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

Veena Gondhalekar
Dept. of Electrical & Computer 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 from a fixed server to large numbers of geographically distributed mobile clients, by broadcasting data and allowing clients to filter out the items of interest to them.

Mobile clients are typically portable devices or Personal Digital Assistants which have restricted resources, and operate using limited battery power. We study the problem of scheduling the order in which data items are broadcast so as to minimize the access time of the clients. 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 then focus on the data layout problem in the situation where the server inserts an index at the start of each broadcast period. We develop a branch-and-bound procedure for solving the problem optimally, and then develop a fast, simple heuristic. We present the results of simulation experiments to evaluate these procedures. As expected, the branch-and-branch procedure produces optimal solutions substantially faster than a naive enumeration algorithm. However, the heuristic is found to run substantially faster than the branch-and-bound procedure, and yet produces schedules that are only slightly longer. We end with a brief discussion of further experimental work in progress and conclude with a summary.

*Address correspondence to Ravi Jain, Applied Research, Bellcore, 445 South St, Morristown, NJ 07960. e-mail: rjain@thumper.bellcore.com

1 Introduction

The delivery of information over wireless media is rapidly becoming an important and expanding application area. It seems likely that the use of various wireless media, such as paging, FM subcarrier, cellular data, and PCS wireless networks [23], will continue to increase for delivery of a wide variety of data applications. Applications in the realm of personal information services, such as the delivery of periodically updated news, weather, stock, traffic, and sports information to users who are geographically distributed and mobile, appear to be poised for explosive growth and widespread proliferation [21]. We focus on this class of distributed information systems and applications in this paper.

The constraints imposed by wireless communication and user mobility, and the resulting limitations in user terminal capabilities, pose interesting challenges in the design of schemes for efficient delivery of personal information services. A significant amount of attention has been paid to the development of wireless data protocols for the various wireless communication media mentioned above. Imielinski et al [14] have considered the problem of retrieving data using an index, as is commonly done for magnetic disks, where both the data and the index are broadcast periodically over a wireless medium. This approach builds upon previous work done at Bellcore on the Datacycle computer architecture [11, 10], in which data is periodically broadcast over a fixed high-bandwidth *wired* network to clients that filter the information relevant to them, rather than being sent via point-to-point messages to individual clients. This approach is particularly well-suited for applications, such as traffic information [21], 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; as we will discuss later, by slipping into a *doze mode* while waiting, substantial savings in power consumption can be achieved.

The ideas of Datacycle and Imielinski et al can be taken a step further by modeling the periodic broadcast of data as a virtual disk, which we call an *airdisk* [15, 16]. This is similar to the notion of broadcast disks developed by Acharya et al [1]; in section 7 we will briefly compare our work with other related work in this area. Once the periodic data broadcast has been modeled as a magnetic disk, its performance characteristics can be cast in the same terms as those used for magnetic disks, such as rotational latency, seek time and transfer rate. (In the rest of this paper, the term "disk" or "airdisk" is used to refer to a virtual disk, while a standard magnetic disk will be explicitly called a "magnetic disk".) In sec. 2 we describe the airdisk model, where 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).

In sec. 3 we consider the case where the airdisk is a One-Writer Many-Reader (OWMR) disk. In this case only the server can write the disk. We show expressions for the maximum and mean rotational latency of such a disk under certain assumptions.

One of the significant advantages of an airdisk compared to a magnetic disk is that the layout of the data on the disk can be changed easily in response to changes in the data access patterns. For magnetic disks such operations are sometimes carried out in order to overcome the I/O performance bottleneck presented by magnetic disks in parallel computing. For example, del Rosario et al [9] describe compiler techniques to detect different data access patterns in different phases of a parallel program; they then show how changing the data layout between two phases of a parallel program can significantly improve overall performance. However, such operations are expensive for magnetic disks as they involve reading, buffering, and re-writing of data. For airdisks, on the other hand, where data is broadcast afresh at every period, they involve negligible overhead at the sender and can be of substantial benefit in improving performance and conserving the limited resources of mobile clients.

In sec. 4 we study how the data layout on an airdisk could be changed, if information about data access patterns is available, so as to minimize mean access time. The data layout for a given broadcast period is determined simply by the order in which data items are broadcast by the sender. We use airdisk access time as a measure of performance. (We also briefly discuss issues of energy consumption at the mobile client.) We model two problems of determining the optimal broadcast scheduling (i.e., airdisk data layout) in graph-theoretical terms and show that they are NP-complete [12], i.e., computationally intractable. The two problems correspond to situations in which an index is included in the broadcast and where it is not.

In sec. 5 we focus on the data layout problem in the situation where an index is broadcast along with the data. We develop an optimal algorithm for solving the problem using the branch-and-bound procedure, developing a simple and provably correct lower bound for this purpose. In sec. 6 we develop a pair of fast simple heuristics for this problem, and present experimental results on the behavior of the more sophisticated one of them. In sec. 7 we have a brief discussion of ongoing work, and we end with a summary.

