Optimal multi-channel data allocation with flat broadcast per channel

A.A. Bertossi* M.C. Pinotti † S. Ramaprasad ‡ R. Rizzi §

M.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. 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 $O(NK \log N)$

*Department of Computer Science, Mura Anteo Zamboni, 7, University of Bologna, 40127 Bologna, ITALY, bertossi@cs.unibo.it

†Department of Computer Science and Telecommunications, University of Trento, 38050 Povo, Trento, ITALY, pinotti@science.unitn.it

‡Department of Computer Science, Brown University, Providence, RI 02912, USA, shashank@cs.brown.edu

§Department of Computer Science and Telecommunications, University of Trento, 38050 Povo, Trento, ITALY, rrizzi@science.unitn.it

¶Department of Cognitive and Neural Systems, Boston University, Boston, MA 02215, USA, mvss@cns.bu.edu

time algorithm is proposed, which improves over the previously known $O(N^2K)$ time algorithm. When $K \leq 4$, faster $O(N)$ time algorithms are exhibited. Moreover, for data items with non-uniform 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 pseudo-polynomial algorithm is discussed, whose time is $O(NZ)$ where Z is the sum of the data lengths. In the latter case, two algorithms are devised with time exponential in the maximum data length.

Keywords: 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. A server of a base-station continuously 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 listen to many unwanted data before receiving his own data. The efficiency can be improved 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, the clients can access either a single channel at a time or all available channels simultaneously. 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. Index information for data allocation or for broadcast schedule can help the client to fast locate the desired item on the proper channel.

Several solutions for data allocation and broadcast scheduling have been proposed in the literature. The proposed solutions depend on the perspectives faced by the research communities.

Specifically, the networking community faces a version of the problem, known as the *Broadcast Problem*, which consists in finding an infinite schedule on a single channel [11, 3, 6, 7]. Such a problem was first introduced in the teletext systems by [2]. 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 [3, 6]), 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)*. Such a rule produces 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 [2]. The multi-channel schedule is obtained by distributing in a round robin fashion the schedule for a single channel [11]. Table 1 summarizes the results known in the literature for the Broadcast Problem depending on the number of channels and on the item lengths. For *uniform* lengths, namely all items of the same length, the problem complexity is open to our knowledge, while for *non-uniform* lengths the problem has been shown to be strong NP-hard.

On the other hand, the database community seeks for a periodic broadcast scheduling which should be easily indexed [5]. However, the solutions of the networking community preclude indexing. For the single channel, the obvious schedule that admits index is the *flat* one which, fixed an order among the data items, transmits them once at a time, in a round-robin fashion [1]. In a flat schedule, however, the client expected delay is half of the schedule period and becomes infeasible for a large period. To decrease the client expected delay, still preserving indexing, flat schedules on multiple channels can be adopted [9, 10, 13]. 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 [9, 13]. 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 [13], the optimal solution for uniform lengths can be found, by dynamic programming, in time polynomial in the number of items and channels. For non-uniform lengths, the problem tractability was unknown, but a heuristic has also been proposed in [13].

In this paper, the problem of data broadcasting over multiple channels, with the objective of minimizing the average waiting time of the clients, is considered under the same assumptions as in [13], that is skewed allocation to multiple channels and flat scheduling per channel.

Both the uniform and non-uniform length problems are faced and solved to the optimum, establishing also their tractability. All the proposed algorithms are based on dynamic programming, and provide optimal solutions for N data items and K channels as summarized in Table 2. Specifically, for uniform lengths, an $O(NK \log N)$ time algorithm is proposed, which improves over the previously known $O(N^2K)$ time algorithm by [13]. When $K \leq 4$, faster $O(N)$ time algorithms are exhibited. Moreover, for non-uniform lengths, it is shown that the problem is NP-hard when $K = 2$, and *strong* NP-hard for arbitrary K. When $K = 2$, a pseudo-polynomial 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. Such an algorithm 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 $O(N^2)$. The above algorithm is as effective as the standard pseudo-polynomial time algorithm for Knapsack, commonly judged to be extremely effective in practice [8], and allows Fully Polynomial Time Approximation Schemes (FPTAS) to be obtained as it is for the Knapsack problem. For arbitrary K, two algorithms are devised with time exponential in the maximum data length z. The two algorithms require $O(K(\Pi_{i=1}^{z}(L_i + 1))^2)$ and $O(z^2 \log K(\Pi_{i=1}^{z}(L_i + 1))^2)$ time, respectively, where L_i is the number of items of length i. When $z = 1$, the algorithms reduce to those presented for the uniform case. They are practical as far as z is a small constant.

