Scheduling Data Broadcasts on Wireless Channels: Exact Solutions and Heuristics *†

Alan A. Bertossi ‡ M. Cristina Pinotti § Romeo Rizzi¶

1 Introduction

In wireless asymmetric communication, broadcasting is an efficient way of simultaneously disseminating data to a large number of clients. Consider data services on cellular networks, such as stock quotes, weather infos, traffic news, where data are continuously broadcast to clients that may desire them at any instant of time. In this scenario, a server at the base-station repeatedly transmits data items from a given set over a wireless channel, while clients passively listen to the shared channel waiting for their desired item. The server follows a broadcast schedule for deciding which item of the set has to be transmitted at any time instant. An efficient broadcast schedule minimizes the client expected delay, that is, the average amount of time spent by a client before receiving the item he needs. The client expected delay increases with the size of the set of the data items to be transmitted by the server. Indeed, the client has to wait for many unwanted data before receiving his own data. The efficiency can be improved by augmenting the server bandwidth, for example, allowing the server to transmit over multiple disjoint physical channels and therefore defining a shorter schedule for each single channel. In a multi-channel environment, in addition to a broadcast schedule for each single channel, an allocation strategy has to be pursued so as to assign data items to channels. Moreover, each client can access either only a single channel or any available channel at a time. In the former case, if the client can access only one prefixed channel and can potentially retrieve any available data, then all data items must be replicated over all channels. Otherwise, data can be partitioned among the channels, thus assigning each item to only one channel. In this latter case, the efficiency can be improved by adding an index that informs

*Portions of this chapter reprinted, with permission, from [5] *IEEECS Log Number TC-0238-0704*, © 2005, IEEE.

†This work has been supported by ISTI-CNR under the BREW research grant.

‡Department of Computer Science, University of Bologna, 40127 Bologna, Italy, E-mail: bertossi@cs.unibo.it

§Department of Computer Science and Mathematics, University of Perugia, 06123 Perugia, Italy, E-mail: pinotti@unipg.it

¶Department of Comp. Sci. and Telec., University of Trento, 38050 Povo, Trento, Italy, E-mail: rrizzi@science.unitn.it

the client at which time and on which channel the desired item will be transmitted. In this way, the mobile client can save battery energy and reduce the tuning time because, after reading the index info, it can sleep and wake up on the proper channel just before the transmission of the desired item.

Several variants for the problem of data allocation and broadcast scheduling have been proposed in the literature, which depend on the perspectives faced by the research communities [2, 3, 6, 8, 9, 10, 12, 13, 15, 16, 17].

Specifically, the networking community faces a version of the problem, known as the *Broadcast Problem*, whose goal is to find an infinite schedule on a single channel [15, 6, 9, 10]. Such a problem was first introduced in the teletext systems by [2, 3]. Although it is widely studied (e.g., it can be modeled as a special case of the Maintenance Scheduling Problem and the Multi-Item Replenishment Problem [6, 9]), its tractability is still under consideration. Therefore, the emphasis is on finding near optimal schedules for a single channel. Almost all the proposed solutions follow the *square root rule (SRR)* [3]. The aim of such a rule is to produce a broadcast schedule where each data item appears with equally spaced replicas, whose frequency is proportional to the square root of its popularity and inversely proportional to the square root of its length. The multi-channel schedule is obtained by distributing in a round robin fashion the schedule for a single channel [15]. Since each item appears in multiple replicas which, in practice, are not equally spaced, these solutions make indexing techniques not effective. Briefly, the main results known in the literature for the Broadcast Problem can be summarized as follows. For *uniform* lengths, namely all items of the same length, it is still unknown whether the problem can be solved in polynomial time or not. For a constant number of channels, the best algorithm proposed so far is the Polynomial Time Approximation Scheme (PTAS) devised in [10]. In contrast, for *non-uniform* lengths, the problem has been shown to be strong NP-hard even for a single channel, a 3-approximation algorithm was devised for one channel, and a heuristic has been proposed for multiple channels [9].

On the other hand, the database community seeks for a periodic broadcast scheduling which should be easily indexed [8]. For the single channel, the obvious schedule that admits index is the *flat* one. It consists in selecting an order among the data items, and then transmitting them once at a time, in a round-robin fashion [1], producing an infinite periodic schedule. In a flat schedule indexing is trivial, since each item will

appear once, and exactly at the same relative time, within each period. Although indexing allows the client to sleep and save battery energy, the client expected delay is half of the schedule period and can become infeasible for a large period. To decrease the client expected delay, still preserving indexing, flat schedules on multiple channels can be adopted [12, 13, 17]. However, in such a case the allocation of data to channels becomes critical. For example, allocating items in a balanced way simply scales the expected delay by a factor equal to the number of channels. To overcome this drawback, *skewed* allocations have been proposed where items are partitioned according to their popularities so that the most requested items appear in a channel with shorter period [12, 17]. Hence, the resulting problem is slightly different from the Broadcast Problem since, in order to minimize the client expected delay, it assumes skewed allocation and flat scheduling. This variant of the problem is easier than the Broadcast Problem. Indeed, as proved in [17], the optimal solution for uniform lengths can be found in polynomial time. In contrast, the problem becomes computationally intractable for non-uniform lengths [5]. For this latter case, several heuristics have been developed in [17, 4], which have been tested on some benchmarks whose popularities follow Zipf distributions. Such distributions are used to characterize the popularity of one element among a set of similar data, like a web page in a web site [7].

The present chapter reviews the work of [17, 5, 4] on the broadcasting problem of N data items over K wireless channels, under the assumptions of skewed data allocation to channels and flat data scheduling per channel. Both the uniform and non-uniform length cases are surveyed showing their exact and heuristic solutions, respectively.

For the case of data items with uniform lengths, three exact polynomial time algorithms are presented, all based on dynamic programming. The first algorithm, called *DP* and originally proposed in [17], takes $O(N^2K)$ time, while the second algorithm, called *Dichotomic* and proposed later in [5], is faster as it runs in $O(NK \log N)$ time. The third algorithm, presented in [4], is designed for the specific case of $K = 2$. Although it requires $O(N \log N)$ time, and hence it is asymptotically not faster than Dichotomic, it exploits a specific characterization of the optimal solution when there are only two channels.

For the case of data items with non-uniform lengths, the problem is NP-hard when $K = 2$, and *strong* NP-hard for arbitrary K. In this latter case, the *Optimal* algorithm presented in [5] is reviewed. It requires

$O(KN^{2z})$ time, where z is the maximum data length, and reduces to the DP algorithm when $z = 1$. Since algorithm Optimal can solve only small instances in a reasonable time, three heuristics are described, all having an $O(N(K + \log N))$ time complexity.

The first heuristic, called *Greedy*, has been proposed in [17]. Fixed N, Greedy starts with all data items assigned to one channel, and then proceeds by splitting the items of one channel between two channels, thus adding a new channel, until K channels are reached. The other two heuristics, both presented in [4], pretend that the characterization of the optimal solution of the problem for $K = 2$ and uniform lengths holds also for the general case of arbitrary K and non-uniform lengths. One heuristic is called *Greedy+* since combines such solution characterization with the Greedy approach, while the second heuristic is called *Dlinear* and combines the same characterization with the dynamic programming relation proposed in [17].