2 Disks and airdisks

We first summarize magnetic disk terminology [13]. A magnetic disk consists of a set of circular *platters*, each of which rotates under a read/write *head* which can move radially across the platter surface. Typically, only a single platter can be read or written at a time. The platter is divided radially into *sectors*. An annular portion of data on the platter, which can be accessed by moving the head only once to the appropriate radial position, is called a *track*. The time to access data on the magnetic disk can be roughly divided into three components. The *seek time* is the time to move the head from its current position to the appropriate track. The *rotational latency* is the time which the head spends waiting for the platter to rotate so that the desired sector is accessible. The *transfer time* is the time required by the head to then physically read or write the data.

2.1 System model

We consider the delivery of personal information services via 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 [21]. The clients receive the broadcasts and filter out the information which is not desired. A client typically consists of a battery-powered portable device, such as a palmtop Personal Digital Assistant (PDA) or a laptop computer. (Note that a client is a device, while the ultimate human recipient of the data is called the user.)

In the simplest case, data transmission is simplex (one-way), from the server to the clients. Communication could also be half- or full-duplex. The communication channel from the server to the clients is called the *downlink*, and from the clients to the server is called the *uplink*. If an uplink channel exists it is shared amongst the clients by means of ALOHA or Ethernet-type contention protocols, or using TDMA, FDMA, or CDMA multiplexing [22]. The clients can use the uplink channel to send read or write requests to the server. The server responds to read and write requests by modifying the stream of data items sent in subsequent broadcast periods. In this paper we will assume that the uplink, if any, is not used for sending acknowledgements to the server about which data items have been correctly received, since the data items sent by the server are not addressed to individual clients and, depending upon radio channel conditions, some clients may receive them correctly and some may not. Thus the server cannot know which clients received the information correctly and which did not. The server inserts forward error correction [22] encoding into the data written onto the airdisk, and typically periodically re-broadcasts the data even if it has not been updated by the information provider.

The architecture and details of the communications system depend upon the technology used. If paging is used, the wireless bandwidth is low and (until very recently) limited to simplex communication. FM subcarrier communication can potentially provide higher bandwidth and full-duplex communication, although currently most systems are used in low-bandwidth simplex mode. Cellular data communication typically provides full-duplex communication, with a low to moderate bandwidth. PCS systems such as Bellcore's WACS [6] typically provide full-duplex services and data rates up to around 32 kbps. We omit further details of the architecture of the server and of the communication network.

Each broadcast period (see Fig. 1) consists of a *preamble*, a pattern of zero or more bits known to the clients and used for synchronization. The preamble is followed by a broadcast *period flag*, a pattern of bits 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 and a *trailer* indicating its end. Typically, the item header identifies

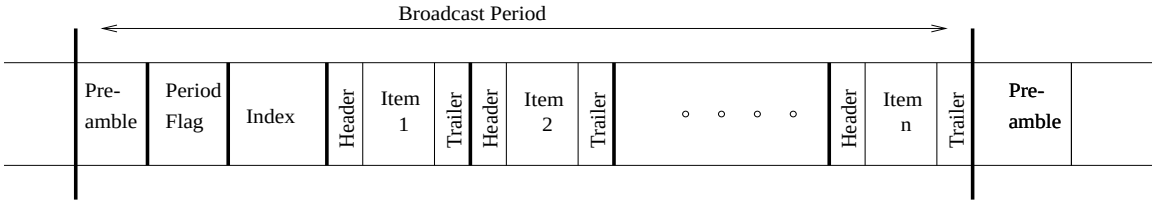


Figure 1: A general periodic wireless broadcast

the item (e.g., it specifies that the item contains current market information about gold prices) and may contain other information also (e.g. length of the item), while the trailer is simply a known pattern of bits marking the end of the item.

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.

In this paper we will consider situations where the length and sequence of data items is fixed, as well as those where it is not.

3 Simplex broadcast communication

In this section we consider the situation in which communication is simplex and the clients simply filter the data broadcast by the server. The server broadcasts all the information it has at the start of each broadcast period. Thus suppose the broadcast channel is capable of supporting C bits/sec, and the server has D_i bits of data to send during period i, for $i \geq 0$. Then broadcast period i takes $R_i = D_i/C$ seconds.

From the clients' point of view, the broadcast data can be modeled as an airdisk, with each period corresponding to an airdisk rotation, where the airdisk has a data transfer rate of C. Since there is only one track, the airdisk has a seek time $S = 0$. If the server sends a maximum of D_{max} bits during any period, the maximum rotational latency of the airdisk is $R_{max} = D_{max}/C$, since a client may have to wait up to R_{max} seconds for the next rotation if it just misses the data it needed in the current rotation.

Suppose the average amount of data in any rotation is D. Then the average length of a rotation is $R = D/C$ seconds. Suppose also that each data item has a header, which can be read by the client to determine if the data item is of interest, and the sequence of data items on the disk is fixed, but the length of each item can vary from rotation to rotation. It might seem that the average rotational latency for a client which is reading data from the disk is $R/2$, and in fact this value has

sometimes been used in previous work on periodic wireless data broadcast [14]. However, it can be shown that mean rotational latency depends not only upon the mean rotation length but also on its variance. (This argument has been described qualitatively in [1].)

