk” or “airdisk” is used to refer to a virtual disk.) In the rest of this section we describe the system model. In the next section we describe the indexed data layout problem: how to minimize the average time experienced by clients for reading data items of interest from the broadcast when the broadcast includes an index; we show that solving this problem optimally is NP-complete. In sec. 3 we describe a branch-and-bound algorithm for the indexed data layout problem. In sec. 4 we present a fast suboptimal heuristic for its solution, as well as simulation results on the performance of the branch-and-bound algorithm and the heuristic. In sec. 5 we discuss our results and related work.

1.1 System model

In the airdisk model, a (logically) centralized server broadcasts data (writes on the disk) and many clients can receive the broadcast (read the disk) and also send messages to the server to modify the content of the next broadcast (i.e., write the disk).

Each broadcast period (see Fig. 1) consists of a *preamble* used for synchronization followed by a *period flag* indicating the start of the data in the period. In general, the period flag is followed by an *index* which gives the sequence of data items broadcast in this period. Each data item has an *item header* at the start (identifying the item and, e.g., its length) and a *trailer* indicating its end. Depending upon the scheme used for sending the data, the preamble, index, headers and trailers may be of zero or more bits. The item headers and trailers are not required if all items are of the same length, and that length is fixed and known to the clients. An index is not required if the data items are fixed and sent in a fixed sequence known to the clients.

We define the *mean rotational latency* as the average time that a client which starts reading the airdisk at an arbitrary point of time has to wait for the start of the

*Address correspondence to Ravi Jain, Applied Research, Bellcore, 331 Newman Springs Rd., Red Bank, NJ 07701. Phone: (908)-758-2844. Fax: (908)-758-4371. E-mail: rjain@bellcore.com.

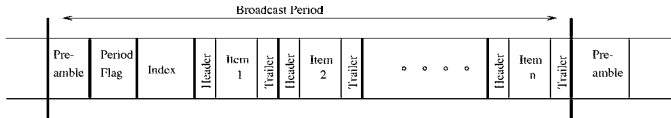


Figure 1: A general periodic wireless broadcast

next rotation (broadcast period); an expression for this quantity can be derived [9, 10].

2 Dynamic data layout

We have previously considered the situation in which communication is simplex and the clients simply filter the data broadcast by the server [9, 10]. In this paper we consider the situation where the server has some information about the data items which clients are interested in, and dynamically can change the data layout in response to client read patterns. (We have discussed schemes for obtaining this client interest information in [9]; for the moment, suppose this information is available.) In that case, it may be possible to improve the performance of the airdisk system by exploiting this client interest information [8, 4]. We examine how to do so by scheduling the order of individual items in the periodic broadcast. Scheduling the periodic broadcast is analogous to the problem of laying out the data on the airdisk. We have previously [9, 10] considered the case in which there is no index on the airdisk (as assumed in [4]); in this paper we consider the situation where there is an index (as assumed in [8].) Note that in the first situation client energy consumption may be high since the client has to read the airdisk all the time, while in the second situation more wireless bandwidth is consumed for the index.

2.1 Indexed data layout

Suppose the airdisk has an index which specifies the sequence of data items in each rotation, and for simplicity, assume that all the data items are of the same length. A client then accesses the data items of interest by first reading the index and then reading only each data item of interest. A client starts reading the disk at an arbitrary instant of time, and a client’s access is not considered completed unless it has read all the items it is interested in. The access time for a client will be the sum of four components: (1) the rotational latency for the index, i.e., waiting for the index of the next rotation to appear, (2) the data transfer time to read the index, (3) the *item reach*, i.e., the time until the last item the client is interested in appears, and (4) the data transfer time to read the last item. (Note that a client *always* waits for the index before any data transfer, and that the data transfer time for items other than the last one is overlapped with the item reach.) We are interested in minimizing the mean access time over all clients; we call this the Indexed Data Layout (IDL) problem.

The mean rotational latency and data transfer times are fixed once the set of items in a given rotation is decided. However, the item reach can still be varied by vary-

ing the sequence of the items on the disk. Since client interest information is known in advance, a disk layout can be chosen to reduce the mean item reach and hence the mean access time.