All the three heuristics are then tested on benchmarks whose popularities are characterized by Zipf distributions. The experimental tests reveal that Dlinear finds optimal solutions almost always, requiring reasonable running times. Although Greedy remains the fastest heuristic, it gives the worst sub-optimal solutions. Both the running times and the quality of the solutions of Greedy+ are intermediate between those of Dlinear and Greedy. However, Greedy and Greedy+ have the feature to scale well with respect to the parameter changes.

The rest of this chapter is so organized. Section 2 gives notations, definitions and the problem statement. Section 3 illustrates the DP and Dichotomic algorithms for items of uniform lengths, as well as the solution characterization for the particular case of two channels. In contrast, Section 4 studies the non-uniform length case. It first recalls the strong NP-hardness for an arbitrary number of channels, and then presents the exponential time Optimal algorithm. Then, Section 5 gives the Greedy, Greedy+, and Dlinear heuristics. Section 6 reports the experimental tests on some benchmarks, whose popularities follow Zipf distributions. Finally, conclusions are offered in Section 7.

2 Problem Formulation

Consider a set of K identical channels, and a set $D = \{d_1, d_2, \ldots, d_N\}$ of N data items. Each item d_i is characterized by a *probability* p_i and a *length* z_i, with $1 \leq i \leq N$. The probability p_i represents the popularity

of item d_i, namely its probability to be requested by the clients, and it does not vary along the time. Clearly, $\sum_{i=1}^{N} p_i = 1$. The length z_i is an integer number, counting how many time units are required to transmit item d_i on any channel. When all data lengths are the same, i.e. $z_i = z$ for $1 \leq i \leq N$, the lengths are called *uniform* and are assumed to be unit, i.e. $z = 1$. When the data lengths are not the same, the lengths are said *non-uniform*.

The items have to be partitioned into K groups $G_1, \ldots, G_K$. Group G_j collects the data items assigned to channel j, with $1 \leq j \leq K$. The cardinality of G_j is denoted by N_j, the sum of its item lengths is denoted by Z_j, i.e. $Z_j = \sum_{d_i \in G_j} z_i$, and the sum of its probabilities is denoted by P_j, i.e. $P_j = \sum_{d_i \in G_j} p_i$. Note that since the items in G_j are cyclically broadcast according to a flat schedule, Z_j is the schedule period on channel j. Clearly, in the uniform case $Z_j = N_j$, for $1 \leq j \leq K$. If item d_i is assigned to channel j, and assuming that clients can start to listen at any instant of time with the same probability, the *client expected delay* for receiving item d_i is half of the period, namely $\frac{Z_j}{2}$. Assuming that indexing allows clients to know in advance the content of the channels [17], the *average expected delay* (AED) over all channels is

$$\text{AED} = \frac{1}{2} \sum_{j=1}^{K} Z_j P_j \tag{1.1}$$

Given K channels, a set D of N items, where each data item d_i comes along with its probability p_i and its integer length z_i, the *K-Non-Uniform Allocation Problem* consists in partitioning D into K groups $G_1, \ldots, G_K$, so as to minimize the objective function AED given in Equation 1.1. In the special case of equal lengths, the above problem is called *K-Uniform Allocation Problem* and the corresponding objective function is derived replacing Z_j with N_j in Equation 1.1.

As an example, consider a set of $N = 6$ items with uniform lengths and $K = 3$ channels. Let the demand probabilities be $p_1 = 0.37$, $p_2 = 0.25$, $p_3 = 0.18$, $p_4 = 0.11$, $p_5 = 0.05$ and $p_6 = 0.04$. The optimal solution assigns item d_1 to the first channel, items d_2 and d_3 to the second channel, and the remaining items to the third channel. The corresponding AED is $\frac{1}{2}(0.37 + 2(0.25 + 0.18) + 3(0.11 + 0.05 + 0.04)) = 0.915$.

Consider the sequence $d_1, \ldots, d_N$ of items ordered by their indices, and assume that each channel contains items with consecutive indices. Then, the cost of assigning to a single channel consecutive items within the sequence, say from d_i to d_j, is $C_{i,j} = \frac{1}{2} \left(\sum_{h=i}^{j} p_h \right) \left(\sum_{h=i}^{j} z_h \right)$. Letting $P_{i,j} = \sum_{h=i}^{j} p_h$ and $Z_{i,j} = \sum_{h=i}^{j} z_h$, one notes that all the $P_{1,n}$ and $Z_{1,n}$, for $1 \leq n \leq N$, can be computed in $O(N)$ time by two prefix

sum computations. Hence, a single $C_{i,j}$ can be computed on the fly in constant time as $C_{i,j} = \frac{1}{2}(P_{1,j} - P_{1,i-1})(Z_{1,j} - Z_{1,i-1})$. From now on, in order to simplify the presentation, $C_{i,j}$ is defined to be 0 whenever $i > j$. Note that, for uniform lengths, the formula of $C_{i,j}$ simplifies as $C_{i,j} = \frac{1}{2}(j - i + 1)\sum_{h=i}^{j} p_h$.

Moreover, a *segmentation* is a partition of the ordered sequence $d_1, \ldots, d_N$ into $G_1, \ldots, G_K$, such that if $d_i \in G_k$ and $d_j \in G_k$ then $d_h \in G_k$ whenever $i \leq h \leq j$. A segmentation

$$\underbrace{d_1, \ldots, d_{B_1}}_{G_1}, \underbrace{d_{B_1+1}, \ldots, d_{B_2}}_{G_2}, \ldots, \underbrace{d_{B_{K-1}+1}, \ldots, d_N}_{G_K}$$

is compactly denoted by the $(K-1)$-tuple

$$(B_1, B_2, \ldots, B_{K-1})$$

of its *right borders*, where border B_k is the index of the last item that belongs to group G_k. Notice that it is not necessary to specify B_K, the index of the last item of the last group, because its value will be N for any solution. From now on, B_{K-1} will be referred to as the *final border* of the solution. The cardinality of G_k, i.e. the number N_k of items in the group, is $N_k = B_k - B_{k-1}$, where $B_0 = 0$ and $B_K = N$ are assumed.

3 Uniform Lengths

This section is devoted to take a look to the dynamic programming algorithms proposed in [17, 5, 4] for the K-Uniform Allocation Problem. The following results show that the optimal solutions for the K-Uniform Allocation Problem can be sought within the class of segmentations.

Lemma 1.1. [17] *Let G_h and G_j be two groups in an optimal solution for the K-Uniform Allocation Problem. Let d_i and d_k be items with $d_i \in G_h$ and $d_k \in G_j$. If $N_h < N_j$, then $p_i \geq p_k$. Similarly, if $p_i > p_k$, then $N_h \leq N_j$.*

In other words, the most popular items are allocated to less loaded channels so that they appear more frequently. As a consequence, if the items are sorted by non-increasing probabilities, then the group sizes are non-decreasing.

Corollary 1.1. [17] *Let $d_1, d_2, \ldots, d_N$ be N uniform length items with $p_i \geq p_k$ whenever $i < k$. Then,*

there exists an optimal solution for partitioning them into K groups $G_1, \ldots, G_K$, where each group is made of consecutive elements.

Thus, assuming the items sorted by non-increasing probabilities, any sought solution S will be a segmentation. Moreover, the sought segmentations $S = (B_1, B_2, \ldots, B_{K-1})$ for the uniform case can be restricted to those verifying $N_1 \leq N_2 \leq \ldots \leq N_K$, which will be called *feasible* segmentations.