Lemma 3.1 *Consider an airdisk where each item is broadcast once per rotation, the sequence of data items in every rotation is fixed, but the length of each item, and hence of each rotation, can vary. The mean rotational latency for a client accessing a given data item from such an airdisk, if the airdisk has mean rotation length R and variance s, is*

$$\frac{R}{2} + \frac{s^2}{2R}$$

proof. The proof follows directly from the "residual life paradox" (e.g. see [17].) We sketch the underlying intuition here. Assume that a client begins reading the airdisk at at an arbitrary point of time, and is interested in item x. On average, item x will appear every R seconds. However, some of the intervals between appearances of x are long, and some are short. Since the client begins reading the airdisk at an arbitrary point of time, it is statistically more likely that the client will begin reading during a long interval than during a short one. Thus the mean rotational latency experienced by the client will be more than $R/2$. (For the derivation of the formula, see [17].) Note that only if all the items were of the same length, and the rotation length were R for every rotation (i.e., zero variance), would the mean latency be $R/2$. □

The mean rotational latency affects not only the access time experienced by the user, but also affects the energy consumption at the client. If the mobile client has a limited energy source (e.g. standard battery cells), energy is a very precious resource and needs to be conserved [5, 14]. In the scenario just described, suppose the clients know the sequence of data items. If, in addition, the length of the data items were fixed, the variance of the rotation length would be zero and the mean rotational latency would be reduced to $R/2$. The smaller mean rotational latency will result in reduced energy consumption at the mobile. In addition, the fixed rotation length could be used to substantially reduce the mobile's energy consumption, as follows. Initially, the client reads the preamble and period flag of one rotation. It is then able to synchronize its internal clock to that of the server, so that for subsequent reads of the airdisk, it knows where on each rotation each item appears, and waits until a very short interval before that time to start reading the airdisk. Note that during the waiting interval, the client can slip into *doze mode*, as provided by most laptop and portable computers, resulting in substantial savings in energy consumption. Then although the mean rotational latency remains $R/2$, the energy consumed by the mobile will be much less. In order to correct for clock drifts at the server or client, the client can maintain synchronization by periodically re-reading the preamble or period flag.

In Lemma 3.1 it was assumed that the sequence of data items on the airdisk was fixed. If the sequence of data items can change from one rotation to the next, the calculation of mean rotational latency is more complicated. First, the client has to wait until the end of the current period; this

time interval equals that given in Lemma 3.1. The client then has to read the index and wait until the desired item appears on the airdisk; this second time interval depends upon the manner in which data items are scheduled for broadcast, i.e., for layout on the airdisk, at each rotation.

Note that for simplex data broadcast described in this section, the airdisk is a One-Writer, Many-Readers (OWMR) disk, where the single writer is the server. In such an airdisk the data layout on the disk may be changed by the server in response to the frequency with which the values of data items change. Thus the server may change the data layout in response to the pattern of data writes by the information provider. (This is in contrast to the situation considered in previous work [14, 8] where the data layout is changed in response to the pattern of data reads by the clients; we will consider this in the next section also.) For example, if a data item is changing rapidly the server may write it multiple times during a single rotation.

4 Changing data layout in response to client read patterns

We now consider the situation where the server has some information about the data items which clients are interested in. We have discussed schemes for obtaining this client interest information in [15]; 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. For example, it is likely that some items will be more popular (i.e., of interest to more clients) than others. Imielinski et al [14] and Chiueh [8] have referred to these popular items as hot spots, by analogy with hot spots in magnetic disks, and Chiueh has essentially proposed replicating hot spots on each airdisk rotation in order to minimize the mean access time. As Chiueh has pointed out, while replication can be very useful, it consumes more wireless bandwidth and, if not performed correctly, can actually result in increased mean access times.

In this paper we examine another method of exploiting client interest information to improve airdisk performance, which does not rely on replication (although possibly could be used in conjunction with it.) We consider two problems of laying out the data on the airdisk. We first consider the situation in which there is no index on the airdisk (as assumed in [8]), and then consider the situation where there is an index (as assumed in [14].) 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.

4.1 Non-indexed data layout

Suppose that the airdisk has no index but the sequence of data items can be varied at each rotation. Every item has a header identifying the item, and, for simplicity, assume that the items are all of the same length. A client then reads the airdisk, examining each item header to see if the item

is of interest. In general, a client will be interested in one or more items in each rotation, and a client's access is not considered completed unless it has read all the items it is interested in. As before, suppose a client starts reading the disk at an arbitrary instant of time. The access time for a client will be the sum of four components: (1) the rotational latency, i.e., waiting for the first item the client is interested in to appear, (2) the data transfer time to read the first item, (3) the *item spread*, i.e., the time waiting until the last item the client is interested in appears, and (4) the data transfer time to read the last item. Note that if the client is interested in more than two items, the data transfer time for those items is overlapped with the item spread and so need not be counted. Also note that the client may slip into doze mode between data items of interest to reduce energy consumption. We are interested in minimizing the mean access time over all clients.

The mean rotational latency and data transfer times are fixed once the set of items in a given rotation is decided. However, the item spread can still be varied by varying 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 spread and hence the mean access time.

We cast the problem of minimizing item spread 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. Thus the length of a hyperedge represents the item spread for that client. The non-indexed data layout problem then becomes the problem of numbering the vertices so as to minimize the mean of the hyperedge lengths.

