Optimal Skewed Data Allocation on Multiple Channels with Flat Broadcast per Channel

Elia Ardizzoni, Alan A. Bertossi, M. Cristina Pinotti, *Member*, *IEEE Computer Society*, Shashank Ramaprasad, Romeo Rizzi, and Madhusudana V.S. Shashanka

Abstract—Broadcast is an efficient and scalable way of transmitting data to an unlimited number of clients that are listening to a channel. Cyclically broadcasting data over the channel is a basic scheduling technique, which is known as *flat* scheduling. When multiple channels are available, a data allocation technique is needed to assign data to channels. Partitioning data among channels in an unbalanced way, depending on data popularities, is an allocation technique known as *skewed* allocation. In this paper, the problem of data broadcasting over multiple channels is considered, assuming skewed data allocation to channels and flat data scheduling per channel, with the objective of minimizing the average waiting time of the clients. First, several algorithms, based on dynamic programming, are presented which provide optimal solutions for N data items and K channels. Specifically, for data items with uniform lengths, an $O(NK \log N)$ time algorithm is proposed, which improves over the previously known $O(N^2K)$ time algorithm. When $K \leq 4$, a simpler $O(N \log N)$ time algorithm is exhibited which requires only $O(N)$ time if the data items are sorted. Moreover, for data items with nonuniform lengths, it is shown that the problem is *NP*-hard when $K = 2$ and *strong NP*-hard for arbitrary K. In the former case, a pseudopolynomial algorithm is discussed whose time is $O(NZ)$, where Z is the sum of the data lengths. In the latter case, an algorithm is devised with time exponential in the maximum data length, which can optimally solve, in reasonable time, only small instances. For larger instances, a new heuristic is devised which is experimentally tested on some benchmarks whose popularities are characterized by Zipf distributions. Such experimental tests reveal that the new heuristic proposed here always outperforms the best previously known heuristic in terms of solution quality.

Index Terms—Wireless communication, data broadcast, multiple channels, skewed allocation, flat scheduling, average waiting time, dynamic programming.

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 info, 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 multichannel 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 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], [4], [7], [8], [9], [12], [13], [15], [17], [18].

Specifically, the networking community faces a version of the problem, known as the *Broadcast Problem*, whose goal

- *E. Ardizzoni and A.A. Bertossi are with the Department of Computer Science, University of Bologna, 40127 Bologna, Italy. E-mail: ardizzoni@toomuchsoftware.it, bertossi@cs.unibo.it.*
- *M.C. Pinotti is with the Department of Computer Science and Mathematics, University of Perugia, 06123 Perugia, Italy. E-mail: pinotti@unipg.it.*
- *S. Ramaprasad is with the Department of Computer Science, Brown University, Providence, RI 02912. E-mail: shashank@cs.brown.edu.*
- *R. Rizzi is with the Department of Computer Science and Telecommunications, University of Trento, 38050 Povo, Trento, Italy. E-mail: rrizzi@science.unitn.it.*
- *M.V.S. Shashanka is with the Department of Cognitive and Neural Systems, Boston University, Boston, MA 02215. E-mail: mvss@cns.bu.edu.*

Manuscript received 17 July 2004; revised 19 Nov. 2004; accepted 22 Nov. 2004; published online 16 Mar. 2005.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0238-0704.

is to find an infinite schedule on a single channel [15], [4], [8], [9]. Such a problem was first introduced in 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 [4], [8]), 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 multichannel 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 [9]. In contrast, for *nonuniform* 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 [8].

On the other hand, the database community seeks a periodic broadcast scheduling which should be easily indexed [7]. For the single channel, the obvious schedule that admits index is the *flat* one. It consists of selecting an order among the data items and then transmitting them one 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], [18]. 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], [18]. 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 proven in [18], the optimal solution for uniform lengths can be found in polynomial time. Indeed, in that paper, two algorithms have been proposed, which assume that an $O(N \log N)$ sorting preprocessing step has been done on the N data items. Precisely, an $O(N^2K)$ dynamic programming algorithm, called *DP*, was proposed, where K is the number of channels. A faster $O(N \log N)$ heuristic, called *Greedy*, was also proposed in the same paper, which experimentally provides good suboptimal solutions on some benchmarks whose popularities are characterized by Zipf distributions. Such distributions have been shown to characterize the popularity of one element among a set of similar data, like a Web page in a Web site [5]. For nonuniform lengths, the problem tractability was unknown, but the Greedy heuristic was easily adapted, as shown in [17], [18].

The present paper expands the work started in [18], under the same assumptions, namely, skewed data allocation to channels and flat data scheduling per channel. Both the uniform and nonuniform length problems are faced and solved to the optimum, proposing faster algorithms for the uniform case and establishing the intractability (*NP*-hardness) of the nonuniform case. A new heuristic is also given which outperforms Greedy in terms of the solution quality, in both the uniform and nonuniform cases, at the cost of a slightly larger running time.

Specifically, for uniform lengths, a new $O(NK \log N)$ time algorithm is designed, called here *Dichotomic*, which is based on dynamic programming and finds optimal solutions. When $K \leq 4$, a simpler $O(N \log N)$ time algorithm, named *Le4*, is exhibited which requires only $O(N)$ time when the data items are already sorted. Moreover, for nonuniform lengths, it is shown that the problem is *NP*-hard when $K = 2$, and *strong NP*-hard for arbitrary K. When $K = 2$, a pseudopolynomial time algorithm is discussed which incrementally solves several Knapsack instances. Its overall time is $O(NZ)$, where Z is the sum of the data lengths. For arbitrary K, an algorithm, called *Optimal*, is devised with time exponential in the maximum data length z. The Optimal algorithm requires $O(KN^{2z})$ time and reduces to the DP algorithm when $z = 1$. When $z > 1$, algorithm Optimal can solve only small instances in a reasonable time. Therefore, for larger instances, a new heuristic, called *Move1*, is proposed whose time complexity is $O(N(K + \log N))$. Experimental tests reveal that Move1 always outperforms Greedy in terms of solution quality at the cost of a slightly larger running time. Moreover, in the uniform case, Move1 always finds optimal solutions on the same benchmarks on which Greedy has been tested.

The rest of this paper is organized as follows: Section 2 gives notations, definitions, and the problem statement. Section 3 efficiently solves the problem assuming uniform lengths. In particular, Section 3.1 illustrates the Dichotomic algorithm for an arbitrary number of channels, while Section 3.2 presents the simpler Le4 algorithm when there are at most four channels. In contrast, Section 4 studies the nonuniform length case. It first shows the strong *NP*-hardness for an arbitrary number of channels and then presents the Optimal algorithm, which requires exponential time. Section 4.1 discusses the nonuniform problem with only two channels, providing the *NP*-hardness proof and the reduction to the Knapsack problem, whereas Section 4.2 gives the Move1 heuristic. Section 5 reports the experimental tests on some benchmarks. Finally, conclusions and open questions are offered in Section 6.

2 PRELIMINARIES

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 demand probability of item d_i being 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 (or ticks) 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 to be *nonuniform*.

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, while the sum of its item lengths is denoted by Z_j, i.e., $Z_j = \sum_{d_i \in G_j} z_i$. 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, as in [18], that indexing allows clients to know in advance the content of the channels, the *average expected delay* (AED) over all channels is

$$\text{AED} = \frac{1}{2} \sum_{j=1}^{K} \left(Z_j \sum_{d_i \in G_j} p_i \right). \tag{1}$$

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 of partitioning D into K groups $G_1, \ldots, G_K$ so as to minimize the objective function AED given in (1). In the special case of equal lengths, the above problem is called the *K-Uniform Allocation Problem* and the corresponding objective function is derived replacing Z_j with N_j in (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

$$\frac{1}{2}(0.37 + 2(0.25 + 0.18) + 3(0.11 + 0.05 + 0.04)) = 0.915.$$

For ease of reference, all of the symbols that will be defined in the rest of this paper are summarized in Table 1. The rest of this section is devoted to briefly recalling the dynamic programming solution proposed in [18] for the *K*-Uniform Allocation Problem.

Lemma 1 [18]. *Let G_h and G_j be two groups in an optimal solution. 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. The following corollary shows how to exploit Lemma 1 in cleaning the structure of the *K*-Uniform Allocation Problem.

Corollary 1 [18]. *Let $d_1, d_2, \ldots, d_N$ be N 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.*

Hereafter, thus, it is assumed that the items are sorted by nonincreasing probabilities and the optimal solutions will be sought within the class of the *segmentations*. A *segmentation* is a partition $G_1, \ldots, G_K$ such that if $d_i \in G_j$ and $d_k \in G_j$, then $d_h \in G_j$ whenever $i \leq h \leq k$. A segmentation

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

will be more compactly denoted by the $(K-1)$-tuple

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

of its *right borders*, where border B_j is the index of the last item that belongs to group G_j. 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_j, i.e., the number N_j of items in the group, is $N_j = B_j - B_{j-1}$, where $B_0 = 0$ and $B_K = N$ are assumed.