<table>
<tr><th>♯ channels</th><th>item lengths</th><th>complexity</th><th>solution</th><th>references</th></tr>
<tr><td rowspan="2">1</td><td rowspan="2">uniform</td><td rowspan="2">?</td><td>$\frac{9}{8}$-approximation</td><td>[3]</td></tr>
<tr><td>heuristic</td><td>[11]</td></tr>
<tr><td>$O(1)$</td><td>uniform</td><td>?</td><td>$PTAS$</td><td>[7]</td></tr>
<tr><td>1</td><td>non-uniform</td><td>strong NP-hard</td><td>3-approximation</td><td>[6]</td></tr>
<tr><td>K</td><td>non-uniform</td><td>strong NP-hard</td><td>heuristic</td><td>[6, 11]</td></tr>
</table>

Table 1: Known results for the Broadcast Problem. $PTAS$ stands for *Polynomial Time Approximation Scheme*.

<table>
<tr><th>♯ channels</th><th>item lengths</th><th>complexity</th><th>solution</th><th>running time</th><th>references</th></tr>
<tr><td>1</td><td>non-uniform</td><td>P</td><td>optimal</td><td>$O(1)$</td><td>folklore</td></tr>
<tr><td>≤ 4</td><td>uniform</td><td>P</td><td>optimal</td><td>$O(N)$</td><td>this paper</td></tr>
<tr><td rowspan="2">K</td><td rowspan="2">uniform</td><td rowspan="2">P</td><td rowspan="2">optimal</td><td>$O(N^2K)$</td><td>[13]</td></tr>
<tr><td>$O(NK \log N)$</td><td>this paper</td></tr>
<tr><td>2</td><td>non-uniform</td><td>NP-hard</td><td>optimal</td><td>$O(NZ)$</td><td>this paper</td></tr>
<tr><td rowspan="3">K</td><td rowspan="3">non-uniform</td><td rowspan="3">strong
NP-hard</td><td rowspan="2">optimal</td><td>$O(K(\Pi_{i=1}^{z}(L_i+1))^2)$</td><td rowspan="2">this paper</td></tr>
<tr><td>$O(z^2 \log K(\Pi_{i=1}^{z}(L_i+1)^2)$</td></tr>
<tr><td>heuristic</td><td>$O((N+K)\log K)$</td><td>[13]</td></tr>
</table>

Table 2: Known results for the broadcast problem with skewed allocation and flat scheduling. In the table, Z is the sum of the data lengths, z is the maximum data length, and L_i is the number of data items of length i. When $K > 1$, all the algorithms assume a sorting preprocessing step on the data items, which requires $O(N \log N)$ time.

The rest of this paper is so organized. Section 2 gives notations, definitions and the problem statement. Section 3 efficiently solves the problem assuming uniform lengths and an arbitrary number of channels. In particular, Subsection 3.1 presents more efficient algorithms when there are at most four channels. Section 4 studies the non-uniform length case, with an arbitrary number of channels. Then, Subsection 4.1 discusses the non-uniform problem with only two channels. Conclusions are offered in Section 5, while the NP-hardness proofs for the non-uniform case are exhibited in the Appendix.

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 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 (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 *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, 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, the *client expected delay* for receiving item d_i is half of the period, namely $\frac{Z_j}{2}$. Therefore, the *average expected delay* (AED) over all data items and over all channels is

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

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 in partitioning D into K groups $G_1, \ldots, G_K$, so as to minimize the objective function AED given in Equation 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.

The rest of this section is devoted to briefly recalling the dynamic programming solution proposed in [13] for the K-Uniform Allocation Problem.

Lemma 1. [13] *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 $d_j \in G_k$. 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. *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 their 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.

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 putting consecutive items $d_i, \ldots, d_h$ into one group, i.e. $C_{i,h} = (h-i+1)\sum_{q=i}^{h} p_q$. 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 algorithm proposed in [13] is a straightforward dynamic programming implementation of Recurrence 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 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 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 above 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 algorithm is $O(N^2K)$.

3 Uniform Lengths

A first improvement on the algorithm proposed in [13] for the K-Uniform Allocation Problem can be achieved observing that the group cardinalities are bounded.