We cast the problem of minimizing item reach in graph-theoretic terms as follows. Let each data item be represented by a vertex of a graph, and let the vertices be numbered consecutively. We will introduce a hyperedge for representing the items of interest to each client. Recall that a hyperedge is simply an edge that can connect more than two vertices; i.e., a hyperedge is an arbitrary subset of the set of vertices, instead of only a pair of vertices. Self-loops (i.e., hyperedges which connect only one vertex) are allowed. Thus, for each client we introduce one hyperedge which connects the vertices corresponding to the data items the client is interested in. As an example, in Fig. 2 (a), we show a graph corresponding to a broadcast period with five data items, numbered 1 to 5, which are modeled as five vertices numbered 1 to 5. There are eleven clients, shown by eleven edges; in this example each client is interested in only two items and so there are no hyperedges. Parallel edges, such as the two edges connecting vertices 1 and 2, correspond to clients which are interested in the same data items. The length of a hyperedge is the difference between its lowest and highest numbered vertex. For example, in Fig. 2 (a), the length of the edge connecting vertices 1 and 4 is 3.

We can show that the indexed data layout problem is in general NP-complete i.e., computationally intractable.

Theorem 2.1 *The problem of minimizing the mean access time, over all clients, for the indexed data layout problem is NP-complete, even if each client is interested in only two items.*

The proof is omitted for brevity (see [6].) However, note that minimizing mean access time reduces to minimizing mean item reach; the latter problem can be rewritten as follows. Given a graph $G = (V, E)$ with vertex set V and edges E , find a one-to-one function $f : V \rightarrow \{1, 2, \dots, |V|\}$ such that

$$s_2 = \sum_{(u,v) \in E} \max(f(u), f(v)) \quad (1)$$

is minimized.

Note that for the special case where every client is only interested in one item on every rotation, the indexed data layout problem can be solved trivially by ordering the items in descending order of popularity, i.e., items with the most client interest first.

3 An optimal algorithm for data layout

We will continue further discussion of the indexed data layout problem, and its solution, in terms of its graphical formulation in sec. 2.1. It is clear that the execution time for solving IDL by the naive algorithm, i.e., exhaustive enumeration, will be prohibitively large for all but a small number of items, n . We develop an optimal algorithm for this problem using the well-known branch-and-bound

approach [12]. The better the lower bound used in the B&B algorithm, the more branches of the enumeration tree will be eliminated. In the following we present a lower bound on the mean item reach. We will use the following observations.

Def. For a permutation f of the vertices of G , where f is a one-to-one function $f : V \rightarrow \{1, 2, \dots, |V|\}$, the *left degree* of a vertex under f is the number of edges it has connected to vertices which are placed earlier in f , i.e., the left degree of vertex v is

$$ld(v) = |\{(u, v) : (u, v) \in E \wedge f(u) < f(v)\}|$$

Observation. For a given vertex permutation f of the vertices of graph $G = (V, E)$,

$$s_2 = \sum_{(u,v) \in E} \max(f(u), f(v)) = \sum_{v \in V} f(v)ld(v) \quad (2)$$

This observation can be explained operationally as follows. The first equation above calculates s_2 by considering each edge, and finding the position of its right endpoint, while the second does so counting the number of edges which are terminated at each vertex, and multiplying by the position of that vertex.

It is clear from Eq. 2 that to minimize s_2 a permutation f which places vertices in descending order of left degree is desired. We use this to motivate the following lower bound.

Lemma 3.1 *Suppose that i out of n vertices have been placed, i.e., the first i positions of the permutation have been decided. Let $E_i \subseteq E$ denote the edges which are completed, i.e., have both endpoints placed, after $i \leq n$ vertices have been placed. Then a lower bound on the objective function s_2 of Eq. 1 for the remaining vertices is given by*

$$L_1 = \sum_{j=i+1}^n j * \max(0, \min(|E| - |E_j|, d(S(j-i)))) \quad (3)$$

where $d(S(k))$ is the degree of the k th vertex in the list S created by sorting the remaining vertices in order of descending degree.

The proof is omitted for brevity; see [6].

4 A fast heuristic for data layout

Although the B&B algorithm is much faster than exhaustive enumeration, (as we verified experimentally for values up to $n = 8$ [6]), it is likely to be too slow for many practical applications. It may be the case that B&B could be made faster by designing a more sophisticated lower bound. An alternative approach is to design a heuristic which, while not guaranteeing optimality, gives results close to optimal for a much smaller execution time than B&B. We design two such heuristics below, and for the second, more sophisticated one, present results of simulation experiments showing that, for the situations studied, it does indeed behave as desired.