For any two integers $n \leq N$ and $k \leq K$, 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. Let $C_{i,h}$ be the cost of assigning consecutive items $d_i, \ldots, d_h$ to one group, i.e., $C_{i,h} = \frac{1}{2}(h - i + 1) \sum_{q=i}^{h} p_q$. Hence, $opt_{n,1} = C_{1,n}$ for every n. For $k > 1$, the following recurrence holds:

$$opt_{n,k} = \min_{\ell \in \{1,2,\ldots,n-1\}} \{opt_{\ell,k-1} + C_{\ell+1,n}\}. \tag{2}$$

The DP algorithm proposed in [18] is a straightforward dynamic programming implementation of the recurrence in (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 the recurrence in (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 the recurrence of (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 $k = 1, \ldots, K-1$.

To evaluate the time complexity of the DP algorithm, observe that $O(N)$ comparisons are required to fill every entry of the matrix M, 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 $O(N^2K)$.

TABLE 1
Table of Symbols

Symbol	Description
AED	average expected delay
B_j	maximum index (right border) among items of G_j
B_j^i	maximum index (right border) among items of G_j in $LMO_{i,k}$
$B_j^{(i)}$	highest index among items of length i in G_j
$\bar{B}_j$	the z-tuple $(B_j^{(1)}, B_j^{(2)}, \ldots, B_j^{(z)})$
$C_{i,h}$	cost of assigning items d_i through d_h to one group
$C_{l_1,n_1,\ldots,l_z,n_z}$	cost of assigning items l_i through n_i, $1 \leq i \leq z$, to one group
D	set of data items
D_i	set of all items of length i
d_i	i-th data item
$d_i^{(j)}$	i-th data item of group G_j
d_j^i	the j-th item of length i
$F_{k,n}$	entry of the matrix F storing the final border of $OPT_{n,k}$
G_j	group of data items assigned to channel j
K	number of channels
L_i	number of items of length i
$LMO_{n,k}$	leftmost optimal solution for the first n items on k channels
$LMO_{i,j,2}$	leftmost optimal solution for items d_i through d_j on 2 channels
$M_{k,n}$	entry of the dynamic programming matrix M storing $opt_{n,k}$
$M_{k,n_1,\ldots,n_z}$	entry of the dynamic programming matrix storing $opt_{n_1,\ldots,n_z,k}$
N	number of data items
N_j	number of data items (i.e. cardinality) of G_j
$OPT_{n,k}$	optimal solution for the first n items on k channels
$opt_{n,k}$	cost (i.e. AED) of $OPT_{n,k}$
$opt_{i,j,2}$	cost (i.e. AED) of $LMO_{i,j,2}$
$OPT_{n_1,\ldots,n_z,k}$	optimal solution for grouping items $d_1^i, d_2^i, \ldots, d_{n_i}^i$, $1 \leq i \leq z$, into k groups
$opt_{n_1,\ldots,n_z,k}$	cost of $OPT_{n_1,\ldots,n_z,k}$
p_i	demand probability of item d_i
P_j	sum of the probabilities of data items in G_j
$p_i^{(j)}$	demand probability of $d_i^{(j)}$
p_j^i	demand probability of d_j^i
$S_{n,N,2}$	feasible solution for items d_n through d_N on 2 channels
$s_{n,N,2}$	cost of $S_{n,N,2}$
$SLMO_{N,K}$	the unique strict leftmost optimal solution for N items on K channels
θ	skew of a Zipf distribution
$X_{i,j}$	entry of the boolean matrix X storing the knapsack solution
z	maximum data item length
Z	sum of all data item lengths
z_i	length of item d_i
Z_j	sum of the lengths of data items in G_j

3 Uniform Lengths

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. *Let $d_1, d_2, \ldots, d_N$ be 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 2. *A left-most optimal solution $(B_1, B_2, \ldots, B_{K-1})$ 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 algorithms to be presented will compute a left-most optimal solution for every $i < K$ and, thus, they will find the unique strict left-most optimal solution.

Lemma 2. *Let the items $d_1, d_2, \ldots, d_N$ be sorted by nonincreasing 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, $B'_{K-1} \geq B_{K-1}$.*

Proof. Let the costs of $LMO_{N-1,K}$ and $OPT_{N,K}$ be, respectively, $opt_{N-1,K} = opt_{B_{K-1},K-1} + C_{B_{K-1}+1,N-1}$ and $opt_{N,K} = opt_{B'_{K-1},K-1} + C_{B'_{K-1}+1,N}$.

Consider the feasible solution for partitioning N items into K channels obtained from $(B_1, B_2, \ldots, B_{K-1})$ just adding d_N to the Kth channel. Then:

$$opt_{B'_{K-1},K-1} + C_{B'_{K-1}+1,N} = opt_{N,K} \leq opt_{B_{K-1},K-1} + C_{B_{K-1}+1,N}. \tag{3}$$

By definition,

$$C_{B'_{K-1}+1,N} - C_{B'_{K-1}+1,N-1} = \frac{1}{2}\left(p_N(N - B'_{K-1}) + \sum_{i=B'_{K-1}+1}^{N-1} p_i\right)$$

and

```
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} ← C_{k,i} else M_{k,i} ← ∞;
Loop 1:     for k from 2 to K do
                F_{k,0} ← F_{k,1} ← 1; F_{k,N+1} ← N;
Loop 2:         for t from 1 to ⌈log N⌉ do
Loop 3:             for i from 1 to 2^{t-1} do
                        j ← ⌈(2i-1)/2^t (N+1)⌉; l ← ⌈(i-1)/2^{t-1} (N+1)⌉; r ← ⌈i/2^{t-1} (N+1)⌉;
                        if M_{k,j} = ∞ then
Loop 4:                     for ℓ from F_{k,l} to F_{k,r} do
                                if M_{k-1,ℓ} + C_{ℓ+1,j} < M_{k,j} then
                                    M_{k,j} ← M_{k-1,ℓ} + C_{ℓ+1,j};
                                    F_{k,j} ← ℓ;
```

Fig. 1. The Dichotomic algorithm for the *K*-Uniform Allocation Problem.

$$C_{B_{K-1}+1,N} - C_{B_{K-1}+1,N-1} = \frac{1}{2}\left(p_N(N - B_{K-1}) + \sum_{i=B_{K-1}+1}^{N-1} p_i\right).$$

Thus, assuming, by contradiction, $B'_{K-1} < B_{K-1}$, one obtains:

$$C_{B'_{K-1}+1,N} - C_{B'_{K-1}+1,N-1} \geq C_{B_{K-1}+1,N} - C_{B_{K-1}+1,N-1}. \quad (4)$$

Subtracting (4) from (3) yields:

$$\begin{aligned} opt_{B'_{K-1},K-1} + C_{B'_{K-1}+1,N-1} &\leq opt_{B_{K-1},K-1} + C_{B_{K-1}+1,N-1} \\ &= opt_{N-1,K}, \end{aligned}$$

which contradicts the fact that $(B_1, B_2, \ldots, B_{K-1})$ is $LMO_{N-1,K}$. □

In words, Lemma 2 implies that, given the items sorted by nonincreasing 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^i_j denote the jth border of $LMO_{i,k}$, with $k > j \geq 1$.

Corollary 2. *Let the items $d_1, d_2, \ldots, d_N$ be sorted by nonincreasing probabilities and let $l < c < r \leq N$. Then, $B^l_{K-1} \leq B^c_{K-1} \leq B^r_{K-1}$.*

Proof. Follows directly from Lemma 2. □

3.1 The Dichotomic Algorithm

Corollary 2 plays a fundamental role in speeding up the DP algorithm. Indeed, assume that $LMO_{n,k-1}$ has been found for every $n \in [1, \ldots, N]$. 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^c_{k-1} is between B^l_{k-1} and B^r_{k-1}, for any $l \leq c \leq r$. Thus, the recurrence in (2) can be rewritten as:

$$opt_{c,k} = \min_{\ell \in \{B^l_{k-1}, \ldots, B^r_{k-1}\}} \{opt_{\ell,k-1} + C_{\ell+1,c}\}. \quad (5)$$

As the name suggests, the $O(KN \log N)$ time Dichotomic algorithm is derived by choosing $c = \lceil \frac{l+r}{2} \rceil$ in the recurrence of (5), thus obtaining:

$$opt_{\lceil \frac{l+r}{2} \rceil,k} = \min_{\ell \in \{B^l_{k-1}, \ldots, B^r_{k-1}\}} \{opt_{\ell,k-1} + C_{\ell+1,\lceil \frac{l+r}{2} \rceil}\}, \quad (6)$$

where B^l_{k-1} and B^r_{k-1} are, respectively, the final borders of $LMO_{l,k}$ and $LMO_{r,k}$.

In detail, the Dichotomic algorithm is shown in Fig. 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 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 an example, consider Fig. 2, which illustrates the execution of Loop 2 with $N = 15$ and $t = 3$, where the entries corresponding to $i = 1, 2, 3, 4$ of row k of matrix M are being computed. The fourth, eighth, and 12th entries have already been computed in stages 1 and 2. Let $F_{k,4}$, $F_{k,8}$, and $F_{k,12}$ be the final borders corresponding to the entries above. To compute the entry corresponding to $i = 1$, one only needs to consider entries from $M_{k-1,1}$ to $M_{k-1,F_{k,4}}$. Similarly, for $i = 2$, only the entries from $M_{k-1,F_{k,4}}$ to $M_{k-1,F_{k,8}}$ are to be examined. For $i = 3$, one examines the entries from $M_{k-1,F_{k,8}}$ up to $M_{k-1,F_{k,12}}$ and, finally, for $i = 4$, the entries beyond $M_{k-1,F_{k,12}}$ are visited.