3.1 The DP Algorithm

To describe the DP algorithm [17], let $OPT_{n,k}$ denote an optimal solution for grouping items $d_1, \ldots d_n$ into k groups and let $opt_{n,k}$ be its corresponding cost, for any $n \leq N$ and $k \leq K$. Since $d_1, d_2, \ldots, d_N$ are sorted by non-increasing probabilities, one has:

$$opt_{n,k} = \begin{cases} C_{1,n} & \text{if } k = 1 \\ \min_{1 \leq \ell \leq n-1} \{opt_{\ell,k-1} + C_{\ell+1,n}\} & \text{if } k > 1 \end{cases} \tag{1.2}$$

The DP algorithm is a dynamic programming implementation of Recurrence 1.2. Indeed, in order to find $OPT_{n,k}$, consider the $K \times N$ matrix M with $M_{k,n} = opt_{n,k}$. The entries of M are computed row by row applying Recurrence 1.2. Clearly, $M_{K,N}$ contains the cost of an optimal solution for the K-Uniform Allocation Problem. In order to actually construct an optimal partition, a second matrix F is employed to keep track of the final borders of segmentations corresponding to entries of M. In Recurrence 1.2, the value of ℓ which minimizes the right-hand-side is the *final border* for the solution $OPT_{n,k}$ and is stored in $F_{k,n}$. Hence, the optimal segmentation is given by $OPT_{N,K} = (B_1, B_2, \ldots, B_{K-1})$ where, starting from $B_K = N$, the value of B_k is equal to $F_{k+1,B_{k+1}}$, for $k = K-1, \ldots, 1$.

To evaluate the time complexity of the DP algorithm, observe that $O(n)$ comparisons are required to fill the entry $M_{k,n}$, which implies that $O(N^2)$ comparisons are required to fill a row. Since there are K rows, the complexity of the DP algorithm is $O(N^2K)$.

3.2 The Dichotomic Algorithm

To improve on the time complexity of the DP algorithm for the K-Uniform Allocation Problem, the properties of optimal solutions have to be further exploited.

Definition 1.1. *Let $d_1, d_2, \ldots, d_N$ be uniform length items sorted by non-increasing probabilities. An optimal solution $OPT_{N,K} = (B_1, B_2, \ldots, B_{K-1})$ is called* left-most optimal *and denoted by $LMO_{N,K}$ if, for any other optimal solution $(B'_1, B'_2, \ldots, B'_{K-1})$, it holds $B_{K-1} \leq B'_{K-1}$.*

Clearly, since the problem always admits an optimal solution, there is always a left-most optimal solution. Although the left-most optimal solutions do not need to be unique, it is easy to check that there exists a unique $(B_1, B_2, \ldots, B_{K-1})$ such that $(B_1, B_2, \ldots, B_i)$ is a left-most optimal solution for partitioning into $i+1$ groups the items $d_1, d_2, \ldots, d_{B_{i+1}}$, for every $i < K$.

Definition 1.2. *A left-most optimal solution $(B_1, B_2, \ldots, B_{K-1})$ for the K-Uniform Allocation Problem is called* strict left-most optimal solution*, and denoted by $SLMO_{N,K}$, if $(B_1, B_2, \ldots, B_i)$ is a $LMO_{B_{i+1},i+1}$, for every $i < K$.*

The Dichotomic algorithm computes a left-most optimal solution for every $i < K$, and thus it finds the unique strict left-most optimal solution.

Lemma 1.2. [5] *Let $d_1, d_2, \ldots, d_N$ be uniform length items sorted by non-increasing probabilities. Let $LMO_{N-1,K} = (B_1, B_2, \ldots, B_{K-1})$ and $OPT_{N,K} = (B'_1, B'_2, \ldots, B'_{K-1})$. Then, $B'_{K-1} \geq B_{K-1}$.*

In words, Lemma 1.2 implies that, given the items sorted by non-increasing probabilities, if one builds an optimal solution for N items from an optimal solution for $N-1$ items, then the final border B_{K-1} can only move to the right. Such a property can be easily generalized as follows to problems of increasing sizes. From now on, let B^c_h denote the h-th border of $LMO_{c,k}$, with $k > h \geq 1$.

Corollary 1.2. [5] *Let $d_1, d_2, \ldots, d_N$ be uniform length items sorted by non-increasing probabilities, and let $l < j < r \leq N$. Then, $B^l_{K-1} \leq B^j_{K-1} \leq B^r_{K-1}$.*

Corollary 1.2 plays a fundamental role in speeding up the DP algorithm. Indeed, assume that $LMO_{n,k-1}$ has been found for every $1 \leq n \leq N$. If the $LMO_{l,k}$ and $LMO_{r,k}$ solutions are also known for some $1 \leq l \leq r \leq N$, then one knows that B^j_{k-1} is between B^l_{k-1} and B^r_{k-1}, for any $l \leq j \leq r$. Thus, Recurrence 1.2 can be rewritten as:

$$opt_{j,k} = \min_{B^l_{k-1} \leq \ell \leq B^r_{k-1}} \{opt_{\ell,k-1} + C_{\ell+1,j}\} \tag{1.3}$$

As the name suggests, the $O(KN \log N)$ time Dichotomic algorithm is derived by choosing $j = \lceil \frac{l+r}{2} \rceil$ in Recurrence 1.3, thus obtaining:

$$opt_{\lceil \frac{l+r}{2} \rceil, k} = \min_{B^l_{k-1} \leq \ell \leq B^r_{k-1}} \{opt_{\ell,k-1} + C_{\ell+1, \lceil \frac{l+r}{2} \rceil}\} \tag{1.4}$$

where B^l_{k-1} and B^r_{k-1} are, respectively, the final borders of $LMO_{l,k}$ and $LMO_{r,k}$. Such recurrence is iteratively solved within three nested loops which vary, respectively, in the ranges $1 \leq k \leq K$, $1 \leq t \leq \lceil \log N \rceil$, and $1 \leq i \leq 2^{t-1}$, and where the indices l, r, and j are set as follows: $l = \lceil \frac{i-1}{2^{t-1}}(N+1) \rceil$, $r = \lceil \frac{i}{2^{t-1}}(N+1) \rceil$, and $j = \lceil \frac{l+r}{2} \rceil = \lceil \frac{2i-1}{2^t}(N+1) \rceil$.

In details, the Dichotomic algorithm is shown in Figure 1.1. It uses the two matrices M and F, whose entries are again filled up row by row (Loop 1). A generic row k is filled in stages (Loop 2). Each stage corresponds to a particular value of the variable t (Loop 3). The variable j corresponds to the index of the entry which is currently being filled in stage t. The variables l (left) and r (right) correspond to the indices of the entries nearest to j which have been already filled, with $l < j < r$.

If no entry before j has been already filled, then $l = 1$, and therefore the final border $F_{k,1}$ is initialized to 1. If no entry after j has been filled, then $r = N$, and thus the final border $F_{k,N+1}$ is initialized to N. To compute the entry j, the variable ℓ takes all values between $F_{k,l}$ and $F_{k,r}$. The index ℓ which minimizes the recurrence in Loop 4 is assigned to $F_{k,j}$, while the corresponding minimum value is assigned to $M_{k,j}$.