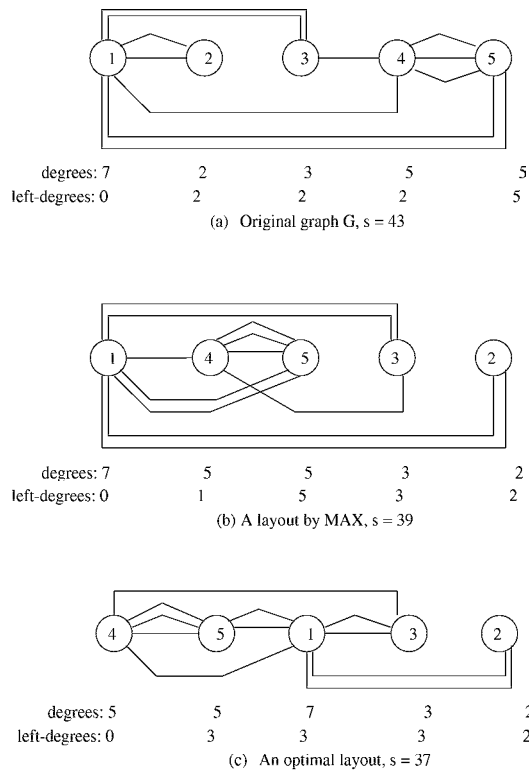


Figure 2: Example of (a) an input graph G , whose ordering is improved by MAX as shown in (b), but which is greater than the cost of (c) an optimal layout.

To motivate the first heuristic, consider again the special case we mentioned in sec. 2, namely, where every client is only interested in one item. In that case the data items should simply be ordered in descending order of degree.

Def. The *MAX* heuristic consists of ordering the vertices (data items) by descending degree.

Clearly, MAX will not produce an optimal layout when (some) clients may be interested in more than one item; an example is shown in Fig. 2. Here the five vertices, which are ordered in some random permutation as shown in Fig. 2 (a), are ordered by descending degree in Fig. 2 (b), with ties broken arbitrarily. While the random permutation has an objective function value of 43, the layout by MAX improves it to 39. However, it can be shown that the optimal layout is the one shown in Fig. 2 (c), which has an objective function value of 37.

However, the virtue of MAX is that for a graph with n vertices and m edges it takes $O(m)$ time to calculate the vertex degrees, and $O(n \log n)$ to sort, giving a total time of only $O(m + n \log n)$.

A better heuristic would be to modify the ordering obtained from MAX by an iterative improvement. From our observation in Eq. 2, to minimize s_2 a permutation which places vertices in descending order of left degree is desired.

Def. The *MAX-LD* heuristic consists of two steps. The first step is to obtain an initial ordering by sorting the vertices by descending degree, i.e., using MAX. In the second step, the following operation is repeated for $i =$

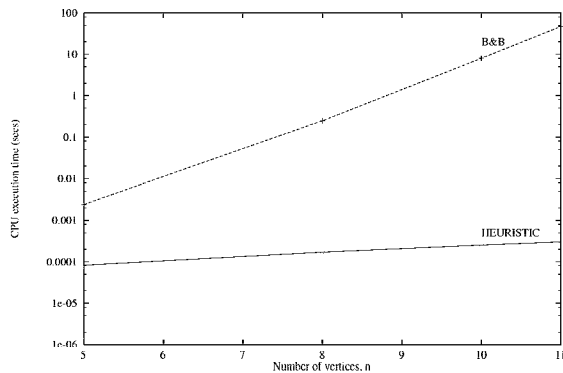


Figure 3: Comparison of mean execution time of B&B and the MAX-LD heuristic for graphs with unit-weight edges and edge density 0.5. *Note log scale on y-axis.*

1, ..., n - 1: if the left degree of vertex $i + 1$ exceeds that of vertex i , the positions of the two vertices are interchanged.

MAX-LD will also not always produce an optimal ordering. For example, considering the instance G shown in Fig. 2, MAX-LD will obtain the vertex ordering (1, 5, 4, 3, 2). We leave it to the reader to verify that this ordering has a cost of 38, which, while better than the cost obtained by MAX, is still not optimal.