Lemma 3. *The total number of comparisons involved in a stage of the Dichotomic algorithm is $O(N)$.*

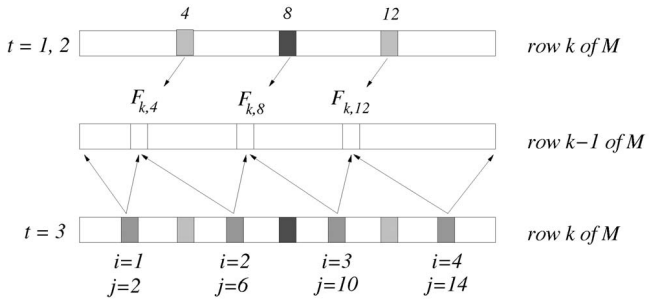

Fig. 2. Illustration of Loop 2 of algorithm Dichotomic.

Proof. The whole execution of Loop 3 of Fig. 1 corresponds to the execution of a stage for a particular value of t. The total number of comparisons involved is equal to the sum of the number of values the variable ℓ takes in Loop 3. This is equal to:

$$\sum_{i=1}^{2^{t-1}}(F_{k,r}-F_{k,l}+1),$$
$$\text{where } l=\left\lceil\frac{i-1}{2^{t-1}}(N+1)\right\rceil \text{ and } r=\left\lceil\frac{i}{2^{t-1}}(N+1)\right\rceil. \tag{7}$$

Therefore:

$$\begin{aligned}
&\sum_{i=1}^{2^{t-1}}(F_{k,\left\lceil\frac{i}{2^{t-1}}(N+1)\right\rceil}-F_{k,\left\lceil\frac{i-1}{2^{t-1}}(N+1)\right\rceil}+1)\\
&=(F_{k,\left\lceil\frac{N+1}{2^{t-1}}\right\rceil}-F_{k,0}+1)+\\
&\quad(F_{k,\left\lceil\frac{2}{2^{t-1}}(N+1)\right\rceil}-F_{k,\left\lceil\frac{N+1}{2^{t-1}}\right\rceil}+1)+\\
&\quad\vdots\\
&\quad(F_{k,\left\lceil\frac{2^{t-1}}{2^{t-1}}(N+1)\right\rceil}-F_{k,\left\lceil\frac{2^{t-1}-1}{2^{t-1}}(N+1)\right\rceil}+1)\\
&=F_{k,N+1}-F_{k,0}+2^{t-1}\\
&=N-1+2^{t-1}\\
&=O(N).
\end{aligned}$$

□

Theorem 1. *The K-Uniform Allocation Problem can be solved in $O(KN\log N)$ time by the Dichotomic algorithm.*

Proof. From Lemma 3, one stage of Fig. 1, corresponding to the execution of Loop 2 for a particular value of t, involves $O(N)$ comparisons. Since Loop 2 runs $\lceil\log N\rceil$ times and Loop 1 is repeated K times, the overall time complexity is $O(NK\log N)$. □

It is worth noting that, in the Dichotomic algorithm, the index values l, j, and r depend only on the stage t, precisely, on every value $1\le i\le 2^{t-1}$. In other words, the values of l, j, and r are the same for every k. Therefore, to save time, they can be computed only once during a preprocessing phase, for $1\le t\le\lceil\log N\rceil$ and $1\le i\le 2^{t-1}$, properly saving them on three arrays, say $l[t,i]$, $j[t,i]$, and $r[t,i]$, respectively. Such values can be directly read from the arrays when needed, without wasting time recomputing them $K-1$ times.

3.2 The Le4 Algorithm

In this subsection, the Le4 algorithm is proposed for the K-Uniform Allocation Problem when the number of channels K is at most four. When the data items are already sorted, the Le4 algorithm requires $O(N)$ time. It is based on an efficient incremental technique when there are two channels. Specifically, when $K=2$, adding a new item with lowest (or, highest) probability to an optimal partial solution can be done in $O(1)$ (respectively, $O(\log N)$) time.

When $K=2$, Lemma 2 can be further simplified. Indeed, in such a case, the final border can move at most one position to the right.

Lemma 4. *Let the items $d_1,d_2,\ldots,d_N$ be sorted by nonincreasing probabilities. Let $LMO_{N-1,2}=(B_1)$ and $LMO_{N,2}=(B'_1)$. Then, $B_1\le B'_1\le B_1+1$.*

Proof. Let $p_i^{(j)}$ be the probability of the ith item $d_i^{(j)}$ in G_j, with $j=1,2$. Since the items are sorted by nonincreasing order, then $p_1^{(j)}\ge p_2^{(j)}\ge\ldots\ge p_{N_j}^{(j)}$. Let $P_1=\sum_{i=1}^{N_1}p_i^{(1)}$ and $P_2=\sum_{i=1}^{N_2}p_i^{(2)}$ be, respectively, the sum of the probabilities of the items in the two groups of $LMO_{N-1,2}$. Assume by contradiction that, after adding d_N, $LMO_{N,2}=(B'_1)=(B_1+t)$ with $t>1$, that is, the final border of the optimal solution moves t positions to the right. Since the items $d_1^{(2)},d_2^{(2)},\ldots,d_t^{(2)}$ migrate from G_2 into G_1, then:

$$\begin{aligned}
opt_{N,2}=\frac{1}{2}\Bigg((N_1+t)(P_1+\sum_{i=1}^{t}p_i^{(2)})\\
+(N_2-t+1)(P_2-\sum_{i=1}^{t}p_i^{(2)}+p_N)\Bigg).
\end{aligned}\tag{8}$$

Similarly, if, after adding d_N, the final border of the optimal solution moves only one position to the right, the above equation becomes:

$$cost_{N,2}=\frac{1}{2}\Big((N_1+1)(P_1+p_1^{(2)})+N_2(P_2-p_1^{(2)}+p_N)\Big). \tag{9}$$

Since, by hypothesis, the solution that moves t positions is the optimal one, $opt_{N,2}\le cost_{N,2}$. Formally:

$$\begin{aligned}
opt_{N,2}-cost_{N,2}=\frac{1}{2}\Bigg((P_1-P_2)(t-1)+(N_1-N_2+2t-1)\\
\sum_{i=2}^{t}p_i^{(2)}+(t-1)(2p_1^{(2)}-p_N)\Bigg)\le 0.
\end{aligned}\tag{10}$$

Consider now the solution obtained from $LMO_{N-1,2}$ by moving the final border $t-1$ positions on the right. Its cost differs from $opt_{N-1,2}$ by

$$\Delta=\frac{1}{2}\left((P_1-P_2)(t-1)+(N_1-N_2+2t-2)\sum_{i=1}^{t-1}p_i^{(2)}\right), \tag{11}$$

which is greater than or equal to 0 from the optimality of $LMO_{N-1,2}$.

Substituting Δ in (10), one gets:

$$opt_{N,2} - cost_{N,2} = \Delta + \frac{1}{2}\left(\sum_{i=2}^{t-1} p_i^{(2)} + (N_1 - N_2)(p_t^{(2)} - p_1^{(2)}) + (2t-1)p_t^{(2)} - (t-1)p_N\right). \quad (12)$$

Observe that $(N_1 - N_2)(p_t^{(2)} - p_1^{(2)}) \geq 0$ since $N_1 \leq N_2$ and $p_t^{(2)} \leq p_1^{(2)}$ and that $(2t-1)p_t^{(2)} - (t-1)p_N \geq 0$ because $2t - 1 > t - 1$ and $p_t^{(2)} \geq p_N$. Recalling that $\Delta \geq 0$ and $\sum_{i=2}^{t-1} p_i^{(2)} \geq 0$, then $opt_{N,2} - cost_{N,2} \geq 0$ results, thus contradicting the optimality of the solution with N items, where the last border moves t positions on the right. □

As a consequence of the above lemma, computing $LMO_{n,2}$ given $LMO_{n-1,2} = (B_1^{n-1})$ can be done in constant time just applying the following recurrence:

$$opt_{n,2} = \min_{\ell \in \{B_1^{n-1}, B_1^{n-1}+1\}} \{C_{1,\ell} + C_{\ell+1,n}\}. \quad (13)$$

Therefore, the following theorem holds:

Theorem 2. *Let the items $d_1, d_2, \ldots, d_N$ be sorted by nonincreasing probabilities. All the solutions $LMO_{n,2}$, with $1 \leq n \leq N$, of the 2-Uniform Allocation Problem can be computed in $O(N)$ time.*

The above result leads to an efficient algorithm for finding the optimal solution $LMO_{N,3}$ of the 3-Uniform Allocation Problem. Indeed, it is easy to see that the solution for $K = 3$ can be obtained by combining the solutions for $K = 2$ and $K = 1$ as follows:

$$opt_{N,3} = \min_{\ell \in \{1,\ldots,N\}} \{opt_{\ell,2} + C_{\ell+1,N}\}. \quad (14)$$

Corollary 3. *Let the items $d_1, d_2, \ldots, d_N$ be sorted by nonincreasing probabilities. The optimal solution $LMO_{N,3}$ of the 3-Uniform Allocation Problem can be computed in $O(N)$ time.*

Following a similar reasoning, the 4-Uniform Allocation Problem can be solved by combining the solutions of two problems with $K = 2$ for, respectively, the first n items and the remaining $N - n$ items. Theorem 2 showed how to solve, in $O(N)$ time, all the problems for the first n items, with $1 \leq n \leq N$. In order to apply the same technique to solve, in $O(N)$ time, all the problems for the remaining $N - n$ items, a result similar to Lemma 4 is needed when the new item to be added is that with the greatest probability.