Lemma 2. *Let the items $d_1, d_2, \ldots, d_N$ be sorted in increasing order of probabilities. There exists an optimal solution where these items are partitioned into K groups $G_1, G_2, \ldots, G_K$, such that $N_1 \geq \lfloor \frac{N}{K} \rfloor$ and $N_K \leq \lceil \frac{N}{K} \rceil$.*

Proof. From Lemma 1 and Corollary 1, there exists an optimal solution such that $N_1 \geq N_2 \geq \ldots \geq N_K$. Hence, the result follows. □

The above result can be used to modify Recurrence 2 achieving a better $O(N^2 \log K)$ time complexity. Let the N items be sorted by increasing probabilities. The entries of the matrices M and F are again filled row by row. However, unlike the previous case, filling each entry requires fewer comparisons. Precisely, when $M_{k,n}$ is being filled, there are n items and k groups and the last group contains *at most* $\lceil \frac{n}{k} \rceil$ items, by Lemma 2. This implies that there is a solution where the value of the *final border* of $OPT_{n,k}$ is greater than or equal to $n - \lceil \frac{n}{k} \rceil$. In other words, for the variable ℓ of Recurrence 2, only the last $\lceil \frac{n}{k} \rceil$ values of the indices $\{1, 2, \ldots, n-1\}$ have to be considered. Then, Recurrence 2 simplifies as:

$$opt_{n,k} = \min_{\ell \in \{n - \lceil \frac{n}{k} \rceil, \ldots, n-1\}} \{opt_{\ell,k-1} + C_{\ell+1,n}\} \tag{3}$$

To evaluate the time complexity, observe that $\lceil \frac{n}{k} \rceil$ comparisons are required to calculate $opt_{n,k}$. Hence, to fill all entries of row k in matrix M, $\sum_{n=1}^{N} \lceil \frac{n}{k} \rceil = O(\frac{N^2}{k})$ comparisons are needed. Thus, the overall time complexity is $O(\sum_{k=1}^{K} \frac{N^2}{k}) = O(N^2 \log K)$.

A better improvement on the time complexity can be achieved further exploiting the properties of optimal solutions.

Definition 1. *Let $d_1, d_2, \ldots, d_N$ be items sorted by decreasing 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 $B_{K-1} \leq B'_{K-1}$.*

The left-most optimal solutions do not need to be unique. However, 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 $i < K$.*

Lemma 3. *Let the items $d_1, d_2, \ldots, d_N$ be sorted by decreasing 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 putting d_N into the K-th channel. Then:

$$opt_{B'_{K-1},K-1} + C_{B'_{K-1}+1,N} = opt_{N,K} \leq opt_{B_{K-1},K-1} + C_{B_{K-1}+1,N}. \tag{4}$$

Assuming by contradiction $B'_{K-1} < B_{K-1}$ implies that:

$$C_{B'_{K-1}+1,N} - C_{B'_{K-1}+1,N-1} \geq C_{B_{K-1}+1,N} - C_{B_{K-1}+1,N-1} \tag{5}$$

Subtracting Equation 5 from Equation 4 yields:

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

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

In practice, Lemma 3 says that, given the items sorted by decreasing probabilities, building an optimal solution for N items from an optimal solution for $N-1$, the final border B_{K-1} can only move on the right. Such a property can be easily generalized as follows to problems of increasing sizes. From now on, let B_j^i denote the j-th 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 decreasing probabilities, and let $a < b < c \leq N$. Then, $B_{K-1}^a \leq B_{K-1}^b \leq B_{K-1}^c$.*

Proof. Follows directly from Lemma 3. $\square$

Lemma 3 plays a fundamental role in speeding up the dynamic programming algorithm. Indeed, it leads to an $O(KN \log N)$ time algorithm, as detailed below. The new algorithm solves subproblems in a dichotomic fashion. Formally, 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 the algorithm can compute $LMO_{\frac{l+r}{2},k}$ by the following recurrence:

$$opt_{\frac{l+r}{2},k} = \min_{\ell \in \{B_{k-1}^l, \ldots, B_{k-1}^r\}} \{opt_{\ell,k-1} + C_{\ell+1,\frac{l+r}{2}}\} \tag{6}$$

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

In details, the algorithm is shown in Figure 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 $F_{k,N}$ is initialized to N. To compute the entry j, the variable ℓ takes all values between $F_{k,l}$

Input:	N items sorted by decreasing probabilities, and K groups;
	`begin`
Initialize:	`for` i `from` 1 `to` N `do`
	$M_{1,i} \leftarrow C_{1,i}$;
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`
	$j \leftarrow \lceil \frac{2i-1}{2^t} N \rceil$; $l \leftarrow \lceil \frac{i-1}{2^{t-1}} N \rceil$; $r \leftarrow \lceil \frac{i}{2^{t-1}} N \rceil$; $M_{k,j} \leftarrow \infty$;
	`if` $i = 2^{t-1}$ `then` $r \leftarrow r + 1$;
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`
	$M_{k,j} \leftarrow M_{k-1,\ell} + C_{\ell+1,j}$;
	$F_{k,j} \leftarrow \ell$;
	`end`

Figure 1: The $O(KN \log N)$ time algorithm for the K-Uniform Allocation Problem.

