

# Algorithms for Storage Allocation Based on Client Preferences

Tami Tamir \*

Benny Vaksendiser †

## Abstract

We consider a packing problem arising in storage management of *Video on Demand* (VoD) systems. The system consists of a set of video files (movies) and several servers (disks), each having a limited storage capacity,  $C$ , and a limited bandwidth (load capacity),  $L$ . The goal in the *storage allocation* problem is to assign the video files to the servers and the bandwidth to the clients. The induced *class-constrained packing* problem was studied in the past assuming each client provides a single request for a single movie. This paper considers a more general and realistic model – in which each client ranks all the movies in the system. Specifically, for each client  $j$  and movie  $i$ , it is known how much client  $j$  is willing to pay in order to watch movie  $i$ . The goal is to maximize the system's profit.

We prove that the problem is NP-complete and present approximation algorithms and heuristics for systems with a single or multiple disks. For a single disk we present an  $(1 - 1/e)$ -approximation algorithm that we extended for systems with storage costs, and for  $k$ -round broadcasting, in which each client might be serviced multiple times. For multiple disks we present a  $(C - 1)(e - 1)/Ce$ -approximation algorithm, two heuristics for determining the storage allocation, and an optimal bandwidth-allocation algorithm.

In our simulation of a VoD system, we compared the performance of the suggested heuristics for systems with variable parameters and clients with variable preference distributions. We focused on systems in which client preferences and payment are power-law distributed: a few movies are very popular and clients are willing to pay significantly more for watching them.

Our results can be applied to other packing and subset selection problems in which clients provide preferences over the elements.

---

\*School of Computer Science, The Interdisciplinary Center, Herzliya, Israel. tami@idc.ac.il

†School of Computer Science, The Interdisciplinary Center, Herzliya, Israel. vaksendise.benny@idc.ac.il

# 1 Introduction

This paper considers a variant of the knapsack problem arising in storage management of *Video on Demand* (VoD) systems. VoD services allow users to select and watch video content at their desired time. Formally, a VoD system services  $n$  clients, that are interested in watching movies from a collection of  $M$  movies  $f_1, f_2, \dots, f_M$ . The system has limited resources: it consists of  $N$  identical disks  $d_1, d_2, \dots, d_N$ , each having a limited storage capacity,  $C$ , and a limited bandwidth (load capacity),  $L$ . Each transmission requires a dedicated stream of one bandwidth (load) unit. This implies that each of the  $N$  disks can store movies of total size  $C$  and can transmit broadcasts to at most  $L$  clients simultaneously. The  $L$  transmissions are of movies stored on the disk, with no restrictions on their distribution (in particular, all  $L$  streams might broadcast the same movie).

The problem is therefore reduced to a *class-constrained packing* problem, in which the items to be packed (streams) are drawn from  $M$  classes (movies) and have the same unit size. The bins (disks) have a limited capacity,  $L$ , and can pack items from at most  $C$  classes. This storage management problem motivated the study of class-constrained packing in recent years (see e.g. [7, 12, 18]). In all previous work it is assumed that each client specifies a single movie he wishes to watch and the goal is to allocate storage to movies and transmissions to clients in a way that maximizes the number of clients whose request is granted. In this paper we define and study a more generalized setting: For each client  $j$  and movie  $i$ , let  $b_{i,j}$  denote the payment that client  $j$  is willing to pay for watching movie  $i$ . That is, each client provides his complete preference over the whole collection of movies. The previously studied system is in fact the special case in which for each client  $j$ ,  $b_{i,j} = 1$  for a single movie  $i$ , and  $b_{i',j} = 0$  for any  $i' \neq i$ <sup>1</sup>. The objective is to allocate storage to movies and transmissions to clients in a way that maximizes the system's profit given by  $P = \sum_{j=1}^n \{b_{i,j} \mid \text{movie } i \text{ is transmitted to client } j\}$ .

For this objective, we consider systems with a single disk or with multiple disks. We distinguish between the *single-round problem*, in which each client is watching at most one movie, and the *k-round problem*, in which there are  $k$  synchronized rounds and the goal is to maximize the total profit. For multiple disks, we assume that all movie files require the same storage. For single disk our results are suitable also when movie files have arbitrary sizes or are associated with arbitrary storage costs (that are reduced from the profit  $P$ ). Table 1 provides a glossary of the notations used in the paper.