In the rest of this subsection, the notation is slightly modified in order to consider both the above problems. Specifically, consider the 2-Uniform Allocation Problem. Let $opt_{i,j,2}$ denote the cost of the leftmost optimal solution $LMO_{i,j,2}$ for allocating the items $d_i, \ldots, d_j$ to two channels.

Lemma 5. *Let the items $d_1, d_2, \ldots, d_N$ be sorted by nonincreasing probabilities. Let $LMO_{2,N,2} = (B_1)$ and $LMO_{1,N,2} = (B_1')$. Then, $B_1' \leq B_1$.*

Proof. Similar to Lemma 2. □

For the aim of determining the exact index of the final border B_1' of $LMO_{1,N,2}$, consider the feasible solutions obtained by inserting d_1 into G_1 and moving left the border B_1 of $LMO_{2,N,2}$ one position at a time. Continue to move left B_1 while the cost of the resulting feasible solution decreases, but stop moving and fix $B_1' = B_1$ as soon as its cost starts increasing. The following lemma guarantees that the thus found B_1' is optimal.

Lemma 6. *Let the items $d_1, d_2, \ldots, d_N$ be sorted by nonincreasing probabilities. Let $S_{n,N,2} = (B)$ and $S'_{n,N,2} = (B-1)$ be feasible solutions such that their costs are increasing, that is, $s_{n,N,2} < s'_{n,N,2}$. Then, for $S''_{n,N,2} = (B-2)$, its cost $s''_{n,N,2} > s'_{n,N,2}$.*

Proof. As in the proof of Lemma 4, let $p_i^{(j)}$ be the probability of the ith item $d_i^{(j)}$ in G_j, with $j = 1, 2$, and let $P_1 = \sum_{i=1}^{N_1} p_i^{(1)}$ and $P_2 = \sum_{i=1}^{N_2} p_i^{(2)}$ of $S_{n,N,2}$. By definition, $s_{n,N,2} = \frac{1}{2}(N_1 P_1 + N_2 P_2)$ and

$$s'_{n,N,2} = \frac{1}{2}\left((N_1 - 1)(P_1 - p_{N_1}^{(1)}) + (N_2 + 1)(P_2 + p_{N_1}^{(1)})\right).$$

Thus, $s'_{n,N,2} > s_{n,N,2}$ if and only if

$$p_{N_1}^{(1)} > \frac{P_1 - P_2}{N_2 - N_1 + 2}. \quad (15)$$

Note that (15) holds true since, by hypothesis, $s'_{n,N,2} > s_{n,N,2}$. Moreover, rewriting the above equation for $s''_{n,N,2}$ and $s'_{n,N,2}$ implies that $s''_{n,N,2} > s'_{n,N,2}$ if and only if

$$p_{N_1-1}^{(1)} > \frac{P_1 - P_2 - 2p_{N_1}^{(1)}}{N_2 - N_1 + 4}. \quad (16)$$

To complete the proof, note that (16) also holds true because

$$p_{N_1-1}^{(1)} > p_{N_1}^{(1)} > \frac{P_1 - P_2}{N_2 - N_1 + 2} > \frac{P_1 - P_2 - 2p_{N_1}^{(1)}}{N_2 - N_1 + 4}.$$

□

As a consequence of the above lemma, given $LMO_{n,N,2} = (B_1^n)$, $LMO_{n-1,N,2}$ can be computed just applying the following recurrence:

$$opt_{n-1,N,2} = \min_{\ell \in \{n-1,\ldots,B_1^n\}} \{C_{1,\ell} + C_{\ell+1,n}\}. \quad (17)$$

Note that, in (17), a single $opt_{n-1,N,2}$ can be found in $O(\log(B_1^n - n))$ time by applying a binary search in the range $[n-1, \ldots, B_1^n]$. However, $opt_{n,N,2}$ for all $1 \leq n \leq N$ can be found in linear time.

Theorem 3. *Let the items $d_1, d_2, \ldots, d_N$ be sorted by nonincreasing probabilities. All the solutions $LMO_{n,N,2}$, with $1 \leq n \leq N$, of the 2-Uniform Allocation Problem can be computed in $O(N)$ time.*

Proof. Consider the sequence of solutions $LMO_{N-1,N,2} = (B_1^{N-1})$, $LMO_{N-2,N,2} = (B_1^{N-2}), \ldots, LMO_{1,N,2} = (B_1^1)$. By Lemma 6, the overall number of comparisons is $O(\sum_{n=1}^{N-1}(B_1^{n+1} - B_1^n)) = O(N)$. □

Theorems 2 and 3 yield an efficient way of finding the optimal solution $LMO_{N,4}$ of the 4-Uniform Allocation Problem by combining two solutions for $K = 2$:

$$opt_{1,N,4} = \min_{\ell \in \{1,\ldots,N\}} \{opt_{1,\ell,2} + opt_{\ell+1,N,2}\}. \quad (18)$$

Corollary 4. *Let the items $d_1, d_2, \ldots, d_N$ be sorted by nonincreasing probabilities. The optimal solution $LMO_{N,4}$ of the 4-Uniform Allocation Problem can be found in $O(N)$ time.*

The Le4 algorithm solves the Uniform Allocation Problem with at most four channels by cases. Specifically, it applies the recurrences in (13), (14), or (18), for $K = 2, 3$, and 4, respectively. If the items are already sorted, its time complexity is $O(N)$; otherwise, it takes $O(N \log N)$ time.

4 Nonuniform 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 4. *The K-Non-Uniform Allocation Problem is strong NP-hard.*

Proof. See the Appendix. □

As a consequence of the above result, there is no pseudopolynomial 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.

Recall that the sum of the item lengths in group G_j is denoted by Z_j. The following result generalizes Lemma 1.

Lemma 7. *Let G_h and G_j be two groups in an optimal solution. 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 $Z_h \leq Z_j$.*

Proof. Assume by contradiction that $Z_h < Z_j$ and $p_i < p_k$ and swap d_i and d_k between the two groups. Since both items have the same length, Z_h and Z_j do not change. Instead, the overall change in cost is:

$$\begin{aligned}\Delta &= \frac{1}{2}\big(Z_h(p_k - p_i) + Z_j(p_i - p_k)\big) \\ &= \frac{1}{2}\big((Z_h - Z_j)(p_k - p_i)\big) < 0,\end{aligned}$$

which is a contradiction. □

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.

Corollary 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 $d_b^i \in G_j$.*

Proof. Follows from Lemma 7. □

In the following, the items in each D_i are assumed to be sorted by nonincreasing probabilities and optimal solutions will be sought of the form:

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

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:

$$opt_{n_1,\ldots,n_z,k} = \min_{\substack{\bar{\ell}=(\ell_1,\ldots,\ell_z) \\ 0 \leq \ell_i \leq n_i, 1 \leq i \leq z}} \Big\{opt_{\ell_1,\ldots,\ell_z,k-1} + C_{\ell_1+1,n_1,\ldots,\ell_z+1,n_z}\Big\}. \quad (19)$$

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 $k = 1, \ldots, K-1$. A dynamic programming algorithm, called Optimal, derives directly from the recurrence in (19).

Theorem 5. *The K-Non-Uniform Allocation Problem can be solved in $O(KN^{2z})$ time.*

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

4.1 Two Channels

Now, consider a special case of the *K*-Non-Uniform Allocation Problem where the number of channels is equal to 2.

Theorem 6. *The 2-Non-Uniform Allocation Problem is NP-hard.*

Proof. See the Appendix. □

Although the 2-Non-Uniform Allocation Problem is *NP*-hard, it is not *NP*-hard in the strong sense. Therefore, it is possible to devise a *pseudopolynomial* time algorithm, that is an algorithm whose time is polynomial in the item lengths.

The problem is to find a solution G_1 and G_2 such that $\frac{1}{2}(Z_1P_1 + Z_2P_2)$ is minimized, where P_1 and P_2 denote the sum of the demand probabilities of items in G_1 and G_2, respectively. From now on, let $P_1 + P_2 = 1$ and $Z_1 + Z_2 = Z$ and assume, without loss of generality, that $Z_1 \le Z_2$. Observe that there are only $\lfloor Z/2 \rfloor$ possible values for Z_1.

Consider the 2-Non-Uniform Allocation Problem for a fixed value of Z_1. Observe that

$$Z_1P_1 + Z_2P_2 = Z_1P_1 + Z_2(1-P_1) = P_1(Z_1 - Z_2) + Z_2.$$

Since Z_1 is fixed and, hence, Z_2 is also fixed and, observing that $Z_1 - Z_2 \le 0$, minimizing $Z_1P_1 + Z_2P_2$ is equivalent to maximizing P_1. Therefore, the problem reduces to finding a subset G_1 of $\{d_1, d_2, \ldots, d_N\}$, which maximizes P_1. The basic idea of the algorithm to be proposed is that, once the value of Z_1 is fixed, the 2-Non-Uniform Allocation Problem with N items $\{d_1, d_2, \ldots, d_N\}$ reduces to a particular *Knapsack problem* [11] of capacity Z_1 with the same N items, where each item d_i is characterized by a *profit* p_i and a *weight* z_i. Specifically, the Knapsack problem consists of finding a subset S of $\{d_1, d_2, \ldots, d_N\}$ subject to the constraint $\sum_{d_k \in S} z_k = Z_1$ so as to maximize the objective function $\sum_{d_k \in S} p_k$.