Theorem 4.1 *The problem of minimizing the mean access time, over all clients, for the non-indexed data layout problem, is NP-complete, even if each client is interested in only two items.*

proof. As discussed, the mean access time is minimized by minimizing mean item spread. The problem of minimizing mean item spread can be rewritten as follows for the case where each client is interested in only two items. Given a graph $G = (V, E)$ with vertex set V and edges E, find a

one-to-one function $f : V \rightarrow \{1, 2, ..., |V|\}$ such that

$$s_1 = \sum_{(u,v) \in E} |f(u) - f(v)| \tag{1}$$

is minimized. This problem is identical to the known NP-complete problem Optimal Linear Arrangement [4]. □

The Non-indexed data layout problem has been studied extensively in other contexts (e.g. see [7]) and optimal as well as heuristic algorithms have been designed for it. We will not consider it further in this paper. Instead, we will consider the situation where the airdisk contains an index, which will allow the client to save power by simply reading the index, and then only reading the items of its interest. We define this in the following.

4.2 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 as for the non-indexed data layout problem, 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.

Once again, the problem reduces to minimizing one component, namely the item reach. We can express it in graph-theoretic terms using the formulation used for the non-indexed data layout, i.e., create a graph using the set of vertices and hyperedges as before. We can show that the item reach minimization problem is also NP-complete; however, the proof requires designing a polynomial-time reduction.

Theorem 4.2 *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.*

proof. Minimizing mean access time reduces to minimizing mean item reach. The problem of minimizing mean item reach can be rewritten as follows. Given a graph $G = (V, E)$ with vertex set

V and edges E, find a one-to-one function $f : V \rightarrow \{1, 2, ..., |V|\}$ such that

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

is minimized.

The proof is by a reduction from the problem Optimal Linear Arrangement (OLA). The proof is in three steps. We first define a weighted version of IDL, (weighted IDL, or WIDL) in which edges may have positive or negative integer weights, and show in Lemma 4.1 that the known NP-complete problem OLA can be reduced to this weighted layout problem. We then show in Lemma 4.2 that WIDL layout problem can be reduced to a version of WIDL with positive weights, problem PWIDL. Finally in Lemma 4.3 we show that PWIDL can be reduced to the target problem, IDL. □

In the following $G = (V, E)$ denotes a graph G with vertex set V and edge set E.

Def. The weighted indexed data layout problem (WIDL) is that given a graph $G = (V, E)$ and a weight function $w : E \rightarrow I$ where I is the set of integers, and a positive integer K, does there exist a one-to-one function $f : V \rightarrow \{1, 2, ..., |V|\}$ such that

$$s_3 = \sum_{(u,v) \in E} w((u, v)) * \max(f(u), f(v)) \leq K$$

Def. The Optimal Linear Arrangement problem (OLA) is that given a graph $G = (V, E)$ and a positive integer K, does there exist a one-to-one function $f : V \rightarrow \{1, 2, ..., |V|\}$ such that

$$s_1 = \sum_{(u,v) \in E} |f(u) - f(v))| \leq K$$

Lemma 4.1 *The OLA problem reduces to WIDL.*

proof. For an arbitrary instance of OLA, define a graph $G' = (V', E')$ and a positive integer K' such that:

$$V' = V$$

$$E' = E \cup \{(u, u) : u \in V\}$$

$$\forall\ u \in V, w(u, u) = -d(u)$$

$$\forall\ u, v \in V : u \neq v, w(u, v) = 2$$

Now let

$$K' = \sum_{(u,v) \in E'} w(u, v) * \max(f(u), f(v)) \tag{3}$$

$$
\begin{aligned}
&= 2 \sum_{(u,v) \in E} \max(f(u), f(v)) - \sum_{(u,u) \in E'} d(u) \max(f(u), f(u)) \\
&= 2 \sum_{(u,v) \in E} (\min(f(u), f(v)) + |f(v) - f(u)|) - \sum_{u \in V} d(u) f(u) \\
&= 2 \sum_{(u,v) \in E} (.5(f(u) + f(v)) + .5(|f(v) - f(u)|)) - \sum_{u \in V} d(u) f(u) \\
&= \sum_{(u,v) \in E} |f(v) - f(u)| + \sum_{(u,v) \in E} f(u) + f(v) - \sum_{u \in V} d(u) f(u) \\
&= \sum_{(u,v) \in E} |f(v) - f(u)| \\
&= K
\end{aligned} \tag{4}
$$

The construction can clearly be performed in polynomial time. □

Def. The Positive-weighted indexed data layout problem (PWIDL) is identical to WIDL except that the weight function maps the edges to non-negative integers instead of the set of integers.

Lemma 4.2 *The WIDL problem reduces to PWIDL.*

proof. Given an arbitrary instance of WIDL, construct an instance of PWIDL in polynomial time as follows. Let

$$\forall\ u \in V, w(u,u) = |E| - d(u)$$

Then,

$$
\begin{aligned}
K' &= 2 \sum_{(u,v) \in E} \max(f(u), f(v)) - \sum_{u \in V} d(u) f(u) + \sum_{u \in V} |E| f(u) \\
&= \sum_{(u,v) \in E} |f(u) - f(v)| + |E| \sum_{i \in \{1,\ldots,|V|\}} i \\
&= K + \frac{|E||V|(|V| - 1)}{2}
\end{aligned} \tag{5}
$$

□

Note that in the reduction from WIDL to PWIDL, it suffices to set $w(u,u)$ to some value which will be positive for all $u \in V$; thus for instance, one could choose $w(u,u) = \Delta - d(u) + 1$, where Δ is the degree of the graph in the instance of WIDL.