For a single disk, we also consider a variant in which, instead of specifying its potential payments, each client provides an ordered tie-free list of preferences of the movies. That is, a list of length  $M$  in which the first element is the 1<sup>st</sup> choice of the client, etc. We consider two objectives for this variant: maximizing the *lexicographic profile* of an assignment, which quantify the total satisfaction of the clients, and maximizing the *fairness*, which measures the satisfaction of the least-satisfied client.

## 1.1 Related Work

VoD systems have been studied extensively in recent years. Many new algorithmic problems arose with the study of these systems. In particular, the storage allocation problem got much attention

---

<sup>1</sup>In practice, naturally, clients will not rank the whole selection of movies. Our model does not require a complete ranking, and the payment for any number of movies can be zero.

notation	meaning
$M$	Number of movies.
$n$	Number of clients.
$b_{i,j}$	The payment that client $j$ is willing to pay for watching movie $i$ .
$N$	Number of disks.
$L$	Load capacity of each disk.
$C$	Storage capacity of each disk.
$P$	System's profit.
$k$	Number of rounds in the multiple-round problem.

Table 1: Glossary of notations.

(e.g. [7, 12, 23, 24, 26, 16]). In particular, the induced class-constrained packing problem was studied in [7, 18, 19] and the generalization to packing with a sharable dimension in [12, 20]. In all the above work it is assumed that each client specifies a single request for a single transmission (no payment vectors), and the goal is to maximize the number of serviced clients. In [18] the problem is defined and shown to be NP-hard. The *moving window* (MW) algorithm is presented and shown to be optimal for certain systems. The paper [7] gives a general analysis of the MW algorithm and presents the first PTAS for the storage allocation problem. More general PTASs were developed for systems with variable disks [19], or variable movie files [12].

Other works deal with *dynamic* storage allocation, where the load on the disks is balanced using file deletion and replication. The papers [4, 23, 26] suggest dynamic storage allocation schemes – that are suitable for systems in which client preferences (movies popularity) is changing over time. Better performance of the VoD system can be achieved also by data sharing techniques. i.e., a single stream is used to service several clients. This can be done by batching (see e.g., in [22]), stream-merging [1], or buffering [3, 14]. Each of these techniques can be applied independently of the data placement scheme, to improve the overall performance of the system.

Our work is also related to the study of set-cover [8] and subset-selection [13, 17, 21]. Generally, in traditional covering or selection problems, the input is a collection of sets  $S = \{S_1, S_2, \dots, S_m\}$  over elements  $U$ , and the goal is to find a subset of  $S$  that covers as many elements from  $U$  as possible. Given the selected subset, an element is either covered (if it is a member of a selected set) or not. The problem considered in this paper is a generalization of set-cover: the elements of  $U$  are not simply covered or not by the selected set, but there is a profit associated with each element from  $U$  and set from  $S$ , and the goal is to find a subset of  $S$  that maximizes the total profit (where the profit from each element is the maximal profit of this element from a selected set). Note that set cover is the special case in which the profits are in  $\{0, 1\}$ .

## 1.2 Our Results

For a VoD system with a single disk we present a hardness proof (Section 2.1) and a  $(1 - 1/e)$ -approximation algorithm (Section 2.2). We then extend the approximation algorithm for systems with variable file sizes and/or storage costs (Section 2.3), and for  $k$ -round broadcasting (Section 2.4).

Rank-related objectives are considered in Section 2.6. We show that the problem of finding an

assignment with maximal lexicographic profile is NP-hard, and present a greedy algorithm that finds an assignment maximizing the number of clients getting their top-preference. The fairness assignment problem is also shown to be NP-hard. Moreover, for some instances some clients are guaranteed to get their  $(M - C + 1)^{th}$  choice. A simple greedy algorithm achieves this bound.