To apply dynamic programming, consider two $(N+1) \times (\lfloor Z/2 \rfloor + 1)$ matrices M and X. The entry $M_{i,j}$, with $0 \le i \le N$ and $0 \le j \le \lfloor Z/2 \rfloor$, stores the value of the objective function for the above Knapsack problem with items $\{d_1, \ldots, d_i\}$ and capacity j. Formally, $M_{i,j} = \max \sum_{d_k \in S} p_k$ such that $\sum_{d_k \in S} z_k = j$, where $S \subseteq \{d_1, \ldots, d_i\}$. By definition, let $M_{i,j} = -\infty$ if the capacity j cannot be completely filled by any S. The Boolean entry $X_{i,j}$ records whether the item d_i has been selected or not in the solution of the Knapsack problem with items $\{d_1, \ldots, d_i\}$ and capacity j, with $0 \le i \le N$ and $0 \le j \le \lfloor Z/2 \rfloor$.

The dynamic programming algorithm starts by initializing the first row of the matrices in such a way that $M_{0,0} = 0$, $M_{0,j} = -\infty$ for $1 \le j \le \lfloor Z/2 \rfloor$, and $X_{0,j} = \texttt{false}$ for $0 \le j \le \lfloor Z/2 \rfloor$. Then, the matrices M and X are filled row by row as follows: For $i = 1, 2, \ldots, N$ and $j = 0, 1, \ldots, \lfloor Z/2 \rfloor$, $M_{i,j}$ and $X_{i,j}$ are filled by using the following relations:

$$M_{i,j} = \begin{cases} M_{i-1,j} & \text{if } j < z_i \\ \max\{M_{i-1,j},\ M_{i-1,j-z_i} + p_i\} & \text{if } j \ge z_i \end{cases}$$

$$X_{i,j} = \begin{cases} \texttt{true} & \text{if } M_{i,j} = M_{i-1,j-z_i} + p_i \ne -\infty \\ \texttt{false} & \text{otherwise.} \end{cases}$$

Note that it is possible that, for certain values of j with $0 \le j \le \lfloor Z/2 \rfloor$, there is no solution for items $\{d_1, \ldots, d_i\}$ such that the total sum of weights is exactly j. In such cases, according to the definition, the recurrence above gives $M_{i,j} = -\infty$. In contrast, if there is a solution for items $\{d_1, \ldots, d_i\}$ such that the total sum of weights is exactly j, then $M_{i,j} \ne -\infty$ and $M_{i,j}$ gives the optimal value of the objective function.

Consider the last row of M. Any entry $M_{N,j} \ne -\infty$ gives the optimal P_1 for the 2-Non-Uniform Allocation Problem with items $\{d_1, \ldots, d_N\}$ and $Z_1 = j$. Therefore, the entry, say $M_{N,\bar{j}}$, which minimizes $\frac{1}{2}(M_N, \bar{j}\bar{j}(1 - M_{N,\bar{j}})(Z - \bar{j}))$ gives the optimal AED for the original 2-Non-Uniform Allocation Problem. Once $M_{N,\bar{j}}$ has been found, it is easy to list out the items which have been picked up in the optimal solution by tracing back the solution path. Specifically, if $X_{N,\bar{j}} = \texttt{true}$, then item d_N is selected and the entry $X_{N-1,\bar{j}-z_N}$ is examined next; if $X_{N,\bar{j}} = \texttt{false}$, then item d_N is not selected and the entry $X_{N-1,\bar{j}}$ is examined next. Such a procedure is repeated backward until the row 0 of X is reached. The selected items are assigned to channel 1, while the remaining items to channel 2.

Theorem 7. *The 2-Non-Uniform Allocation Problem can be solved in $O(NZ)$ time.*

Proof. The matrices M and X have $(N+1) \times (\lfloor Z/2 \rfloor + 1)$ entries. Each entry can be computed in constant time. Moreover, the minimum on the last row of M costs $O(Z)$ time, while tracing back the solution path takes $O(N)$ time. Hence, the time complexity of the dynamic programming algorithm is $O(NZ)$. □

The algorithm shown in Fig. 3 is effective when the items have small length. For instance, if each item length is bounded by a constant, then $Z = O(N)$ and the overall time becomes $O(N^2)$. Such an algorithm is as effective as the standard pseudopolynomial time algorithm for the Knapsack problem and allows Fully Polynomial Time Approximation Schemes (FPTAS) to be obtained by standard techniques [11].

4.2 The Move1 Heuristic

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 subsection, where a new heuristic, called *Move1*, is presented for the *K*-Non-Uniform Allocation Problem. Move1 is similar to a local search since it iteratively considers a solution and examines a set of neighbor solutions which can be obtained from the previous one by means of a transformation. Unlike local search, however, Move1 always replaces the previous solution with the best solution in the neighborhood, even if the new selected

```
Input:          N items sorted by non-increasing p_i/z_i, and K groups;
Initialize:     for i from 1 to K − 1 do
                    B*_i ← B_i ← i;
                global-AED ← AED(B*_1, ..., B*_{K−1});
Iteration:      for j from 1 to N − K + 1 do
                    local-AED ← ∞;
                    for i from 1 to K − 1 do
Compute S_i:            B_{K−i} ← B_{K−i} + 1;
                        if AED(B_1, ..., B_{K−1}) < local-AED then
Find best S_i:              best ← K − i;
                            local-AED ← AED(B_1, ..., B_{K−1});
Update SOL:         for i from best − 1 downto 1 do
                        B_i ← B_i − 1;
Update SOL*:        if local-AED < global-AED then
                        global-AED ← local-AED;
                        for i from 1 to K − 1 do
                            B*_i ← B_i;
Output:         return SOL* = (B*_1, B*_2, ..., B*_{K−1});
```

Fig. 3. The Move1 heuristic for the *K*-Non-Uniform Allocation Problem.

solution is not better than the previous one. The output is the best solution among the solutions considered in all the above iterations.

As for the previously known Greedy heuristic, Move1 also assumes that the items are sorted by nonincreasing $\frac{p_i}{z_i}$ ratios. Initially, Move1 assigns all the N items to the K channels in such a way that each of the first $K-1$ channels holds just one item, while the last channel holds all the remaining $N-K+1$ items. In other words, according to the segmentation notation introduced in Section 2, the starting solution SOL is represented by $(B_1 = 1, B_2 = 2, \ldots, B_{K-1} = K-1)$.

Consider a generic iteration with

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

Move1 tries to shift right, by one position, the rightmost borders. Precisely, it considers the $K-1$ solutions $S_1, S_2, \ldots, S_{K-1}$, where S_i is derived from SOL by moving the rightmost i borders. Namely,

$$S_i = (B_1, B_2, \ldots, B_{K-i-1}, B_{K-i}+1, \ldots, B_{K-1}+1),$$

for $1 \leq i \leq K-1$. Move1 computes the AED for each S_i and then replaces SOL with that S_i having the lowest AED. Note that, in the new SOL, the group size N_K of items assigned to channel K decreases by one.

The above iteration is repeated $N-K+1$ times until the group size N_K becomes 1. The solution $SOL^* = (B_1^*, B_2^*, \ldots, B_{K-1}^*)$ output by the Move1 algorithm is that with the lowest AED among the $N-K+1$ different values of SOL found during the above iterations.

The algorithm is detailed in Fig. 3. The outer loop is iterated $O(N-K)$ times, while each of the two inner loops is repeated $O(K)$ times. In the first (second) inner loop, the borders are shifted right (left, respectively) by one position. In doing this, both the group length $\sum_{h=B_{i-1}+1}^{B_i} z_h$ and group probability $\sum_{h=B_{i-1}+1}^{B_i} p_h$ can be easily updated in $O(1)$ time (where it is assumed that $B_0 = 0$ and $B_K = N$). Indeed, it is sufficient to maintain two prefix sums sequences and update them only when a border shifts (see, e.g., [17] for details about this technique). In this way, evaluating $\mathrm{AED}(B_1, \ldots, B_{K-1})$ requires $O(1)$ time as well. Since $O(N \log N)$ time is needed for sorting, the overall time complexity is $O(N(K + \log N))$. If the items are already sorted, such a time complexity becomes $O(K(N-K))$, which in turn is $O(N)$ when K is a constant.

The Move1 heuristic follows a completely different approach with respect to the Greedy heuristic, which instead starts with a single segmentation where all the N items are assigned to a single group. Then, for $K-1$ times, one of the groups is split into two groups that will be assigned to two different channels. To find which group to split along with its actual split point, Greedy considers as split point candidates all the possible points of all groups, and selects the one that decreases AED the most. 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. All the details can be found in [17].

5 Experimental Tests

In this section, experimental results, performed on implementations of both the Greedy and Move1 heuristics, are discussed for the *K*-Non-Uniform Allocation Problem. In particular, the implementation of Greedy as detailed in [17] is used. The algorithms are written in C and the experiments are run on a Pentium III, 550 Mhz, with 2 GB RAM.