Finally, we will show that PWIDL can be reduced to IDL.

Lemma 4.3 *The OLA problem reduces to IDL.*

proof. From Lemmas 4.1 and 4.2 we know that an arbitrary instance of OLA can be transformed into an arbitrary instance of PWIDL in polynomial time. The reduction from PWIDL to IDL can be done simply by replacing any edge of weight w in an instance of PWIDL by w edges of unit weight in IDL. This is also clearly a polynomial time reduction. □

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.

Several variations and generalizations of these problems can be considered. The generalization where data items are not of the same length may make the problems significantly more complicated. It may also be possible to allow clients to specify not only their interest in a data item, but also their degree of interest. In the graphical formulation, this corresponds to assigning weights to the vertices of the graph.

The indexed data layout problem is of interest as it corresponds to situations of practical applicability, e.g. for delivery of stocks or traffic information [21]. In the following section we describe methods for solving the problem optimally as well as by efficient suboptimal heuristics.

5 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: a vertex corresponds to a data item, and a (hyper)edge corresponds to the (two or more) data items of interest to a given client. Multiple (hyper)edges correspond to multiple clients with identical items of interest, and can be represented by a single (hyper)edge with weight equal to the number of multiple edges.

The naive algorithm for solving the indexed data layout problem is by exhaustive enumeration. For n vertices (items of data), there are obviously $n!$ permutations which need to be examined. For each permutation of the vertices, the item reach for each client can be computed given the vertices in which it is interested, and hence the mean item reach over all clients can be calculated. It is clear that the execution time for exhaustive enumeration will be prohibitively large for all but small n.

We develop an optimal algorithm for this problem using a branch-and-bound approach. The branch-and-bound method is well-known [19] and we only summarize it's application to our situation. The algorithm (which we call B&B) begins with a candidate (generally, non-optimal) solution for the problem. In our case, for instance, it may simply be a random permutation of the vertices. The B&B algorithm essentially consists of traversing a tree, where the root node of the tree (level 0) is the situation in which no vertices have been decided in the permutation. (Note that we use the term "node" to refer to the nodes in the branch-and-bound search tree; while the term "vertices"

refers to the vertices which represent the data items to be scheduled.) Each leaf node of the tree corresponds to a complete permutation of all n vertices, and each interior node at level i, $0 < i < n$, corresponds to the situation in which the positions of $i-1$ vertices have been decided. At each node in the branch-and-bound tree, the B&B algorithm computes a lower bound on the value of the objective function for all the leaves in the subtree rooted at that node; only if this lower bound is less than the candidate solution does the algorithm proceed further down this path, i.e., investigate the vertex permutations represented by the nodes in the subtree. If the lower bound exceeds the current candidate solution, the subtree is not considered further. If a leaf node for which the objective function is less than the candidate solution is found, it becomes the current candidate; the search continues until all leaves have either been eliminated or evaluated in this fashion.

Clearly, the better the lower bound used in the B&B algorithm, the more branches of the tree will be eliminated. Thus the execution efficiency of B&B depends critically upon how good the lower bound is. In the following we present a lower bound on the mean item reach, and prove its correctness (i.e., it is indeed a lower bound.) We will use the following observations.

Def. For a permutation f of the vertices of G, where f is a one-to-one function $f : V \rightarrow \{1, 2, ..., |V|\}$, 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 $G = (V, E)$,

$$s_2 = \sum_{(u,v) \in E} \max(f(u), f(v)) \tag{6}$$

$$= \sum_{v \in V} f(v) ld(v) \tag{7}$$

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. 7 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 5.1 *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. 2 for the remaining vertices is given by

$$L_1 = \sum_{j=i+1}^{n} j * \max(0, \min(|E| - |E_j|, d(S(j-i)))) \tag{8}$$

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

proof. After i vertices have been placed, $|E| - |E_i|$ edges remain which need to have one or both endpoints placed. To minimize s_2, these edges should be completed, i.e., have both endpoints placed, as early in the vertex permutation as possible. In the best case, for every vertex from $i+1$ onwards, all its incident edges are completed at that vertex, i.e. it has its left degree equal to its degree. From Eq. 7, the vertices from $i+1$ onwards should be arranged in order of descending degree.

The *max* term of L_1 ensures that the calculation of L_1 stops when all the remaining edges have been completed. The *min* term ensures that at the last vertex for which edges remain, only the remaining number of edges, and not the vertex's degree, is used in the calculation; otherwise, the formula calculates an estimate based upon Eq. 7 assuming that the left degree of each remaining vertex equals its degree. □

We studied the effectiveness of the B&B algorithm with the lower bound of Lemma 5.1 by implementing it as a C program and running simulation experiments in which it was presented with random graphs as input. For the sake of comparison, and to check the correctness of the implementation, an exhaustive enumeration algorithm was also implemented as a C program and run on the same inputs. B&B was very significantly faster than exhaustive enumeration, even for a broadcast period with only 8 items ($n = 8$); for larger n, B&B's speed advantage increases even more. Since this is as expected, we omit details of these experimental results in this paper.