The MAX-LD heuristic will take $O(m + n \log n)$ time to run MAX, followed by $O(m + n)$ time to perform the left-degree check, for a total of $O(m + n \log n)$.

4.1 Experimental results

We show preliminary experimental results from a set of experiments which compare the performance of MAX-LD with B&B. The MAX-LD algorithm was implemented as a C program. The experiments were all performed by compiling the programs (with optimization level 4 enabled) and executing on a Sun Sparc 1 station.

Expt. 1. The experiment was performed for several values of the number of vertices n . For each value of n , 25 random graphs with edges of unit weight were generated (i.e., no parallel edges were allowed.) The edge density was 0.5, i.e., the number of edges is half the maximum possible number of edges. For each graph, the CPU time for executing the heuristic was computed, as was the CPU time for executing B&B; these values were averaged over the 25 random graphs. In addition, for each input graph, the cost $s_2(MAX - LD)$ of the ordering obtained by MAX-LD was found, as was the optimal cost $s_2(B\&B)$ found by B&B, and the ratio $\frac{s_2(MAX - LD) - s_2(B\&B)}{s_2(B\&B)}$ was computed; this ratio was always averaged over the 25 random graphs.

In Fig. 3, the mean CPU time for B&B is compared with that for MAX-LD, for values of $n = 5, 8, 10, 11$. It is clear that MAX-LD takes much less time than B&B in this case. Fig. 4 shows that the penalty paid by MAX-LD in terms of the increased cost s_2 is only a few percent on average for this experiment.

The comparison of the cost penalty paid by MAX-LD

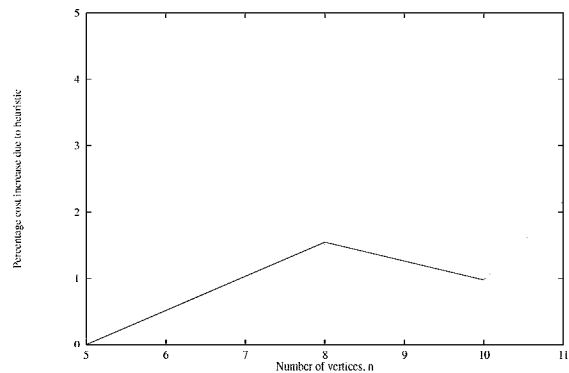


Figure 4: Comparison of mean percentage increase in cost s_2 when the heuristic MAX-LD is used, compared to the optimal solution. (Graphs with unit-weight edges and edge density 0.5.)

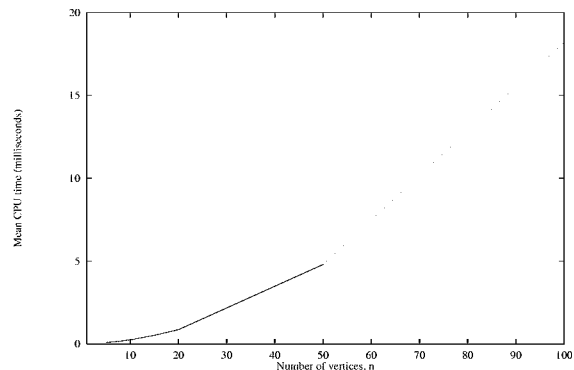


Figure 5: Mean CPU time for MAX-LD heuristic for graphs with unit-weight edges and edge density 0.5.

cannot be extended for large n in general, because the execution time to calculate the optimal solution, even using B&B, becomes prohibitive. We can, however, measure the running time of MAX-LD.

Expt. 2. The experiment consisted of running the MAX-LD heuristic in the same manner as for Expt. 1 above, but continuing it for $n = 15, 20, 50, 100$. (The B&B was not run.)

Fig. 5 shows the mean CPU time for the heuristic for various values of n . It can be shown that the measured mean CPU time grows roughly as $O(n^2)$. Recall that the edge density of the input graphs is 0.5, i.e., the expected number of edges is $0.5 * n(n - 1)/2 = O(n^2)$; thus the measured values for execution time are consistent with the theoretical analysis.

5 Discussion and related work

We have extended the simulation experiments shown above to consider the case that the edges are weighted (i.e., there are parallel edges in the graph). The results are qualitatively similar to those shown here, and are omitted for brevity. We are currently also experimenting with

different (sparser) edge densities and hyperedges. In general we expect that a simple heuristic like MAX-LD will provide excellent results. We are considering additional heuristics which may provide slightly better results, as well as a more sophisticated lower bound which can be used to improve the performance of B&B and hence allow experimentation with larger graphs. We are currently also investigating an implementation of the airdisk model in an experimental wireless LAN environment.