To show the correctness, consider how a generic row k is filled up. In the first stage (i.e. $t = 1$), the entry $M_{k, \lceil \frac{N+1}{2} \rceil}$ is filled and ℓ ranges over all values $1, \ldots, N$. By Corollary 1.2, observe that to fill an entry $M_{k,l}$ where $l < \lceil \frac{N+1}{2} \rceil$, one needs to consider only the entries $M_{k-1,\ell}$ where $\ell \leq F_{k, \lceil \frac{N+1}{2} \rceil}$. Similarly, to fill an entry $M_{k,l}$ where $l > \lceil \frac{N+1}{2} \rceil$, one needs to consider only the entries $M_{k-1,\ell}$ where $\ell \geq F_{k, \lceil \frac{N+1}{2} \rceil}$. In general, one can show that in stage t, to compute the entries $M_{k,j}$ with $j = \lceil \frac{2i-1}{2^t}(N+1) \rceil$ and $1 \leq i \leq 2^{t-1}$, only the entries $M_{k-1,\ell}$ must be considered, where $F_{k,l} \leq \ell \leq F_{k,r}$ and l and r are $\lceil \frac{i-1}{2^{t-1}}(N+1) \rceil$ and $\lceil \frac{i}{2^{t-1}}(N+1) \rceil$, respectively. Notice that these entries have been computed in earlier stages. The above process repeats for every row of the matrix. The algorithm proceeds till the last entry $M_{K,N}$, the required optimal cost, is computed. The strict left-most optimal solution $SLMO_{N,K} = (B_1, B_2, \ldots, B_{K-1})$ is obtained, where $B_{k-1} = F_{k,B_k}$ for $1 < k \leq K$ and $B_K = N$.

As regard to the time complexity, first note that the total number of comparisons involved in a stage of

the Dichotomic algorithm is $O(N)$ since it is equal to the sum of the number of values the variable ℓ takes in Loop 3, that is:

$$\sum_{i=1}^{2^{t-1}} (F_{k,\lceil \frac{i}{2^{t-1}}(N+1)\rceil} - F_{k,\lceil \frac{i-1}{2^{t-1}}(N+1)\rceil} + 1) = F_{k,N+1} - F_{k,0} + 2^{t-1} = N - 1 + 2^{t-1} = O(N)$$

Since Loop 2 runs $\lceil \log N \rceil$ times and Loop 1 is repeated K times, the overall time complexity is $O(NK \log N)$.

3.3 Two Channels

This subsection exploits the structure of the optimal solution in the special case where the item lengths are uniform and there are only two channels. Indeed, as shown later, the values assumed varying ℓ in the right hand side of Recurrence 1.2 for $k = 2$ form a *unimodal* sequence. That is, there is a particular index ℓ such that the values on its left are in non-increasing order, while those on its right are in increasing order. By this fact, one can search the minimum of Recurrence 1.2 in a very effective way, improving on the overall running time.

Formally, the 2-Uniform Allocation Problem consists in finding a partition S into two groups G_1 and G_2 such that $AED_S = \frac{1}{2}(N_1 P_1 + N_2 P_2)$ is minimized. Clearly, $N = N_1 + N_2$, and by Lemma 1.1, $N_1 \leq N_2$ holds for any optimal solution. Moreover, recall that any feasible segmentation S for $K = 2$ can be denoted by the single border B_1, which coincides with N_1.

Lemma 1.3. [4] *Consider the uniform length items $d_1, d_2, \ldots, d_N$ sorted by non-increasing probabilities, and $K = 2$ channels. Let $S = (N_1)$ be a feasible segmentation such that $P_1 \leq P_2$. If the segmentation $S' = (N_1 + 1)$ is feasible, then $AED_{S'} \leq AED_S$.*

While Lemma 1.1 gives the upper bound $N_1 \leq \lfloor \frac{N}{2} \rfloor$ on the cardinality of group G_1, Lemma 1.3 provides a lower bound b on N_1. Indeed, it guarantees that any optimal solution contains at least the first b items $d_1, \ldots, d_b$, where b is the largest index for which $P_1 = \sum_{h=1}^{b} p_h \leq P_2 = \sum_{h=b+1}^{N} p_h$. Formally, Recurrence 1.2 for $K = 2$ can be rewritten as follows:

$$opt_{N,2} = \min_{b \leq \ell \leq \lfloor \frac{N}{2} \rfloor} \{C_{1,\ell} + C_{\ell+1,N}\} \tag{1.5}$$

where

$$b = \max_{1 \leq s \leq \lfloor \frac{N}{2} \rfloor} \left\{ s \ : \ \sum_{h=1}^{s} p_h \leq \sum_{h=s+1}^{N} p_h \right\}.$$

The following lemma improves on the upper bound of N_1 given by Lemma 1.1, and shows that the values of the feasible segmentations assumed in the right-hand side of Equation 1.5 form a unimodal sequence.

Lemma 1.4. [4] *Consider the uniform length items $d_1, d_2, \ldots, d_N$ sorted by non-increasing probabilities, and $K = 2$ channels. Let $S = (N_1)$ be a feasible solution such that $P_1 > P_2$. Consider the solutions $S' = (N_1 + 1)$ and $S'' = (N_1 + 2)$. If $AED_{S'} > AED_S$, then $AED_{S''} > AED_{S'}$.*

In practice, one can scan the feasible solutions of Equation 1.5 by moving the border ℓ rightwards, one position at a time, starting from the lower bound b obtained applying Lemma 1.3. The scan continues while the average expected delay of the current solution does not increase, but stops as soon as the average expected delay starts to increase. Indeed, by Lemma 1.4, further moving the border ℓ to the right can only increase the cost of the solutions. Hence the border m that minimizes Equation 1.5, that is the optimal solution of the problem, is given by:

$$opt_{N,2} = C_{1,m} + C_{m+1,N} \tag{1.6}$$

where

$$m = \min_{b \le \ell \le \lfloor \frac{N}{2} \rfloor} \{\ell \ : C_{1,\ell} + C_{\ell+1,N} < C_{1,\ell+1} + C_{\ell+2,N}\}\,.$$

Note that, in the above equation, the cost variation is:

$$(C_{1,\ell+1} + C_{\ell+2,N}) - (C_{1,\ell} + C_{\ell+1,N}) = \frac{1}{2}(\sum_{h=1}^{\ell} p_h - \sum_{h=\ell+1}^{N} p_h + p_{\ell+1}(2\ell + 2 - N)).$$

Due to the unimodal property of the sequence of values on the right-hand side of Equation 1.6, the search of m can be done in $O(\log N)$ time by a suitable modified binary search. Let $f(\ell) = C_{1,\ell} + C_{\ell+1,N} = \frac{\ell}{2}\sum_{h=1}^{\ell} p_h + \frac{N-\ell}{2}\sum_{h=\ell+1}^{N} p_h$. Then, the unimodal sequence consists of the values $f(b), f(b+1), \ldots, f(\lfloor \frac{N}{2} \rfloor)$. As said, solving Equation 1.6 is equivalent to find the index m such that $f(b) \ge \ldots \ge f(m) < f(m+1) < \ldots < f(\lfloor \frac{N}{2} \rfloor)$. This can be done by invoking the recursive procedure *BinSearch*, given in Figure 1.2, with parameters $i = b$ and $j = \lfloor \frac{N}{2} \rfloor$. The BinSearch procedure first computes the middle point $m = \lfloor \frac{i+j}{2} \rfloor$. Then, the values $f(m)$ and $f(m+1)$ are compared in the light of the unimodal sequence definition. If $f(m) \ge f(m+1)$, the minimum must belong to the right half, otherwise it must be in the left half. Procedure BinSearch proceeds recursively on the proper half until the minimum is reached.