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 $M_{k,\frac{N}{2}}$ is filled and ℓ ranges over all values $1, \ldots, N$. By Corollary 2, observe that to fill an entry $M_{k,l}$ where $l < \frac{N}{2}$, one needs to consider only the entries $M_{k-1,\ell}$ where $\ell \leq F_{k,\frac{N}{2}}$. Similarly, to fill an entry $M_{k,l}$ where $l > \frac{N}{2}$, one needs to consider only the entries $M_{k-1,\ell}$ where $\ell \geq F_{k,\frac{N}{2}}$. In general, one can show that in stage t, to compute the entries $M_{k,j}$ with $j = \lceil \frac{2i-1}{2^t} N \rceil$ 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 $\lceil \frac{i-1}{2^{t-1}} N \rceil$ and $\lceil \frac{i}{2^{t-1}} N \rceil$, 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 $B_K = N$.

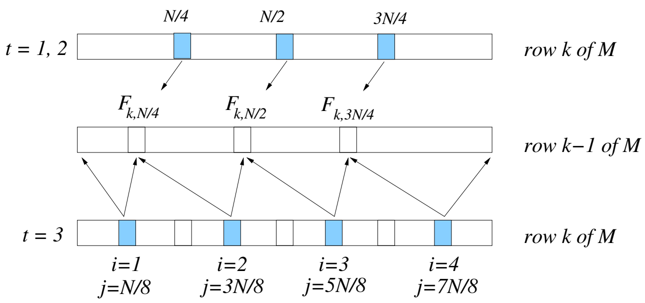

Figure 2: Illustration of Loop 2.

As an example, consider Figure 2 which illustrates the execution of Loop 2 with $t = 3$, where the entries corresponding to $i = 1, 2, 3, 4$ of row k of matrix M are being computed. The $\frac{N}{4}$-th, $\frac{N}{2}$-th, and $\frac{3N}{4}$-th entries have already been computed in stages 1 and 2. Let $F_{k,\frac{N}{4}}$, $F_{k,\frac{N}{2}}$ and $F_{k,\frac{3N}{4}}$ 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 $M_{k-1,F_{k,\frac{N}{4}}}$. Similarly, for $i = 2$, only the entries from $M_{k-1,F_{k,\frac{N}{4}}}$ to $M_{k-1,F_{k,\frac{N}{2}}}$ are to be examined. For $i = 3$, one examines the entries from $M_{k-1,F_{k,\frac{N}{2}}}$ up to $M_{k-1,F_{k,\frac{3N}{4}}}$, and, finally, for $i = 4$, the entries beyond $M_{k-1,F_{k,\frac{3N}{4}}}$ are visited.

Lemma 4. *The total number of comparisons involved in a stage is $O(N)$.*

Proof. The whole execution of Loop 3 of Figure 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:

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

Unrolling Formula 7, one obtains:

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

□

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

Proof. From Lemma 4, one stage of Figure 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 $O(NK \log N)$. □

3.1 At Most Four Channels

In this subsection, faster algorithms are proposed for the K-Uniform Allocation Problem, when the number of channels K is less than or equal to 4. All algorithms require $O(N)$ time to solve the problem and they are 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 $O(1)$ (resp., $O(\log N)$) time.

When $K = 2$, Lemma 3 can be further simplified. Indeed, in such a case, the final border can move at most one position to the right.