The heuristics are tested on some nonuniform length instances generated as follows: 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 generated according to a uniform distribution in the range $1 \leq z_i \leq z$, as in [15]. The items are sorted by nonincreasing $\frac{p_i}{z_i}$ ratios, as suggested in [15]. The parameters N, K, z, and θ vary, respectively, in the ranges: $50 \leq N \leq 2,000$, $3 \leq K \leq 1,000$, $3 \leq z \leq 10$, and $0.3 \leq \theta \leq 0.9$.

Since the Optimal algorithm can find optimal solutions in a reasonable time only for small values of N and z, a lower bound on AED is used for large values of N and z. The lower bound for a nonuniform instance is obtained by transforming it into a uniform instance as follows: Each

TABLE 2
Experimental Results for the Greedy and Move1 Heuristics when $N = 50$ and $z = 3$

$N/K/\theta/z$	Algorithm	AED	% Error	Time
50/3/0.3/3	Greedy	18.67	15.6	27
	Move1	16.67	3.2	101
	Optimal	16.15		525902
50/3/0.6/3	Greedy	16.65	13.9	29
	Move1	15.56	6.5	92
	Optimal	14.61		812037
50/3/0.9/3	Greedy	13.98	11.3	29
	Move1	13.63	8.6	83
	Optimal	12.55		296776
50/5/0.3/3	Greedy	10.48	7.4	39
	Move1	9.82	0.7	205
	Optimal	9.75		822933
50/5/0.6/3	Greedy	10.30	19.4	36
	Move1	9.41	9.5	191
	Optimal	8.62		661980
50/5/0.9/3	Greedy	9.62	23.0	38
	Move1	8.65	10.6	178
	Optimal	7.82		766532

TABLE 4
Experimental Results for the Greedy and Move1 Heuristics when $N = 2,000$, $\theta = 0.8$, and $z = 3$

$N/K/\theta/z$	Algorithm	AED	% Error	Time
2000/4/0.8/3	Greedy	368.82	6.5	522
	Move1	368.27	6.4	5277
	lower bound	346.07		
2000/50/0.8/3	Greedy	32.92	24.5	1204
	Move1	28.34	7.1	109072
	lower bound	26.44		
2000/100/0.8/3	Greedy	15.90	22.5	1561
	Move1	13.90	7.1	216221
	lower bound	12.97		
2000/250/0.8/3	Greedy	6.05	17.0	2935
	Move1	5.61	8.5	528124
	lower bound	5.17		
2000/500/0.8/3	Greedy	3.07	17.6	6803
	Move1	2.94	12.6	939706
	lower bound	2.61		
2000/1000/0.8/3	Greedy	1.63	25.4	22271
	Move1	1.59	22.3	1264056
	lower bound	1.30		

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 thus 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. It is worth noting that such a lower bound is not necessarily achievable. Its goodness decreases as the number K of channels increases with respect to N since, in such a case, more freedom is introduced.

The simulation results are exhibited in Tables 2, 3, and 4. The tables report the time (measured in microseconds), the AED, and the percentage of error, which is computed as

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

When $\mathrm{AED}_{\mathrm{optimal}}$ is not known, it is replaced by its lower bound. By observing the tables, one notes that Move1 always outperforms Greedy in terms of the solution quality. In particular, Table 2 shows the AED, error, and time when $K = 50$ and $z = 3$, with θ assuming the values 0.3, 0.6, and 0.9. Since N and z are small, the optimal solutions, computed by the Optimal algorithm, are reported. With regard to the running time, both Move1 and Greedy always take less than one millisecond, while Optimal is much slower. By observing Table 2, one notes that Move1 gives solutions closer to the optimal ones when θ is small. In the best instance ($K = 5$ and $\theta = 0.3$), the error is 0.2 percent only, while, in the worst instance ($K = 5$ and $\theta = 0.9$), the error is about 10 percent. In addition, Table 3 shows the AED, error, and time when $K = 500$, $\theta = 0.8$, $z = 10$, and $10 \leq K \leq 300$. The running time of Move1 is less than 1/10th of a second, while that of Greedy is less than 10 milliseconds. Since N and z are not small, the lower bounds are reported. Although Move1 always finds better solutions than Greedy, the resulting error is larger than in the previous table. However, it is worth noting that the lower bound degrades as the number of channels increases. Therefore, the error of both algorithms is actually smaller than that reported in Table 3, especially when K approaches N. The same observation holds for the AED and error reported in Table 4, where $K = 2,000$, $\theta = 0.8$, $z = 3$, and $4 \leq K \leq 1,000$. Indeed, one notes a smaller error when $K = 4$. For large K, the running time of Move1 becomes almost one second, while that of Greedy is always less than 1/10th of a second.

TABLE 3
Experimental Results for the Greedy and Move1 Heuristics when $N = 500$, $\theta = 0.8$, and $z = 10$

$N/K/\theta/z$	Algorithm	AED	% Error	Time
500/10/0.8/10	Greedy	100.01	9.2	235
	Move1	94.68	3.3	5057
	lower bound	91.57		
500/50/0.8/10	Greedy	21.51	16.9	391
	Move1	19.62	6.6	26133
	lower bound	18.40		
500/100/0.8/10	Greedy	10.54	18.5	616
	Move1	9.78	10.0	47564
	lower bound	8.89		
500/150/0.8/10	Greedy	7.56	19.6	906
	Move1	7.24	14.5	63332
	lower bound	6.32		
500/200/0.8/10	Greedy	5.39	19.7	1290
	Move1	5.25	16.6	74914
	lower bound	4.50		
500/300/0.8/10	Greedy	4.56	27.0	2388
	Move1	4.36	21.4	71771
	lower bound	3.59		

For the sake of completeness, experimental tests are also performed on some uniform length benchmarks. In addition to Move1 and Greedy, the Dichotomic and Le4 algorithms are also run in order to check their time improvements over the DP algorithm not only analytically, but also experimentally. In particular, the DP algorithm is also implemented as described by its authors. The uniform length benchmarks are built as described in [17], [18], where the probabilities are generated according to a Zipf distribution with $\theta = 0.8$, while the parameters N and K are chosen so as to vary, respectively, in the ranges $4 \leq K \leq 2,500$ and $10 \leq N \leq 5,000$. The fixed values of K and N are, respectively, 4 and 2,500.

The results of the simulations are reported in Tables 5 and 6, where the running times do not include the time for sorting. Observing such tables, one notes that Move1 always finds the optimal solutions, while Greedy gives suboptimal solutions which are less than 3 percent far from

TABLE 5
Experimental Results for the DP, Dichotomic, Le4, Move1, and Greedy Algorithms when $K = 4$, $\theta = 0.8$, and $z = 1$

$N/K/\theta/z$	Algorithm	AED	% Error	Time
10/4/0.8/1	DP	1.17		267
	Dichotomic	1.17		82
	Le4	1.17		18
	Move1	1.17	0	23
	Greedy	1.17	0	21
500/4/0.8/1	DP	47.53		32254
	Dichotomic	47.53		2624
	Le4	47.53		100
	Move1	47.53	0	488
	Greedy	47.77	1.6	21
1000/4/0.8/1	DP	92.82		377365
	Dichotomic	92.82		5767
	Le4	92.82		186
	Move1	92.82	0	953
	Greedy	93.34	0.5	253
2000/4/0.8/1	DP	181.80		584836
	Dichotomic	181.80		12749
	Le4	181.80		362
	Move1	181.80	0	1877
	Greedy	183.00	0.6	452
3000/4/0.8/1	DP	269.73		1252477
	Dichotomic	269.73		20133
	Le4	269.73		620
	Move1	269.73	0	2922
	Greedy	271.64	0.7	738
4000/4/0.8/1	DP	357.04		2132911
	Dichotomic	357.04		26545
	Le4	357.04		690
	Move1	357.04	0	3781
	Greedy	359.71	0.7	888
5000/4/0.8/1	DP	443.92		3382007
	Dichotomic	443.92		39126
	Le4	443.92		1035
	Move1	443.92	0	4672
	Greedy	447.34	0.7	1238

TABLE 6
Experimental Results for the DP, Dichotomic, Move1, and Greedy Algorithms when $N = 2,500$, $\theta = 0.8$, and $z = 1$

$N/K/\theta/z$	Algorithm	AED	% Error	Time
2500/4/0.8/1	DP	225.86		844699
	Dichotomic	225.86		17682
	Le4	225.86		410
	Move1	225.86	0	2446
	Greedy	227.42	0.6	563
2500/250/0.8/1	DP	3.38		81365208
	Dichotomic	3.38		1394727
	Move1	3.38	0	217704
	Greedy	3.45	2.0	3358
2500/500/0.8/1	DP	1.71		176531621
	Dichotomic	1.71		2834693
	Move1	1.71	0	391633
	Greedy	1.75	2.3	7315
2500/1000/0.8/1	DP	0.91		287194065
	Dichotomic	0.91		5812711
	Move1	0.91	0	613077
	Greedy	0.93	2.1	23551
2500/1500/0.8/1	DP	0.66		371107406
	Dichotomic	0.66		7531264
	Move1	0.66	0	606480
	Greedy	0.68	3.0	46852
2500/2000/0.8/1	DP	0.55		442640777
	Dichotomic	0.55		9999065
	Move1	0.55	0	392988
	Greedy	0.56	1.8	80219
2500/2500/0.8/1	DP	0.50		477388359
	Dichotomic	0.50		12501690
	Move1	0.50	0	663
	Greedy	0.50	0	122889

the optimum. Note that, when $N = K = 2,500$, Move1 is extremely fast because its initial solution, which has one item assigned to each channel, is the optimal solution and no extra work is needed. Moreover, observing Table 5, one notes that the Le4 algorithm is very fast. For instance, it requires about 1 millisecond for solving the problem on $N = 5,000$ and $K = 4$. Also observing Table 6, one checks that the Dichotomic algorithm is between one and two orders of magnitude faster than the DP algorithm. For instance, when $N = 2,500$ and $K = 1,500$, the DP algorithm requires about six minutes, while the Dichotomic algorithm just takes about seven seconds.