4 Non-Uniform Lengths

Consider now the K-Non-Uniform Allocation Problem for an arbitrary number K of channels. In contrast to the uniform case, introducing items with different lengths makes the problem computationally intractable.

Theorem 1.1. [5] *The K-Non-Uniform Allocation Problem is NP-hard for $K = 2$, and strong NP-hard for an arbitrary K.*

As a consequence of the above result, there is no pseudo-polynomial time optimal algorithm or Fully Polynomial Time Approximation Scheme (FPTAS) for solving the K-Non-Uniform Allocation Problem (unless P=NP). However, when the maximum item length z is bounded by a constant, a polynomial time optimal algorithm can be derived where z appears in the exponent. When $z = 1$, this algorithm reduces to the DP algorithm. The following result generalizes Lemma 1.1.

Lemma 1.5. [5] *Let G_h and G_j be two groups in an optimal solution for the K-Non-Uniform Allocation Problem. Let d_i and d_k be items with $z_i = z_k$ and $d_i \in G_h$, $d_k \in G_j$. If $Z_h < Z_j$, then $p_i \geq p_k$. Similarly, if $p_i > p_k$, then $Z_h \leq Z_j$.*

Based on the above lemma, some additional notations are introduced. The set D of items can be viewed as a union of disjoint subsets $D_i = \{d_1^i, d_2^i, \ldots, d_{L_i}^i\}$, $1 \leq i \leq z$, where D_i is the set of items with length i, L_i is the cardinality of D_i, and z is the maximum item length. Let p_j^i represent the probability of item d_j^i, for $1 \leq j \leq L_i$.

The following corollary generalizes Corollary 1.1.

Corollary 1.3. [5] *Let $d_1^i, d_2^i, \ldots, d_{L_i}^i$ be the L_i items of length i with $p_m^i \geq p_n^i$ whenever $m < n$, for $i = 1, \ldots, z$. There is an optimal solution for partitioning the items of D into K groups $G_1, \ldots, G_K$, such that if $a < b < c$ and $d_a^i, d_c^i \in G_j$, then $d_b^i \in G_j$.*

In the rest of this section, the items in each D_i are assumed to be sorted by non-increasing probabilities,

and optimal solutions will be sought of the form:

$$\underbrace{d_1^1,\ldots,d_{B_1^{(1)}}^1}_{G_1},\underbrace{d_{B_1^{(1)}+1}^1,\ldots,d_{B_2^{(1)}}^1}_{G_2},\ldots,\underbrace{d_{B_{K-1}^{(1)}+1}^1,\ldots,d_{N_1}^1}_{G_K}$$
$$\underbrace{d_1^2,\ldots,d_{B_1^{(2)}}^2}_{G_1},\underbrace{d_{B_1^{(2)}+1}^2,\ldots,d_{B_2^{(2)}}^2}_{G_2},\ldots,\underbrace{d_{B_{K-1}^{(2)}+1}^2,\ldots,d_{N_2}^2}_{G_K}$$
$$\vdots$$
$$\underbrace{d_1^z,\ldots,d_{B_1^{(z)}}^z}_{G_1},\underbrace{d_{B_1^{(z)}+1}^z,\ldots,d_{B_2^{(z)}}^z}_{G_2},\ldots,\underbrace{d_{B_{K-1}^{(z)}+1}^z,\ldots,d_{N_z}^z}_{G_K}$$

where $B_j^{(i)}$ is the highest index among all items of length i in group G_j. The solution will be represented as $(\bar{B}_1,\bar{B}_2,\ldots,\bar{B}_{K-1})$, where each $\bar{B}_j$ is the z-tuple $(B_j^{(1)},B_j^{(2)},\ldots,B_j^{(z)})$ for $1\leq j\leq K-1$. From now on, $B_{K-1}^{(i)}$ will be referred to as the *final border for length* i and $\bar{B}_{K-1}$ as the *final border vector.*

Let $OPT_{n_1,\ldots,n_z,k}$ denote the optimal solution for grouping the $\sum_{i=1}^{z} n_i$ items $d_1^i,d_2^i,\ldots,d_{n_i}^i$, $1\leq i\leq z$, into k groups and let $opt_{n_1,\ldots,n_z,k}$ be its corresponding cost. Let $C_{l_1,n_1,\ldots,l_z,n_z}$ be the cost of putting items l_i through n_i, for all $i=1,2,\ldots,z$, into one group, i.e.

$$C_{l_1,n_1,\ldots,l_z,n_z}=\frac{1}{2}\left(\sum_{i=1}^{z} i(n_i-l_i+1)\right)\left(\sum_{i=1}^{z}\sum_{j=l_i}^{n_i} p_j^i\right)$$

Now, consider the recurrence:

$$opt_{n_1,\ldots,n_z,k}=\min_{\substack{\vec{\ell}=(\ell_1,\ldots,\ell_z)\\ 0\leq\ell_i\leq n_i,1\leq i\leq z}}\left\{opt_{\ell_1,\ldots,\ell_z,k-1}+C_{\ell_1+1,n_1,\ldots,\ell_z+1,n_z}\right\} \tag{1.7}$$

To solve this recurrence by using dynamic programming, consider a $(z+1)$-dimensional matrix M, made of K rows in the first dimension and L_i columns in dimension $i+1$ for $i=1,\ldots,z$. Each entry is represented by a $(z+1)$-tuple $M_{k,n_1,\ldots,n_z}$, where k corresponds to the row index and n_i corresponds to the index of the column in dimension $i+1$. The entry $M_{k,n_1,\ldots,n_z}$ represents the optimal cost for partitioning items d_1^i through $d_{n_i}^i$, for $i=1,2,\ldots z$, into k groups. There is also a similar matrix F where the entry $F_{k,n_1,\ldots,n_z}$ corresponds to the final border vector of the solution whose cost is $M_{k,n_1,\ldots,n_z}$. The matrix entries are filled row by row. The optimal solution is given by $OPT_{L_1,\ldots,L_z,K}=(\bar{B}_1,\bar{B}_2,\ldots,\bar{B}_{K-1})$ where, starting from $\bar{B}_K=(L_1,L_2,\ldots,L_z)$, the value of $\bar{B}_k$ is obtained from the value of $\bar{B}_{k+1}$ and by F as $\bar{B}_k=F_{k+1,\bar{B}_{k+1}}$, for $k=1,\ldots,K-1$. The Optimal algorithm derives directly from Recurrence 1.7. Since the computation

of every entry $M_{k,n_1,\ldots,n_z}$ and $F_{k,n_1,\ldots,n_z}$ requires $\prod_{i=1}^{z}(n_i+1) \leq \prod_{i=1}^{z}(L_i+1)$ comparisons, and every row has $\prod_{i=1}^{z} L_i$ entries, the overall time complexity is $O(K\prod_{i=1}^{z}(L_i+1)^2) = O(KN^{2z})$.