Lemma 5. *Let the items $d_1, d_2, \ldots, d_N$ be sorted by decreasing probabilities. Let $LMO_{N-1,2} = (B_1)$ and $LMO_{N,2} = (B'_1)$. Then, $B_1 \leq B'_1 \leq B_1 + 1$.*

Proof. Let $p_i^{(j)}$ be the probability of the i-th item $d_i^{(j)}$ in G_j, with $j = 1, 2$. Since the items are sorted by decreasing order, then $p_1^{(j)} \geq p_2^{(j)} \geq \ldots \geq p_{N_j}^{(j)}$. 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:

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

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

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

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

$$opt_{N,2} - cost_{N,2} = (P_1 - P_2)(t-1) + (N_1 - N_2 + 2t - 1) \sum_{i=2}^{t} p_i^{(2)} + (t-1)(2p_1^{(2)} - p_N) \leq 0 \tag{10}$$

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

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

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

Substituting Δ in Equation 10, one gets:

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

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:

$$opt_{n,2} = \min_{\ell \in \{B_1^{n-1}, B_1^{n-1}+1\}} \{C_{1,\ell} + C_{\ell+1,n}\} \tag{13}$$

Therefore, the following theorem holds:

Theorem 2. *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:

$$opt_{N,3} = \min_{\ell \in \{1,\ldots,N\}} \{opt_{\ell,2} + C_{\ell+1,N}\} \tag{14}$$

Corollary 3. *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 5 is needed when the new item to be added is that with the greatest probability.

In the rest of this subsection, the items are assumed to be indexed by decreasing probabilities and 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 6. *Let the items $d_1, d_2, \ldots, d_N$ be sorted by decreasing 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 3. □

For the aim of determining the exact index of the final border B'_1 of $LMO_{1,N,2}$, consider the feasible solutions obtained 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 so founded B'_1 is optimal.

Lemma 7. *Let the items $d_1, d_2, \ldots, d_N$ be sorted by decreasing 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 5, let $p_i^{(j)}$ be the probability of the i-th 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, $s_{n,N,2} = N_1 P_1 + N_2 P_2$ and $s'_{n,N,2} = (N_1 - 1)(P_1 - p_{N_1}^{(1)}) + (N_2 + 1)(P_2 + p_{N_1}^{(1)})$. Thus $s'_{n,N,2} > s_{n,N,2}$ if and only if

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

Note that Equation 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

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

To complete the proof, note that Equation 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:

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

Note that in Equation 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. *All the solutions $LMO_{n,N,2}$, with $1 \le n \le N$, 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 7 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 to an efficient algorithm for finding the optimal solution $LMO_{N,4}$ of the 4-Uniform Allocation Problem, by combining two solutions for $K = 2$:

$$opt_{1,N,4} = \min_{\ell \in \{1,\ldots,N\}} \{opt_{1,\ell,2} + opt_{\ell+1,N,2}\} \tag{18}$$

Corollary 4. *The optimal solution $LMO_{N,4}$ of the 4-Uniform Allocation Problem can be found in $O(N)$ time.*

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 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 pseudo-polynomial time optimal algorithm or fully polynomial time approximation scheme (FPTAS) for solving the K-Non-Uniform Allocation Problem. However, when the maximum item length z is bounded by a constant, polynomial time optimal algorithms can be derived where z appears in the exponent. These algorithms are similar to those presented in Section 3 for the K-Uniform Allocation Problem.

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

Lemma 8. *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:

$$\Delta = Z_h(p_k - p_i) + Z_j(p_i - p_k) = (Z_h - Z_j)(p_k - p_i) < 0$$

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 8. □

In the following, the items in each D_i are assumed to be sorted by decreasing 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^i_j 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 $(B^1_j, B^2_j, \ldots, B^z_j)$ for $1 \leq j \leq K-1$. From now on, B^i_{K-1} 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 $d^i_1, d^i_2, \ldots, d^i_{n_i}$, $1 \leq i \leq z$, 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.

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

Now, consider the recurrence:

$$opt_{n_1,\ldots,n_z,k} = \min_{\substack{\ell_i \in \{0,1\ldots,n_i\} \\ 1\leq i\leq z}} \left\{ opt_{\ell_1,\ldots,\ell_z,k-1} + C_{\ell_1+1,n_1,\ldots,\ell_z+1,n_z} \right\} \tag{19}$$

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^i_1 through $d^i_{n_i}$, 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 first straightforward algorithm derives directly from Recurrence 19.

Theorem 5. *The K-Non-Uniform Allocation Problem can be solved in $O(K\prod_{i=1}^{z}(L_i+1)^2)$ time.*

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

A second algorithm, which generalizes the algorithm derived from Recurrence 3, can be designed as follows. As before, let z denote the maximum item length. In addition, let $Z = \sum_{i=1}^{N} z_i$ denote the sum of all item lengths, and consider a given partition $G_1, \ldots, G_K$.