6 A fast heuristic for data layout

Although the B&B algorithm is much faster than exhaustive enumeration, 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. The B&B algorithm is henceforth used only for the purposes of comparing its performance with the heuristic; it is expected that in any practical situation the heuristic would be preferable.

To motivate the first heuristic, consider again the special case we mentioned in sec. 4, 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. Recall from our observation in Eq. 7 that the objective can be written as $s_2 = \sum v \in V f(v) ld(v)$, where $ld(v)$ is the left-degree of vertex v under the permutation f. Thus 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 $i = 1, ..., n-1$: if the left degree of vertex $i+1$ exceeds that of vertex i, the positions of the two vertices are interchanged.

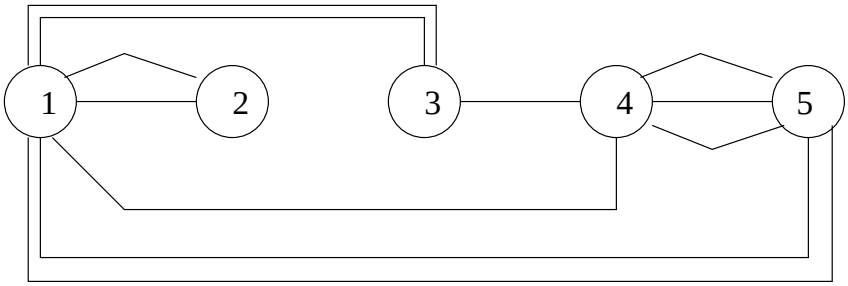
MAX-LD will also not always produce an optimal ordering. For example, considering the instance G shown in Fig. 2, MAX-LD will obtain the vertex ordering (1, 5, 4, 3, 2). We leave it to the reader to verify that this ordering has a cost of 38, which, while better than the cost obtained by MAX, is still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

6.1 Experimental results

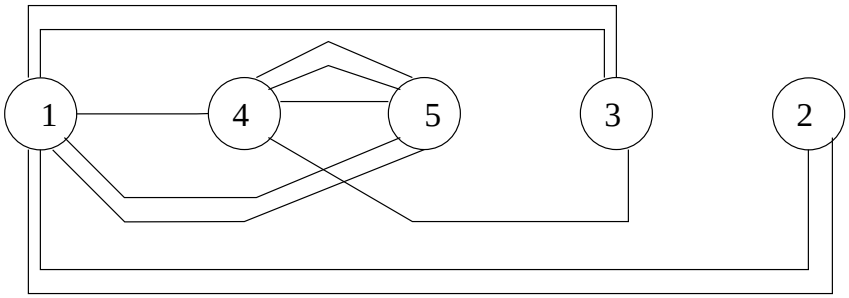
We show preliminary experimental results from a set of experiments which compare the performance of MAX-LD with B&B.

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.



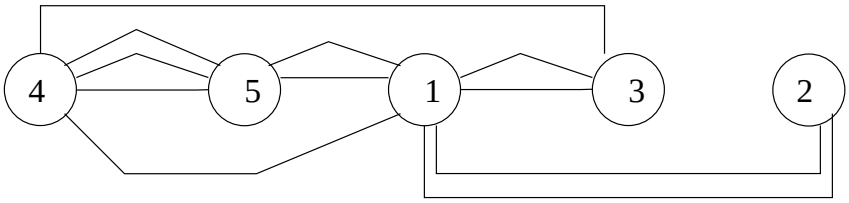
(a) Original graph G

degrees: 7 2 3 5 5
left-degrees: 0 2 2 2 5 s = 43



(b) A layout by MAX

degrees: 7 5 5 3 2
left-degrees: 0 1 5 3 2 s = 39



(c) An optimal layout

degrees: 5 5 7 3 2
left-degrees: 0 3 3 3 2 s = 37

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.

Typically, the CPU execution time of the heuristic was of the same order as of the granularity of time measurement (16.66 ms). To obtain an accurate estimate of CPU time, the heuristic was repeated a number of times for each input graph, so that the time measured was of the order of minutes; the execution time for a single run of the heuristic for that input graph was then calculated by dividing this measured time by the number of repetitions of the heuristic.

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 described above, 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 $n = 5, 8, 10, 11$. Because of the very large difference in CPU times, note that a log-scale is used for the y-axis. 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 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 theoretical running time of $O(m + n \log n)$, where m is the number of edges, becomes $O(n^2 + n \log n) = O(n^2)$ in this case. Thus the measured values for execution time are consistent with the theoretical analysis.

7 Discussion and related work

The simulation experiments shown above have been extended to consider the case that the edges are weighted (i.e., there are parallel edges in the graph). The results are qualitatively similar to