5 Heuristics

Since the K-Non-Uniform Allocation Problem is strong NP-hard, it results to be computationally intractable (unless P=NP). In practice, this implies that one is forced to abandon the search for efficient algorithms which find optimal solutions. Therefore, one can devise fast and simple heuristics that provide solutions which are not necessarily optimal but usually fairly close. This strategy is followed in this section, where the main heuristics are reviewed. All heuristics assume that the items are sorted by non-increasing $\frac{p_i}{z_i}$ ratios, which can be done in $O(N \log N)$ time during a preprocessing step.

5.1 The Greedy Algorithm

The Greedy heuristic [16, 17] initially assigns all the N data items to a single group. Then, for $K-1$ times, one of the groups is split in two groups, that will be assigned to two different channels. To find which group to split along with its actual split point, all the possible points of all groups are considered as split point candidates, and the one that decreases AED the most is selected. In details, assume that the channel to be split contains the items from d_i to d_j, with $1 \leq i < j \leq N$, and let $cost_{i,j,2}$ denote the cost of a feasible solution for assigning such items to two channels. Then, the split point is the index m that satisfies:

$$cost_{i,j,2} = C_{i,m} + C_{m+1,j} = \min_{i \leq \ell \leq j-1} \{C_{i,\ell} + C_{\ell+1,j}\} \tag{1.8}$$

An efficient implementation takes advantage from the fact that, between two subsequent splits, it is sufficient to recompute the costs for the split point candidates of the last group that has been actually split. The time complexity of the Greedy heuristic is $O(N(K + \log N))$ and $O(N \log N)$ in the worst and average cases, respectively [4].

Note that Greedy scales well when changes occur on the number of channels, on the number of items, on item probabilities, as well as on item lengths. Indeed, adding or removing a channel simply requires doing a new split or removing the last introduced split, respectively. Adding a new item first requires to insert such

an item in the sorted item sequence. Assume the new item is added to group G_j, then the border of the two-channel subproblem including items of G_j and G_{j+1} is recomputed by applying Equation 1.8. Similarly, deleting an item that belongs to group G_j requires to solve again the two-channel subproblem including items of G_j and G_{j+1}. Finally, a change in the probability/length of an item is equivalent to first removing that item and then adding the same properly modified item.

5.2 The Greedy+ Algorithm

The Greedy+ heuristic [4] is a refinement of the Greedy heuristic and consists of two phases. In the first phase it behaves as Greedy, except the way the split point is determined. In the second phase, the solution provided by the first phase is refined by working on pairs of consecutive channels.

Specifically, in the first phase, Greedy+ uses an approach similar to that of Equation 1.6 to determine the split point. This is because splitting one channel is the same as solving the problem for two channels. In details, the split point m is given by:

$$cost_{i,j,2} = C_{i,m} + C_{m+1,j} \tag{1.9}$$

where

$$m = \min_{i \leq \ell \leq j-1} \{\ell \ : C_{i,\ell} + C_{\ell+1,j} < C_{i,\ell+1} + C_{\ell+2,j}\}\,.$$

Note that, since the item lengths are not the same, the sequence of values $C_{i,\ell} + C_{\ell+1,j}$, for $i \leq \ell \leq j-1$, is not unimodal. However, Greedy+ behaves as such a sequence were unimodal. Instead of trying all the possible values of ℓ between i and j, as done by Greedy, Greedy+ performs a left-to-right scan starting from i and stopping as soon the AED increases. In this way, a sub-optimal solution $S = (B_1, B_2, \ldots . B_{K-1})$ is found.

The second phase is performed only when $K \geq 3$ and consists in refining the solution S by recomputing its borders. The phase consists in a sequence of odd steps, followed by a sequence of even steps. During the t-th odd step, $1 \leq t \leq \lfloor \frac{K}{2} \rfloor$, the two-channel subproblem including the items assigned to groups G_{2t-1} and G_{2t} is solved. Specifically, Equation 1.9 is applied choosing $i = B_{2t-2} + 1$ and $j = B_{2t}$, thus recomputing the border B_{2t-1} of S. Similarly, during the t-th even step, $1 \leq t \leq \lfloor \frac{K-1}{2} \rfloor$, the two-channel subproblem

including the items assigned to groups G_{2t} and G_{2t+1} is solved by applying Equation 1.9 with $i = B_{2t-1} + 1$ and $j = B_{2t+1}$, recomputing the border B_{2t} of S.

The initial sorting requires $O(N \log N)$ time. Since each split runs in $O(N)$ time, and K splits are computed, the first phase of Greedy+ takes $O(NK)$ time. The second phase of Greedy+ requires $O(N)$ time since each item is considered as a candidate split point at most in a single split computation among all the odd steps, and in a single split computation among the even steps. Therefore, the overall time required in the worst case by the Greedy+ heuristic is $O(N(K + \log N))$. Clearly, Greedy+ maintains the same scaling features as Greedy.

5.3 The Dlinear Algorithm

The Dlinear heuristic [4] follows a dynamic programming approach similar to that provided by Recurrence 1.2. Fixed k and n, Dlinear computes a solution for n items from the previously computed solution for $n-1$ items and k channels, exploiting the characteristics of the optimal solutions for two channels and uniform lengths.

Let $M_{k,n}$ and $F_{k,n}$ be defined as in Subsection 3.1. Dlinear selects the feasible solutions that satisfy the following Recurrence:

$$M_{k,n} = \begin{cases} C_{1,n} & \text{if } k = 1 \\ M_{k-1,m} + C_{m+1,n} & \text{if } k > 1 \end{cases} \tag{1.10}$$

where

$$m = \min_{F_{k,n-1} \le \ell \le n-1} \{\ell \ : M_{k-1,\ell} + C_{\ell+1,n} < M_{k-1,\ell+1} + C_{\ell+2,n}\} .$$

In practice, Dlinear pretends to adapt Equation 1.6, that holds for the 2-Uniform Allocation Problem, also to the K-Non-Uniform Allocation Problem. In particular, the choice of the lower bound $F_{k,n-1}$ in the formula of m is suggested by Lemma 1.2 which says that the border of channel $k-1$ can only move right when a new item with the smallest probability is added. Moreover, m is determined as in Equation 1.6 pretending that the sequence $M_{k-1,\ell} + C_{\ell+1,n}$, obtained for $F_{k,n-1} \le \ell \le n-1$, be unimodal. Therefore, the solution provided by Dlinear is a sub-optimal one.

As regard to the time complexity, computing $M_{k,n}$ requires $O(F_{k,n} - F_{k,n-1})$ time. Hence, row k of M is

filled in $\sum_{n=1}^{N} O(F_{k,n} - F_{k,n-1}) = O(F_{k,N} - F_{k,1}) = O(N)$ time. Since M has K rows and the sorting step takes $O(N \log N)$ time, the overall time complexity of the Dlinear algorithm is $O(N(K + \log N))$.

6 Experimental Tests

In this section, the behaviour of the Greedy, Greedy+ and Dlinear heuristics is tested. The algorithms are written in C and the experiments are run on an AMD Athlon XP 2500+, 1.84 GHz, with 1 GB RAM.