Lemma 9. *Let $d_1^i, d_2^i, \ldots, d_{L_i}^i$ be the L_i items of length i with $p_m^i \leq p_n^i$ whenever $m < n$, for $i = 1, \ldots, z$. There exists an optimal solution where $Z_K \leq \lceil \frac{Z}{K} \rceil$.*

Proof. From Corollary 5, there is an optimal solution where $d_a^i, d_c^i \in G_j$ implies $d_b^i \in G_j$ whenever $a > b > c$, for $1 \leq i \leq z$ and $1 \leq j \leq K$. In other words, items of the same length allocated to the same group appear in consecutive positions. From Lemma 8, for any two items $d_m^i \in G_h$, $d_n^i \in G_j$, $p_m^i < p_n^i$ implies $Z_h \geq Z_j$. This means that, for any two consecutive groups G_j and G_{j+1}, $Z_j \geq Z_{j+1}$. Hence, the result holds. $\square$

By the above lemma, an algorithm is derived where, as in the previous algorithm, the entries of the $(z+1)$-dimensional matrices M and F are filled row by row. However, to compute the value of the entry $M_{k,n_1,\ldots,n_z}$, less than $\prod_{i=1}^{z}(L_i + 1)$ comparisons are required. Indeed, from Lemma 9, it is known that $Z_k \leq (\sum_{i=1}^{z} in_i)/k$. Therefore, in Recurrence 19, only those optimal solutions $OPT_{\ell_1,\ldots,\ell_z,k-1}$ have to be considered for which the sum of the lengths of the items in channel k satisfies Lemma 9, that is,

$$\sum_{i=1}^{z} \sum_{h=\ell_i+1}^{n_i} i = \sum_{i=1}^{z} i(n_i - \ell_i) \leq \Big(\sum_{i=1}^{z} in_i \Big)/k \tag{20}$$

In other words, one only needs to consider the values $(\ell_1, \ldots, \ell_z) \in \mathbb{N}^z$, with $0 \leq \ell_i \leq n_i$, satisfying Inequality 20, which can be written as:

$$\sum_{i=1}^{z} i\ell_i \quad \geq \quad \frac{k-1}{k} \sum_{i=1}^{z} in_i$$

To compute the time complexity, the number of z-tuples $(\ell_1, \ldots, \ell_z)$, which satisfy Inequality 20, has to be counted.

Lemma 10. *Let $\lambda = (\sum_{i=1}^{z} in_i)/k$. The number of integral points $(x_1, \ldots, x_z)$ which satisfy the inequality $\sum_{i=1}^{z} ix_i \leq \lambda$ and such that $0 \leq x_i \leq n_i$, is $O\Big(\frac{\lambda}{mn_m} \prod_{i=1}^{z}(n_i + 1)\Big)$, where $n_m = \max\{n_1, \ldots, n_z\}$.*

Proof. Let the number of integral points satisfying the above condition by denoted by $\#(x_1, \ldots, x_z)$. Rearranging the inequality $\sum_{i=1}^{z} ix_i \leq \lambda$, one has

$$x_m \leq \frac{1}{m}(\lambda - \sum_{\substack{i=1 \\ i \neq m}}^{z} ix_i)$$

which implies that

$$\#(x_1, \ldots, x_z) \leq \frac{\lambda}{m} \prod_{\substack{i=1 \\ i \neq m}}^{z} (n_i + 1) = O\Big(\frac{\lambda}{mn_m} \prod_{i=1}^{z} (n_i + 1)\Big)$$

□

Theorem 6. *The K-Non-Uniform Allocation Problem can be solved in $O\Big(z^2 \log K \big(\prod_{i=1}^{z}(L_i + 1)\big)^2\Big)$ time.*

Proof. The number of integral points which satisfy Inequality 20 is maximal when $(x_1, \ldots, x_z) = (L_1, \ldots, L_z)$. By Lemma 10, the number of operations needed to fill a single entry in row k is $O\Big(\frac{z}{mL_m}\big(\sum_{i=1}^{z} L_i/k\big) \prod_{i=1}^{z}(L_i + 1)\Big)$. To fill all the $\prod_{i=1}^{z} L_i$ entries of row k, $O\Big(\frac{z}{mL_m}\big(\sum_{i=1}^{z} L_i/k\big)\big(\prod_{i=1}^{z}(L_i + 1)\big)^2\Big)$ operations are needed. Summing up over all the K rows, one gets

$$O\Big(\sum_{k=1}^{K} \Big(\frac{z}{mL_m}\big(\sum_{i=1}^{z} L_i/k\big)\big(\prod_{i=1}^{z}(L_i + 1)\big)^2\Big)\Big)$$