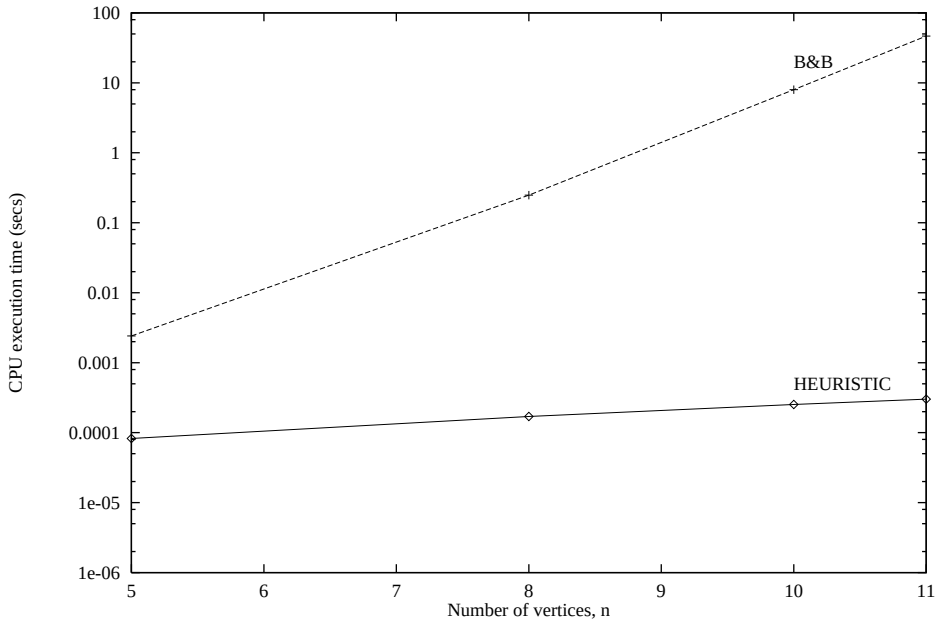


Figure 3: Comparison of mean execution time of B&B and the MAX-LD heuristic for graphs with unit-weight edges and edge density 0.5. *Note log scale on y-axis.* Each data point is the mean for 25 random input graphs. The comparison of MAX-LD with B&B cannot be continued for much beyond $n = 11$ as B&B rapidly becomes prohibitively time consuming to execute.

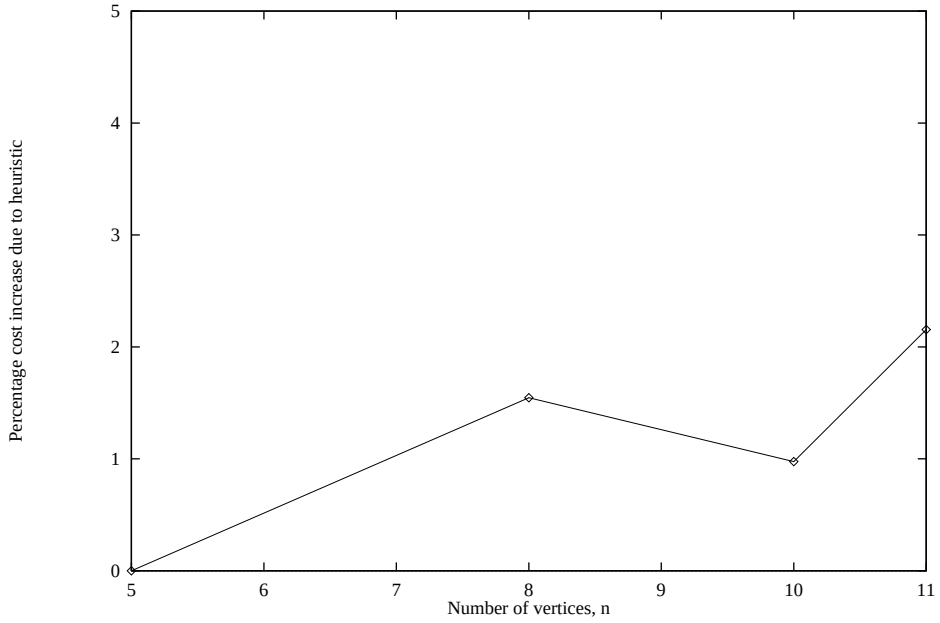


Figure 4: Comparison of mean percentage increase in cost s_2 when the heuristic MAX-LD is used, compared to the optimal solution. (Graphs with unit-weight edges and edge density 0.5.)

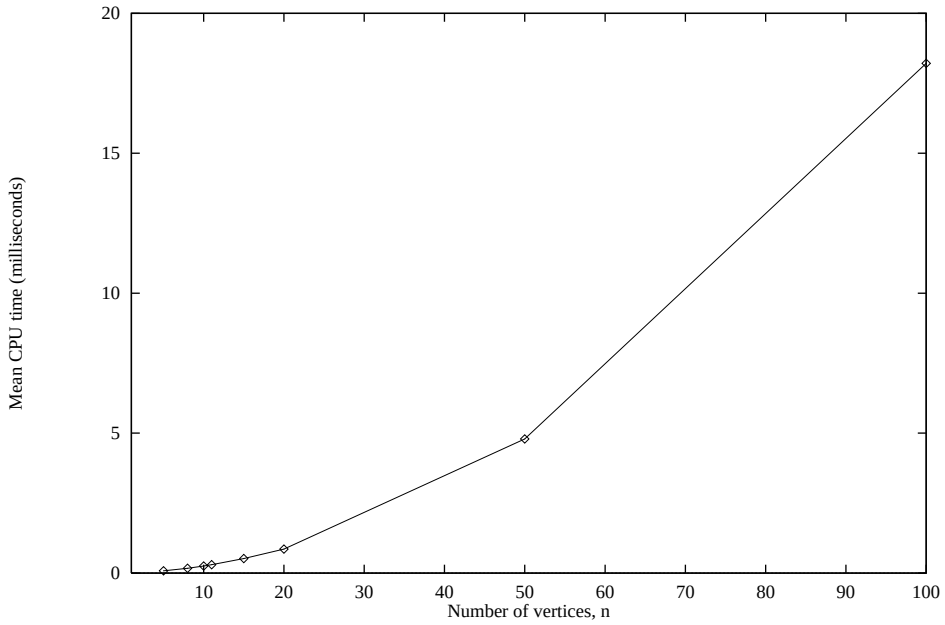


Figure 5: Mean CPU time for MAX-LD heuristic for graphs with unit-weight edges and edge density 0.5.