The heuristics are executed on the following non-uniform length instances. Given the number N of items and a real number $0 \leq \theta \leq 1$, the item probabilities are generated according to a Zipf distribution whose skew is θ, namely:

$$p_i = \frac{(1/i)^\theta}{\sum_{i=1}^{N}(1/i)^\theta} \qquad 1 \leq i \leq N$$

In the above formula, $\theta = 0$ stands for a uniform distribution with $p_i = \frac{1}{N}$, while $\theta = 1$ implies a high skew, namely the range of p_i values becomes larger. The item lengths z_i are integers randomly generated according to a uniform distribution in the range $1 \leq z_i \leq z$. The items are sorted by non-increasing $\frac{p_i}{z_i}$ ratios. The parameters N, K, z, and θ vary, respectively, in the ranges: $500 \leq N \leq 2500$, $10 \leq K \leq 500$, $3 \leq z \leq 10$, and $0.5 \leq \theta \leq 1$.

Since the Optimal algorithm can find the exact solutions in a reasonable time only for small instances, a *lower bound* on AED is used for large values of N, K, and z. The lower bound for a non-uniform instance is obtained by transforming it into a uniform instance as follows. Each item d_i of probability p_i and length z_i is decomposed in z_i items of probability $\frac{p_i}{z_i}$ and length 1. Since more freedom has been introduced, it is clear that the optimal AED for the so transformed problem is a lower bound on the AED of the original problem. Since the transformed problem has uniform lengths, its optimal AED is obtained by running the Dichotomic algorithm.

The simulation results are exhibited in Tables 1.1, 1.2, 1.3, and 1.4. The tables report the time (measured in microseconds), the AED, and the percentage of error, which is computed as

$$\left(\frac{\mathrm{AED}_{\mathrm{heuristic}} - \mathrm{AED}_{\mathrm{lowerbound}}}{\mathrm{AED}_{\mathrm{lowerbound}}}\right) 100$$

The running times reported in the tables do not include the time for sorting.

By observing the tables, one notes that Greedy+ and Dlinear always outperform Greedy in terms of solution quality. In particular, Greedy+ at least halves the error of Greedy, producing solutions whose errors is at most 5.7%. Moreover, Dlinear reaches the optimum almost in all cases, and its maximum error is as high as 1.8% only in one instance. As regard to the running times, although all the three heuristics have the same asymptotic worst case time, Greedy is the fastest in practice. Although Greedy+ and Dlinear are slower than Greedy, their running times are always less than one tenth of second. The experiments show that Greedy+ and Dlinear behave well when the item probabilities follow a Zipf distribution. This suggests that, in most cases, the AED achieved in correspondence of the leftmost value of ℓ satisfying Recurrences 1.9 and 1.10 is the optimal AED or it is very close to the optimal AED. In other words, the sequence of values obtained by varying ℓ is almost unimodal.

In summary, the experimental tests show that the Dlinear heuristic finds optimal solutions almost always. In contrast, Greedy is the fastest heuristic, but produces the worst solutions. Finally, Greedy+ presents running times and sub-optimal solutions which are both intermediate between those of Greedy and Dlinear. Therefore, the choice among the heuristics depends on the goal to be pursued. If one is interested in finding the best sub-optimal solutions, then Dlinear should be adopted. Instead, if the running time is the main concern, then Greedy should be chosen, while if adaptability to parameter changes is the priority, then either Greedy or Greedy+ could be applied. In this scenario, Greedy+ represents a good compromise since it is scalable and produces fairly good solutions.

7 Conclusions

In this chapter, the problem of data broadcasting over multiple channels, with the objective of minimizing the average expected delay (AED) of the clients, was considered under the assumptions of skewed allocation to multiple channels and flat scheduling per channel. Both the uniform and non-uniform length problems were solved to the optimum, illustrating exact algorithms based on dynamic programming. Moreover, effective heuristics for non-uniform lengths have also been shown. All the results reviewed in this chapter are summarized in Table 1.5.

In this chapter, the client delay has been defined as the overall time elapsed from the moment the client

desires a data item to the moment the item download starts. Such a definition assumes that indexing is already available to the client. Hence, the client delay does not include the tuning time spent by the client for actively retrieving the index information and the data item. Thus, after reading the index, the client can be turned into a power saving mode until the data item appears on the proper channel. Therefore, our solution minimizes the average expected delay and keeps as low as possible the tuning time provided that an efficient index strategy is adopted on one or more separate channels. In our solution, the index can be readily derived from the $(K-1)$-tuple $(B_1, B_2, \ldots, B_{K-1})$, which compactly represents the data allocation. However, this tuple is enough for indexing only if all the clients know, as a global information, the relative position of each data item within the set of all data items sorted by probabilities. To overcome this assumption, new solutions can be sought that, without using global information on data items, either mix index and data items within the same channels or optimize the index broadcasting on dedicated channels [11, 14].

References

[1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: data management for asymmetric communication environments. In *Proc. SIGMOD*, May 1995.

[2] M.H. Ammar and J.W. Wong. The design of teletext broadcast cycles. *Performance Evaluation*, 5(4):235–242, 1985.

[3] M.H. Ammar and J.W. Wong. On the optimality of cyclic transmission in teletext systems. *IEEE Transactions on Communications*, 35(11):1159–1170, 1987.

[4] S. Anticaglia, F. Barsi, A.A. Bertossi, L. Iamele, and M.C. Pinotti. Efficient Heuristics for Data Broadcasting on Multiple Channels. *Technical Report*, 2005/5, Department of Mathematics and Computer Science, University of Perugia, 2005.

[5] E. Ardizzoni, A.A. Bertossi, M.C. Pinotti, S. Ramaprasad, R. Rizzi, and M.V.S. Shashanka. Optimal Skewed Data Allocation on Multiple Channels with Flat Broadcast per Channel. *IEEE Transactions on Computers*, 54(5):558–572, 2005.

[6] A. Bar-Noy, R. Bhatia, J.S. Naor, and B. Schieber. Minimizing service and operation costs of periodic scheduling. In *Proc. Ninth ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 11–20, 1998.

[7] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: evidence and implications. In *Proc. IEEE INFOCOM*, 1999.

[8] T. Imielinski, S. Viswanathan, and B.R. Badrinath. Energy efficient indexing on air. In *Proc. SIGMOD*, May 1994.

[9] C. Kenyon and N. Schabanel. The data broadcast problem with non-uniform transmission time. In *Proc. Tenth ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 547–556, 1999.

[10] C. Kenyon, N. Schabanel, and N. Young. Polynomial time approximation scheme for data broadcast. In *Proc. ACM Symp. on Theory of Computing (STOC)*, pages 659–666, 2000.

[11] S.-C. Lo and A.L.P. Chen Optimal index and data allocation in multiple broadcast channels. In *Proc. Sixteenth IEEE Int'l Conf. on Data Engineering (ICDE)*, February 2000.

[12] W.C. Peng and M.S. Chen. Efficient channel allocation tree generation for data broadcasting in a mobile computing environment. *Wireless Networks*, 9(2):117–129, 2003.

[13] K.A. Prabhakara, K.A. Hua, and J. Oh. Multi-level multi-channel air cache designs for broadcasting in a mobile environment. In *Proc. Sixteenth IEEE Int'l Conf. on Data Engineering (ICDE)*, February 2000.

[14] I. Stojmenovic (Editor). *Handbook of Wireless Networks and Mobile Computing.* Wiley, Chichester, 2002.