Finally, observing that $\sum_{k=1}^{K} \frac{1}{k} = O(\log K)$, $\sum_{i=1}^{z} L_i \leq zL_m$, and $m \geq 1$, the overall time complexity is:

$$O\Big(z^2\big(\prod_{i=1}^{z}(L_i + 1)\big)^2 \log K\Big)$$

□

By Theorems 5 and 6, the K-Non-Uniform Allocation Problem can be solved by two alternative algorithms. When $\frac{K}{\log K} < z^2$, the former algorithm has the lowest time complexity, otherwise the latter algorithm is faster. However, both algorithms require a time which is exponential in the maximum item length z. Therefore, they are practical only when z is a small costant (for instance $z = 2$). Observe that there is no hope to devise an algorithm whose time complexity is not exponential since the problem is strong NP-hard.

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 7. *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 *pseudo-polynomial* 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 $Z_1P_1 + Z_2P_2$ 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 $P = P_1 + P_2$ and $Z = Z_1 + Z_2$, and assume, without loss of generality, that $Z_1 \leq Z_2$. Observe that there are only $\lfloor Z/2 \rfloor$ possible values for Z_1. If one solves the 2-Non-Uniform Allocation Problem for a fixed value of Z_1, then $\min\{Z_1P_1 + Z_2P_2\} = \min\{P_1(Z_1 - Z_2)\} = \max\{P_1\}$. 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, then the 2-Non-Uniform Allocation Problem can be reduced to a particular *Knapsack problem* [8], which can be solved in pseudo-polynomial time using dynamic programming.

Consider the 2-Non-Uniform Allocation Problem with N items $d_1, d_2, \ldots, d_N$, where each d_i is characterized by its demand probability p_i and its length z_i, and let Z_1 be fixed to B. Then, define a Knapsack problem of capacity B on the same N items $d_1, d_2, \ldots, d_N$, where each item d_i is characterized by a *profit* $p_i + Pz_i$ and a *weight* z_i. The problem consists in finding a subset S of $\{d_1, d_2, \ldots, d_N\}$ subject to the constraint $\sum_{d_k \in S} z_k \leq B$ so as to maximize the objective function $\sum_{d_k \in S}(p_k + Pz_k)$.

In the following it is shown that, by construction, the constraint $\sum_{d_k \in S} z_k$ reaches B whenever possible, and that in such a case the Knapsack solution S coincides with the optimal solution G_1 for the 2-Non-Uniform Allocation Problem with $Z_1 = B$. Indeed, while maximizing the total profit $\sum_{d_k \in S}(p_k + Pz_k)$, the quantity $\sum_{d_k \in S} z_k$ is maximized earlier than $\sum_{d_k \in S} p_k$, since each increment of z_k contributes by $P = \sum_{i=1}^{N} p_i$. Hence, if the capacity B is reachable, then the optimal solution S will have $\sum_{d_k \in S} z_k = B$. In such a case, the maximum $P_1 = \sum_{d_k \in S} p_k$ is found, and hence the optimal solutions S and G_1 coincide.

To apply dynamic programming, consider an $(N+1) \times \lfloor Z/2 \rfloor$ matrix M, where the entry $M_{i,j}$ stores the solution for the above Knapsack problem for $S_i = \{d_1, \ldots, d_i\}$ and capacity j, with $1 \leq j \leq \lfloor Z/2 \rfloor$ and $0 \leq i \leq N$. Formally, $M_{i,j} = \max \sum_{d_k \in S} (p_k + P z_k)$ such that $\sum_{d_k \in S} z_k \leq j$, where $S \subseteq S_i$.

Starting from $M_{0,j} = 0$, for $1 \leq j \leq \lfloor Z/2 \rfloor$, the matrix M is filled row by row using the following recurrence:

$$M_{i+1,j} = \begin{cases} \max\{M_{i,j},\ M_{i,j-z_{i+1}} + p_{i+1} + P z_{i+1}\} & z_{i+1} \leq j \\ M_{i,j} & z_{i+1} > j \end{cases}$$

Whenever the solution for the Knapsack problem completely fills the capacity, i.e., the sum of the item weights is exactly equal to j, the entry $M_{N,j}$ gives the optimal solution for the 2-Non-Uniform Allocation Problem with $Z_1 = j$. Note that it is possible that for certain values of j, with $1 \leq j \leq \lfloor Z/2 \rfloor$, there is no solution such that the total sum of weights is j. In such cases, the results are discarded since they are not significant.