In conclusion, Move1 always achieves a better quality of the solution with respect to Greedy, both in the uniform and nonuniform cases, at the expense of a slightly larger running time. Therefore, the choice between such two heuristics depends on the goal to be pursued. If one is interested in finding the lowest suboptimal solution, then Move1 should be adopted. However, if adaptability to parameter changes is the priority, then Greedy should be applied. For instance, Greedy scales well with the number of channels (adding/removing a channel requires doing/undoing a single split), while Move1 is designed for a fixed number of channels.

6 Conclusions

In this paper, 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 nonuniform length problems were solved to the optimum, proposing new algorithms based on dynamic programming. All the results presented in this paper are summarized in Table 7. In particular, for nonuniform lengths, it has been shown that the problem is computationally intractable. Therefore, a new heuristic has been proposed, which experimentally outperforms the previously known heuristic in terms of the solution quality.

As a direction for further research, one can derive lower bounds on the time complexity for the uniform case. Moreover, one could try to design $O(N)$ time algorithms in the uniform case when the number K of channels is a constant greater than 4 and the items are already sorted. Finally, for the nonuniform case, one could search for higher lower bounds on the AED and for faster heuristics which can provide better suboptimal solutions and can easily adapt to dynamic changes in the item popularities.

In this paper, 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 AED and keeps the tuning time as low as possible 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 global information, the relative position of each data item within the set of all

TABLE 7
New Results for Broadcasting N Data Items on K Channels with Skewed Allocation and Flat Scheduling

♯ Channels	Item Lengths	Complexity	Solution	Algorithm	Time
≤ 4	uniform	P	optimal	Le4	$O(N \log N)$
K	uniform	P	optimal	Dichotomic	$O(NK \log N)$
2	non-uniform	NP-hard	optimal	Knapsack	$O(NZ)$
K	non-uniform	strong NP-hard	optimal	Optimal	$O(KN^{2z})$
			heuristic	Move1	$O(N(K + \log N))$

Z is the sum of the data lengths and z is the maximum data length. When the items are already sorted, the time of Le4 reduces to $O(N)$, while that of Move1 becomes $O(K(N-K))$, which in turn is $O(N)$ if K is a constant.

data items sorted by probabilities. This is an assumption that precludes the solution from being dynamically adapted to changes in the item parameters, like probabilities, sizes, etc. To overcome this drawback, new solutions can be sought that, without using global information on data items, either mix index and data items within the same channels [10] or optimize the index broadcasting on dedicated channels [14].

APPENDIX A

A.1 Proof of Theorem 4

In order to prove that the *K*-Non-Uniform Allocation Problem is strong *NP*-hard, consider its corresponding decision problem:

***K*-NON-UNIFORM ALLOCATION**
INSTANCE: A set $D = \{d_1, d_2, \ldots, d_N\}$ of items, a positive integer K, a length $z_i \in \mathbb{Z}^+$, and a demand probability $p_i \in \mathbb{R}^+$ for each d_i, with $1 \leq i \leq N$, and a bound $C \in \mathbb{R}^+$.
QUESTION: Can D be partitioned into K groups $G_1, \ldots, G_K$ such that $\sum_{j=1}^{K}((\sum_{d_i \in G_j} z_i)(\sum_{d_i \in G_j} p_i)) \leq C$?

In the following, it is proven that *K*-NON-UNIFORM ALLOCATION is strong *NP*-hard by exhibiting a polynomial time reduction from 3-PARTITION [6].

3-PARTITION
INSTANCE: A set A of $3m$ elements, a bound $B \in \mathbb{Z}^+$, and a size $s(a) \in \mathbb{Z}^+$ for each $a \in A$ such that $s(a)$ satisfies $\frac{B}{4} < s(a) < \frac{B}{2}$ and such that $\sum_{a \in A} s(a) = mB$.
QUESTION: Can A be partitioned into m disjoint sets $S_1, S_2, \ldots, S_m$ such that, for all $1 \leq i \leq m$, $\sum_{a \in S_i} s(a) = B$? (Every set S_i must contain exactly three elements from A.)

Given an instance of 3-PARTITION, the corresponding instance of *K*-NON-UNIFORM ALLOCATION is built by setting $D = A$, $K = m$, $C = 4B$, $z_i = B + s(a_i)$ and $p_i = \frac{z_i}{4mB}$, for $i = 1, \ldots, 3m$.

Consider a "yes" instance of 3-PARTITION, i.e., an instance for which there exist m disjoint subsets of A, each of three elements whose sum is B. Consider the solution of *K*-NON-UNIFORM ALLOCATION where the m triplets correspond to the $K = m$ groups. For each group, the sum of the item lengths is $4B$, while the sum of the item probabilities is $\frac{1}{m}$. Hence, the total cost of this partitioning is $4B = C$. Therefore, the resulting instance of *K*-NON-UNIFORM ALLOCATION is a "yes" instance.

Conversely, consider a solution for *K*-NON-UNIFORM ALLOCATION whose cost is exactly $4B$. Note that $4B$ is the minimum cost. Indeed, recalling that $Z_j = \sum_{d_i \in G_j} z_i$, the overall cost can be written as

$$\frac{1}{4mB}\sum_{j=1}^{m} Z_j^2 \geq \frac{1}{4m^2B}\left(\sum_{j=1}^{m} Z_j\right)^2 = \frac{1}{4m^2B}\left(\sum_{i=1}^{3m} B + s(a_i)\right)^2 = 4B,$$

where $\sum_{j=1}^{m} Z_j^2 \geq \frac{1}{m}(\sum_{j=1}^{m} Z_j)^2$ follows from the Cauchy-Schwartz inequality in $\mathbb{Z}_m$ [16].

Now, it is shown that each group has exactly three items. Indeed, assume by contradiction that there is a group G_p with $|G_p| \leq 2$, which implies that there is also a group G_q with $|G_q| \geq 4$. Let Z_p and Z_q be the sum of item lengths in G_p and G_q, respectively. Since $\frac{B}{4} < s(a_i) < \frac{B}{2}$ and $z_i = B + s(a_i)$ for every i, it follows that $Z_q > 5B$ and $Z_p < 3B$. Consider an item $d_h \in G_q$ and move it to G_p. The resulting change in cost is

$$(Z_p + z_h)^2 + (Z_q - z_h)^2 - Z_p^2 - Z_q^2 = 2(Z_p - Z_q)z_h + 2z_h^2 < 0.$$

Therefore, a solution with a cost smaller than $4B$ has been found, which is a contradiction.

Let S_j denote $\sum_{d_i \in G_j} s(a_i)$. It remains to be proven that $S_j = B$, for $1 \leq j \leq m$. The overall cost can be written as

$$\frac{1}{4mB}\sum_{j=1}^{m}\left(\sum_{d_i \in G_j} z_i\right)^2 = \frac{1}{4mB}\sum_{j=1}^{m}\left(3B + \sum_{d_i \in G_j} s(a_i)\right)^2 \leq 4B$$

because there are exactly three items in each group. Since the above inequality implies that $\sum_{j=1}^{m} S_j^2 \leq mB^2$ and since, by hypothesis, $\sum_{i=1}^{3m} s(a_i) = \sum_{j=1}^{m} S_j = mB$, it follows that:

$$\left(\sum_{j=1}^{m} S_j\right)^2 = m(mB^2) \geq m\sum_{j=1}^{m} S_j^2. \tag{20}$$

On the other hand, again using the Cauchy-Schwartz inequality, one gets

$$\left(\sum_{j=1}^{m} S_j\right)^2 \leq m\sum_{j=1}^{m} S_j^2. \tag{21}$$

Combining (20) and (21), the Cauchy-Schwartz inequality becomes

$$\left(\sum_{i=1}^{m} S_i\right)^2 = (\mathbf{1} \cdot \mathbf{S})^2 = \|\mathbf{1}\|^2 \cdot \|\mathbf{S}\|^2 = m\sum_{i=1}^{m} {S_i}^2,$$

where $\mathbf{S} = (S_1, \ldots, S_m)$ and $\mathbf{1} = (1, \ldots, 1)$ are vectors in $\mathbb{Z}_m$.