[15] N. Vaidya and S. Hameed. Log time algorithms for scheduling single and multiple channel data broadcast. In *Proc. Third ACM-IEEE Conf. on Mobile Computing and Networking (MOBICOM)*, September 1997.

[16] W.G. Yee, Efficient data allocation for broadcast disk arrays. *Technical Report*, GIT-CC-02-20, Georgia Institute of Technology, 2001.

[17] W.G. Yee, S. Navathe, E. Omiecinski, and C. Jermaine. Efficient data allocation over multiple channels at broadcast servers. *IEEE Transactions on Computers*, 51(10):1231–1236, 2002.

Input: N items sorted by non-increasing probabilities, and K groups;
Initialize: **for** i **from** 1 **to** N **do**
for k **from** 1 **to** K **do**
if $k = 1$ **then** $M_{k,i} \leftarrow C_{k,i}$ **else** $M_{k,i} \leftarrow \infty$;
Loop 1: **for** k **from** 2 **to** K **do**
$F_{k,0} \leftarrow F_{k,1} \leftarrow 1$; $F_{k,N+1} \leftarrow N$;
Loop 2: **for** t **from** 1 **to** $\lceil \log N \rceil$ **do**
Loop 3: **for** i **from** 1 **to** 2^{t-1} **do**
$j \leftarrow \lceil \frac{2i-1}{2^t}(N+1) \rceil$; $l \leftarrow \lceil \frac{i-1}{2^{t-1}}(N+1) \rceil$; $r \leftarrow \lceil \frac{i}{2^{t-1}}(N+1) \rceil$;
if $M_{k,j} = \infty$ **then**
Loop 4: **for** ℓ **from** $F_{k,l}$ **to** $F_{k,r}$ **do**
if $M_{k-1,\ell} + C_{\ell+1,j} < M_{k,j}$ **then**
$M_{k,j} \leftarrow M_{k-1,\ell} + C_{\ell+1,j}$;
$F_{k,j} \leftarrow \ell$;

Figure 1.1: The Dichotomic algorithm for the K-Uniform Allocation Problem. (Figure reprinted, with permission, from [5] *IEEECS Log Number TC-0238-0704*, ©2005, IEEE.)

```
Procedure BinSearch (i, j);
    m ← ⌊(i+j)/2⌋;
    if i = j then
        return m
    else
        if f(m) ≥ f(m+1) then
            BinSearch (m+1, j)
        else
            BinSearch (i, m);
```

Figure 1.2: The binary search on a unimodal sequence.

$N/K/\theta/z$	**Algorithm**	**AED**	**% Error**	**Time**
500/20/0.8/3	Greedy	18.72	7.1	102
	Greedy+	17.58	0.6	3514
	Dlinear	17.47		2106
	Lower bound	17.47		
1500/20/0.8/3	Greedy	53.85	7.9	283
	Greedy+	51.71	3.6	21240
	Dlinear	49.90		6519
	Lower bound	49.90		
1750/20/0.8/3	Greedy	62.64	7.9	326
	Greedy+	58.92	1.5	31137
	Dlinear	58.04		7488
	Lower bound	58.04		
2000/20/0.8/3	Greedy	71.24	7.9	373
	Greedy+	66.93	1.4	38570
	Dlinear	65.98		8602
	Lower bound	65.98		
2250/20/0.8/3	Greedy	79.70	7.8	457
	Greedy+	75.06	1.6	45170
	Dlinear	73.87		9749
	Lower bound	73.87		
2500/20/0.8/3	Greedy	88.40	7.8	474
	Greedy+	82.51	0.7	62376
	Dlinear	81.93		10920
	Lower bound	81.93		

Table 1.1: Experimental results when $K = 20$, $\theta = 0.8$, and $z = 3$.

$N/K/\theta/z$	**Algorithm**	**AED**	**% Error**	**Time**
2500/10/0.8/3	Greedy	179.16	7.8	381
	Greedy+	167.86	1.0	97356
	Dlinear	166.14		4919
	Lower bound	166.14		
2500/40/0.8/3	Greedy	44.04	7.9	562
	Greedy+	41.58	1.9	34147
	Dlinear	40.79		22771
	Lower bound	40.79		
2500/80/0.8/3	Greedy	21.98	7.9	685
	Greedy+	20.72	1.7	19179
	Dlinear	20.37		46545
	Lower bound	20.37		
2500/100/0.8/3	Greedy	17.14	5.2	740
	Greedy+	16.75	2.8	27452
	Dlinear	16.29		57906
	Lower bound	16.29		
2500/200/0.8/3	Greedy	8.56	5.1	1009
	Greedy+	8.37	2.8	12974
	Dlinear	8.15	0.1	116265
	Lower bound	8.14		
2500/500/0.8/3	Greedy	3.4	4.2	2313
	Greedy+	3.35	2.7	21430
	Dlinear	3.32	1.8	273048
	Lower bound	3.26		

Table 1.2: Experimental results when $N = 2500$, $\theta = 0.8$, and $z = 3$.

$N/K/\theta/z$	**Algorithm**	**AED**	**% Error**	**Time**
2500/50/0.5/3	Greedy	47.74	9.7	595
	Greedy+	46.02	5.7	23175
	Dlinear	43.52	0.02	29075
	Lower bound	43.51		
2500/50/0.7/3	Greedy	39.59	6.8	600
	Greedy+	38.47	3.8	23606
	Dlinear	37.05	0.02	29132
	Lower bound	37.04		
2500/50/0.8/3	Greedy	34.33	5.2	603
	Greedy+	33.49	2.6	24227
	Dlinear	32.61		29121
	Lower bound	32.61		
2500/50/1/3	Greedy	23.10	3.2	609
	Greedy+	22.53	0.6	27566
	Dlinear	22.38		28693
	Lower bound	22.38		

Table 1.3: Experimental results when $N = 2500$, $K = 50$, and $z = 3$.

$N/K/\theta/z$	**Algorithm**	**AED**	**% Error**	**Time**
500/50/0.8/3	Greedy	7.34	5.3	147
	Greedy+	7.19	3.1	2517
	Dlinear	6.98	0.1	5423
	Lower bound	6.97		
500/50/0.8/5	Greedy	10.78	5.3	147
	Greedy+	10.52	2.8	2938
	Dlinear	10.25	0.1	5490
	Lower bound	10.23		
500/50/0.8/7	Greedy	14.50	4.9	146
	Greedy+	14.16	2.4	3329
	Dlinear	13.85	0.2	5499
	Lower bound	13.82		
500/50/0.8/10	Greedy	19.48	5.1	145
	Greedy+	18.97	2.3	3899
	Dlinear	18.58	0.2	5507
	Lower bound	18.53		

Table 1.4: Experimental results when $N = 500$, $K = 50$, and $\theta = 0.8$.

Item Lengths	**Complexity**	**Solution**	**Algorithm**	**Time**	**References**
uniform	P	optimal	DP	$O(KN^2)$	[17]
		optimal	Dichotomic	$O(KN \log N)$	[5]
non-uniform	strong NP-hard	optimal	Optimal	$O(KN^{2z})$	[5]
		heuristic	Greedy. Greedy+, Dlinear	$O(N(K + \log N))$	[16, 4]

Table 1.5: Results for broadcasting N data items on K channels with skewed allocation and flat scheduling.

Index