As said earlier, to solve the 2-Non-Uniform Allocation Problem, all the values of Z_1 between 1 and $\lfloor Z/2 \rfloor$ have to be considered. The solution costs for such problems can be derived from the last row of M. For this purpose, the sum of the weights corresponding to $M_{i,j}$ is kept in the entry $F_{i,j}$ of an auxiliary matrix F. Then, consider those entries $M_{N,j}$ for which $F_{N,j} = j$, and compute $P_1 = M_{N,j} - jP$. The solution of the 2-Non-Uniform Allocation Problem is $\max_{1 \leq j \leq \lfloor Z/2 \rfloor} \{M_{N,j} - jP \ : \ F_{N,j} = j\}$, which can be found scanning the last row of M and F. Once the entry giving the optimal solution is found, it is easy to list out the items which have been picked up by tracing back the solution path.

Theorem 8. *The 2-Non-Uniform Allocation Problem can be solved in $O(NZ)$ time.*

Proof. The matrices M and F have $(N+1) \times \lfloor Z/2 \rfloor$ entries. Each entry can be computed in constant time. Moreover, the maximum on the last row of M costs $O(Z)$ time. Hence, the time complexity of the dynamic programming algorithm is $O(NZ)$. $\square$

The above algorithm 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 pseudo-polynomial time algorithm for the Knapsack problem, and allows Fully Polynomial Time Approximation Schemes (FPTAS) to be obtained by standard techniques.

5 Conclusions

In this paper, the problem of data broadcasting over multiple channels, with the objective of minimizing the average waiting time 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, proposing new algorithms based on dynamic programming. For uniform lengths, an $O(NK \log N)$ time algorithm has been proposed, which improves over the previously known $O(N^2K)$ time algorithm by [13]. When $K \leq 4$, faster $O(N)$ time algorithms were exhibited. Moreover, for non-uniform lengths, it has been shown that the problem is NP-hard when $K = 2$, and strong NP-hard for arbitrary K. When $K = 2$, a pseudo-polynomial time algorithm has been devised whose overall time is $O(NZ)$, where Z is the sum of the data lengths. For arbitrary K, two algorithms were designed whose time comlexity is exponential in the maximum data length z. When $z = 1$, such algorithms reduce to those presented for the uniform case.

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.

Appendix

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 $\sum_{j=1}^{K} \Big((\sum_{d_i \in G_j} z_i)(\sum_{d_i \in G_j} p_i) \Big) \leq C$?

In the following, it is proved that K-NON-UNIFORM ALLOCATION is strong NP-hard by exhibiting a polynomial time reduction from 3-PARTITION [4].

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 as follows:

$$D = A$$

$$K = m$$

$$C = 4B$$

$$z_i = B + s(a_i) \qquad i = 1, \ldots, 3m$$

$$p_i = \tfrac{z_i}{4mB} \qquad i = 1, \ldots, 3m$$

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

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

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 $\mathbb{Z}_m$ [12].

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

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

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

$$\left(\sum_{j=1}^{m} S_j \right)^2 \leq m \sum_{j=1}^{m} S_j^2 \tag{22}$$

Combining Equations (21) and (22), 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. $\square$

Proof of Theorem 7

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 [4] 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:

$$\begin{aligned}
D &= A \\
C &= \tfrac{S}{2} && \text{where } S = \sum\nolimits_{a \in A} s(a) \\
z_i &= s(a_i) && i = 1, \ldots, n \\
p_i &= \tfrac{z_i}{S} && i = 1, \ldots, n
\end{aligned}$$

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 $\frac{1}{2}\left(\frac{S}{2} + \frac{S}{2}\right) = C$. 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:

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

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

$$\frac{1}{S}(Z_1^2 + Z_2^2) \leq C = \frac{S}{2} \tag{24}$$

Combining Inequalities 23 and 24 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. $\square$

References

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

[2] 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.

[3] 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.

[4] M.R. Garey and D.S. Johnson. *Computers and Intractability.* W.H. Freeman and Company, San Francisco, 1979.

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

[6] 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.

[7] 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.

[8] S. Martello and P. Toth. *Knapsack Problems.* Wiley, Chichester, 1990.

[9] 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.

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

[11] 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.

[12] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra.* Cambridge University Press, 2003.

[13] 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.