Thus, $\mathbf{1} \cdot \mathbf{S} = \|\mathbf{1}\| \cdot \|\mathbf{S}\|$, that is, the vectors $\mathbf{1}$ and $\mathbf{S}$ are collinear. Hence, $S_k = S_j$ for all $1 \leq k, j \leq m$. Since

$\sum_{j=1}^{m} S_j = mB$, then $\sum_{d_i \in G_j} s(a_i) = S_j = B$ for $j = 1, \ldots, m$. Therefore, the resulting instance of 3-PARTITION is a "yes" instance. □

A.2 Proof of Theorem 6

Consider the decision problem K-NON-UNIFORM ALLOCATION, stated in the proof of Theorem 4 and let $K = 2$. To show that 2-NON-UNIFORM ALLOCATION is NP-hard, a polynomial time reduction from PARTITION [6] is provided:

PARTITION
INSTANCE: A finite set A and a size $s(a) \in \mathbb{Z}^+$ for each $a \in A$.
QUESTION: Is there a subset $A' \subseteq A$ such that $\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a)$?

Let $A = \{a_1, a_2, \ldots, a_n\}$ and $s(a_1), s(a_2), \ldots, s(a_n)$ constitute an arbitrary instance of PARTITION. The corresponding instance of 2-NON-UNIFORM ALLOCATION is given by $D = A$, $C = \frac{S}{2}$, where $S = \sum_{a \in A} s(a)$, $z_i = s(a_i)$, and $p_i = \frac{z_i}{S}$, for $i = 1, \ldots, n$.

Consider a "yes" instance of PARTITION, i.e., an instance for which there exists an $A' \subseteq A$ such that the sums of the sizes of the elements in A' and $A - A'$ are equal. Consider the solution of 2-NON-UNIFORM ALLOCATION, where $G_1 = A'$ and $G_2 = A - A'$ are the two groups. Since the sum of the lengths in each group is $\frac{S}{2}$, the total cost is $\frac{1}{2}(\frac{S}{2} + \frac{S}{2}) = C$. Hence, a "yes" instance of 2-NON-UNIFORM ALLOCATION results.

Conversely, consider a "yes" instance of 2-NON-UNIFORM ALLOCATION whose cost is at most C. Let Z_1 and Z_2 be the sum of the item lengths in the two groups G_1 and G_2.

Observing that $S = Z_1 + Z_2$ and applying the Cauchy-Schwartz inequality, one gets:

$$\frac{S^2}{2} = \frac{(Z_1 + Z_2)^2}{2} \leq Z_1^2 + Z_2^2. \tag{22}$$

On the other hand, the cost of the solution is:

$$\frac{1}{S}(Z_1^2 + Z_2^2) \leq C = \frac{S}{2}. \tag{23}$$

Combining (22) and (23) yields:

$$\frac{(Z_1 + Z_2)^2}{2} = Z_1^2 + Z_2^2,$$

which implies that $Z_1 = Z_2$. Therefore, $A' = G_1$ and $A - A' = G_2$ is a solution of PARTITION. □

ACKNOWLEDGMENTS

The authors wish to thank W.G. Yee for having provided the code of the Greedy heuristic and an anonymous referee for his helpful comments. A preliminary version of this paper appeared in the *Proceedings of the 18th IEEE International Parallel and Distributed Processing Symposium* (IPDPS04), Santa Fe, New Mexico, April 2004. This work has been supported by ISTI-CNR under the BREW research grant.

REFERENCES

[1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communication Environments," *Proc. SIGMOD,* May 1995.
[2] M.H. Ammar and J.W. Wong, "The Design of Teletext Broadcast Cycles," *Performance Evaluation,* vol. 5, no. 4, pp. 235-242, 1985.
[3] M.H. Ammar and J.W. Wong, "On the Optimality of Cyclic Transmission in Teletext Systems," *IEEE Trans. Comm.,* vol. 35, no. 11, pp. 1159-1170, 1987.
[4] A. Bar-Noy, R. Bhatia, J.S. Naor, and B. Schieber, "Minimizing Service and Operation Costs of Periodic Scheduling," *Proc. Ninth ACM-SIAM Symp. Discrete Algorithms (SODA),* pp. 11-20, 1998.
[5] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-Like Distributions: Evidence and Implications," *Proc. IEEE INFOCOM,* 1999.
[6] M.R. Garey and D.S. Johnson, *Computers and Intractability.* San Francisco: Freeman, 1979.
[7] T. Imielinski, S. Viswanathan, and B.R. Badrinath, "Energy Efficient Indexing on Air," *Proc. SIGMOD,* May 1994.
[8] C. Kenyon and N. Schabanel, "The Data Broadcast Problem with Non-Uniform Transmission Time," *Proc. 10th ACM-SIAM Symp. Discrete Algorithms (SODA),* pp. 547-556, 1999.
[9] C. Kenyon, N. Schabanel, and N. Young, "Polynomial Time Approximation Scheme for Data Broadcast," *Proc. ACM Symp. Theory of Computing (STOC),* pp. 659-666, 2000.
[10] S.-C. Lo and A.L.P. Chen, "Optimal Index and Data Allocation in Multiple Broadcast Channels," *Proc. 16th IEEE Int'l Conf. Data Eng. (ICDE),* Feb. 2000.
[11] S. Martello and P. Toth, *Knapsack Problems.* Chichester: Wiley, 1990.
[12] W.C. Peng and M.S. Chen, "Efficient Channel Allocation Tree Generation for Data Broadcasting in a Mobile Computing Environment," *Wireless Networks,* vol. 9, no. 2, pp. 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," *Proc. 16th IEEE Int'l Conf. Data Eng. (ICDE),* Feb. 2000.
[14] *Handbook of Wireless Networks and Mobile Computing,* I. Stojmenovic, ed. Chichester: Wiley, 2002.
[15] N. Vaidya and S. Hameed, "Log Time Algorithms for Scheduling Single and Multiple Channel Data Broadcast," *Proc. Third ACM-IEEE Conf. Mobile Computing and Networking (MOBICOM),* Sept. 1997.
[16] J. von zur Gathen and J. Gerhard, *Modern Computer Algebra.* Cambridge Univ. Press, 2003.
[17] W.G. Yee, "Efficient Data Allocation for Broadcast Disk Arrays," Technical Report GIT-CC-02-20, Georgia Inst. of Technology, 2001.
[18] W.G. Yee, S. Navathe, E. Omiecinski, and C. Jermaine, "Efficient Data Allocation over Multiple Channels at Broadcast Servers," *IEEE Trans. Computers,* vol. 51, no. 10, pp. 1231-1236, Oct. 2002.

Elia Ardizzoni received the bachelor's degree in computer science from the University of Bologna, Italy, in 2002. At present, he is working toward his master's degree in computer science at the University of Bologna. He is working at Too Much Software (Cento, Italy) as a system programmer and designer.

Alan A. Bertossi received the Laurea degree summa cum laude in computer science from the University of Pisa, Italy, in 1979. Afterward, he worked as a system programmer and designer. From 1983 to 1994, he was with the University of Pisa, first as a research associate and, later, as an associate professor. From 1995 to 2002, he was with the University of Trento, Italy, as a full professor. Since 2002, he has been with the Department of Computer Science of the University of Bologna, Italy, as a professor of computer science. His main research interests are the computational aspects of high-performance, parallel, VLSI, distributed, fault-tolerant, and real-time systems. He has published about 40 refereed papers in international journals, as well as several papers in international conferences, workshops, and encyclopedias. He has authored a book (on design and analysis of algorithms, in Italian) and he served as a guest coeditor for special issues of *Algorithmica*, *Discrete Applied Mathematics*, and *Mobile Networks and Applications*. He is a member of the editorial board of *Information Processing Letters*. His biography is included in the 1999 edition of *Who's Who in the World* and in the 2000 edition of *Who's Who in Science and Engineering*. Since 1999, he has been a scientific collaborator at the Institute of Information Sciences and Technologies of the Italian National Research Council (ISTI-CNR, Pisa, Italy). During 2001-2003, he was the national coordinator of an Italian research project on algorithms for wireless networks.

M. Cristina Pinotti received the Dr. degree cum laude in computer science from the University of Pisa, Italy, in 1986. During 1987-1999, she was a researcher with the National Council of Research at the Istituto di Elaborazione dell'Informazione, Pisa. From 2000-2003, she was an associate professor at the University of Trento. Currently, she is a professor at the University of Perugia. In 1994 and 1995, she was a research associate in the Department of Computer Sciences, University of North Texas, Denton. In 1997, she visited the Department of Computer Science, Old Dominion University, Norfolk, Virginia. Her research interests include computer arithmetic, residue number systems, VLSI special purpose architectures, design and analysis of parallel algorithms, parallel data structures, distributed data structures, multiprocessor interconnection networks, and wireless networks. She is a member of the IEEE Computer Society.

Shashank Ramaprasad graduated from Birla Institute of Technology and Science, Pilani, India, with a degree in computer science. He is currently pursuing the PhD degree at Brown University, Providence, Rhode Island. His research interests include combinatorial optimization, program debugging, and verification.

Romeo Rizzi received the Laurea degree in electronic engineering from the Politecnico di Milano in 1991 and received the PhD in computational mathematics and informatics from the University of Padova, Italy, in 1997. Afterward, he held postdoctoral and other temporary positions at research centers like CWI (Amsterdam, Holland), BRICS (Aarhus, Denmark), and IRST (Trento, Italy). In March 2001, he became an assistant professor on the Faculty of Science at the University of Trento. He is fond of combinatorial optimization and algorithms and has a background in operations research.

Madhusudana V.S. Shashanka received the BE (Hons) in computer science from the Birla Institute of Technology and Science, Pilani, India, in June 2003. He is currently a graduate student in the Department of Cognitive and Neural Systems at Boston University.