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 [15]. 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 [20, 13] which is in successful and widespread use. We call this design the *AirRAID*, and we have described it briefly in [15].

There has been a surge of related work on periodic wireless broadcast recently. As mentioned previously, the airdisk model [15] 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 (the second issue deals with how an individual client should best manage its cache, and is not directly related to the present paper.)

Thus [1] considers how data items should be replicated in the broadcast so as to minimize mean access times, with the most popular items being broadcast most often. However, [1] does not explicitly consider the ordering of individual data items within the broadcast period. In contrast, we consider how, once the data items to be broadcast have been selected, they are to be ordered so as to minimize the total access time for each client (averaged over all clients), where clients may be interested in one or more data items.

Other work on broadcast disks [2, 3] has focused on policies by which clients may prefetch data, and is not directly related to the work in the present paper. Some of the recent work in the area of periodic broadcast has considered alternatives to indexing for data retrieval from the broadcast. In [18] the authors present single and multi-level signature schemes, as well as schemes for caching the signatures at the mobile clients.

8 Conclusions

We have previously presented a model, called the airdisk model, for representing periodic wireless data broadcast in terms of a magnetic disk [15, 16]. In this paper we have shown expressions for its mean and maximum rotational latency under certain assumptions. We have then defined two problems of laying out the data on the airdisk based upon information about which items are of interest to clients. We have shown that both problems, if the data is to be laid out so as to minimize mean client access time, are NP-complete.

For the data layout problem in which an index is broadcast with the data, we have developed an optimal algorithm using the branch-and-bound method, and also a fast, simple heuristic. We have carried out a set of simulation experiments to evaluate the behavior of these algorithms and found that the heuristic runs much faster than the branch-and-bound algorithm, yet, for these experiments, produces orderings whose cost is within a few percent of the optimal cost.

We are currently continuing the work reported here in two ways. The first is to expand the experimental and theoretical results presented here for the MAX-LD and related heuristics. The second is implement the airdisk model in a server connected to mobile clients connected by an experimental wireless LAN. We are also continuing to investigate other directions [15], such as the use of multiple parallel airdisks to increase airdisk storage and performance via the concept of airRAID.

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] D. Adolphson and T. C. Hu. Optimal linear ordering. *SIAM J. Appl. Math.*, 25:403–423, 1973.

[5] B. R. Badrinath and T. Imielinski. Data management issues in mobile computing. In *Wireless Datacomm '92*, 1992.

[6] Bellcore. Generic criteria for Version 0.1 Wireless Access Communications Systems (WACS). Technical Advisory TA-NWT-001313, Issue 1, Bellcore, July 1992.

[7] J. Bhasker and S. Sahni. Optimal linear arrangement of circuit components. *Journal of VLSI and Comp. Sys.*, 2(1-2):87–109, 1987.

[8] T. Chiueh. Scheduling for broadcast-based file systems. In *Proc. MOBIDATA Workshop*. Rutgers University, Nov. 1994.

[9] Juan Miguel del Rosario, R. Bordawekar, and Alok Chaudhary. Improved parallel I/O via a two-phase run-time access strategy. In *Proc. Workshop on I/O in Parallel Computer Systems*, pages 56–70, 1993. Also in ACM SIGARCH Comp. Arch. News., Dec. 1993.

[10] G. Herman et al. The Datacycle architecture for very large high throughput database systems. In *Proc. SIGMOD*, pages 97–103, 1987.

[11] T. F. Bowen et al. The Datacycle architecture. *Comm. ACM*, pages 71–81, Dec. 1992.

[12] M. Garey and D. Johnson. *Computers and intractability: A guide to the theory of NP-completeness.* Freeman, 1979.

[13] J. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann, San Mateo, CA, 1990.

[14] T. Imielinski, S. Viswanathan, and B.R. Badrinath. Energy efficient indexing on air. In *Proc. SIGMOD*, pages 25–36, 1994.

[15] Ravi Jain and John Werth. Airdisks and AirRAID: Modeling and scheduling periodic wireless data broadcast. DIMACS Tech. Report 95-11, Rutgers Univ., May 1995.

[16] Ravi Jain and John Werth. Airdisks and AirRAID: Modeling and scheduling periodic wireless data broadcast. *ACM SIGARCH Comp. Arch. News.*, Oct. 1995.

[17] Leonard Kleinrock. *Queuing Systems Volume I: Theory.* Wiley, 1975.

[18] 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.).

[19] C. H. Papadimitriou and K. Stieglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice Hall, 1982.

[20] 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.

[21] 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.

[22] A. Tanenbaum. *Computer Networks.* Prentice-Hall, 1988.

[23] G. Varrall. *Data Over Radio.* Quantum Publishing, 1992.