In other directions related to this work, we have considered the problem of obtaining information about data access patterns in a wireless mobile environment, and discussed several alternative solutions [9]. Another issue with airdisks is that as the quantity of data to be broadcast increases, the delays involved in accessing data increase. We use the airdisk model to borrow the solution to this problem used for magnetic disks, i.e., to improve performance via parallelism while maintaining availability via redundancy, as in the Redundant Arrays of Inexpensive Disks (RAID) approach [13, 7] which is in successful and widespread use. We call this design the *AirRAID*, and we have described it briefly in [9].

The ideas of periodic data broadcast were previously introduced by the Datacycle project at Bellcore [5] and by Imielinski et al [8]. There has been a surge of related work on periodic wireless broadcast recently. As mentioned previously, the airdisk model [9] is similar to the notion of broadcast disks studied by Acharya et al [1]. In [1] the authors address two issues, the first being related to how to select the frequency with which data items are broadcast in a broadcast period, and the second issue dealing with how an individual client should best manage its cache. Other work on broadcast disks [2, 3] has focused on policies by which clients may prefetch data. In [11] the authors present single and multi-level signature schemes, as well as schemes for caching the signatures at the mobile clients. Finally, recent work by [15] has considered data broadcast scheduling, although not specifically for the indexed data layout problem as discussed here.

Acknowledgements. We thank Bill Aiello and Sandeep Bhatt of Bellcore for several useful and stimulating discussions regarding the indexed data layout problem.

References

- [1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: Data management for asymmetric communication environments. In *Proc. SIGMOD*, June 1995.
- [2] S. Acharya, M. Franklin, and S. Zdonik. Dissemination-based data delivery using broadcast disks. *IEEE Pers. Comm.*, Dec. 1995.
- [3] S. Acharya, M. Franklin, and S. Zdonik. Prefetching from a broadcast disk. In *Proc. Intl. Conf. Data Eng.*, Feb. 1996.
- [4] T. Chiueh. Scheduling for broadcast-based file systems. In *Proc. MOBIDATA Workshop*. Rutgers University, Nov. 1994.
- [5] T. F. Bowen et al. The Datacycle architecture. *Comm. ACM*, pages 71–81, Dec. 1992.
- [6] V. Gondhalekar, R. Jain, and J. Werth. Scheduling on airdisks: Efficient access to personalized information services via periodic wireless data broadcast. Technical Report CS-TR-96-25, Univ. Texas at Austin, Dept. of Comp. Sci., Oct 1996.
- [7] J. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, San Mateo, CA, 1990.
- [8] T. Imielinski, S. Viswanathan, and B.R. Badrinath. Energy efficient indexing on air. In *Proc. SIGMOD*, pages 25–36, 1994.
- [9] Ravi Jain and John Werth. Airdisks and AirRAID: Modeling and scheduling periodic wireless data broadcast. DIMACS Tech. Report 95-11, Rutgers Univ., May 1995.
- [10] Ravi Jain and John Werth. Airdisks and AirRAID: Modeling and scheduling periodic wireless data broadcast. *ACM SIGARCH Comp. Arch. News.*, Oct. 1995.
- [11] W. C. Lee and D. K. Lee. Using signature techniques for information filtering in wireless and mobile environments. *Distrib. and Par. Databases*, 4(3), 1996. (To appear: Special Issue on Databases and Mobile Computing.)
- [12] C. H. Papadimitriou and K. Stieglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, 1982.
- [13] David Patterson, Garth Gibson, and Randy Katz. A case for redundant arrays of inexpensive disks (RAID). In *ACM SIGMOD Conference*, pages 109–116, June 1988.
- [14] S. Schulman, R. Jain, M. Kramer, and A. Virmani. Traveler information: Tailored to meet the needs of the traveler. In *Proc. Intl. Transp. Sys. (ITS) America Conf.*, Apr. 1996.
- [15] N. Vaidya and S. Hameed. Data broadcast scheduling (Part 1). Technical Report TR 96-012, Texas A& M Univ., Dept of Comp. Sci., May 1996.