For multiple disks we first present, in Section 3.1, a  $(C - 1)(e - 1)/Ce$ -approximation algorithm. Next, we propose algorithms for solving the problem in two stages. In the first stage an allocation of movies to the disks is determined. In the second stage, given the storage allocation, the bandwidth allocation problem is to decide which of the clients will be serviced by which disk. We present two heuristics for the first task (Section 3.3), and an optimal algorithm for the second task (Section 3.2). The bandwidth-allocation problem can be reduced to a special case of a min-cost max-flow problem. For this flow problem we present and implement an algorithm based on dynamic programming for efficient detection of negative-cost cycles (in our terms, these are profit-improving cycles).

In order to better evaluate the performance of our algorithms we simulated a VoD system, and compared their performances. In our simulated system, as in the real-world, client preferences and payment vectors are power law distributed [25]. We used the Zipf distribution to determine the popularity and payment-readiness of clients [27]. As a result, a few movies are very popular and clients are willing to pay significantly more for watching them. All the algorithms suggested in the paper for the single-round problem, and also some intuitive greedy heuristics, were simulated and their performances was compared. The experiments and their results are described in Section 2.5 (single disk) and Section 3.4 (multiple disks).

## 2 Storage Allocation on a Single Disk

In this section we consider the case in which the system consists of a single disk. The resulting problem is a  $C$ -out-of- $M$  subset-selection problem. For each subset  $S$  of  $C$  movies, let  $b_{S,j}$  be the profit from servicing client  $j$  assuming the movies of  $S$  are stored. Since the client can be transmitted any stored movie,  $b_{S,j} = \max_{i \in S} b_{i,j}$ . Let  $P(S)$  denote the profit gained by storing  $S$  on the disk. If  $L < n$ , then  $P(S)$  is the sum of the  $L$  highest  $b_{S,j}$  values. If  $L \geq n$ , that is, all clients can be serviced, then  $P(S) = \sum_{j=1}^n b_{S,j}$ .

We show that the problem is NP-complete and prove that a greedy algorithm achieves  $(1 - 1/e)$ -approximation to the maximal profit. Note that the problem can be solved by a brute-force  $O(\min(2^M, M^C))$ -time algorithm. Thus, we assume that  $M$  and  $C$  are non constants.

### 2.1 Hardness Proof

**Theorem 2.1** *The storage allocation problem is NP-complete even for a single disk, unit-size files, and unlimited load capacity.*

**Proof:** First, the problem is in NP since it is possible to check efficiently if a given subset of files provides a specific profit. The hardness proof is by reduction from the *maximum coverage* problem. An instance of maximum coverage consists of elements  $U = 1, 2, \dots, n$ , a family of subsets  $S = S_1, S_2, \dots, S_M$  of  $U$ , and a number  $C$ . The goal is to select  $C$  members of  $S$  that cover as many items of  $U$  as possible. Given an instance for maximum coverage we construct the following instance for the storage allocation problem: There are  $|U|$  clients and  $M = |S|$  movies. Define  $b_{i,j}$

to be 1 if  $j \in S_i$  and otherwise  $b_{i,j} = 0$ , also define the load capacity of the single disk to be  $L = n$  and its storage capacity to be  $C$ . We get that a selection of  $C$  subsets covers  $z$  elements if and only if the associated set of movies satisfies  $z$  clients (who are willing to pay 1 for at least one of the selected movies). ■

We note that the above reduction is approximation-preserving, therefore, by [5] the best we can expect is a  $(1 - 1/e)$ -approximation. The next section provides a  $(1 - 1/e)$ -approximation algorithm.

## 2.2 An $(1 - 1/e)$ -approximation Greedy Algorithm

The paper [17] presents a general greedy algorithm for maximizing the profit gained by selecting a subset of  $C$  out of  $M$  elements given that the profit function is nondecreasing, polynomially computable, and sub-modular. A function  $f$  is sub-modular for a given set system if for every collection of subsets  $S$  and  $T$  it satisfies  $f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$ . The greedy algorithm starts with an empty set  $S$  and performs  $C$  iterations in each of which the element  $v$  maximizing  $f(S \cup \{v\}) - f(S)$  is added. This algorithm is shown to achieve approximation  $(1 - 1/e)$  to the optimal profit.

In our setting, as explained above, the profit  $P(S)$  from a subset  $S$  of movies is the sum of  $\min(L, n)$  highest  $b_{S,j}$  values. In order to show that the above greedy algorithm can be used, we note that for any movie  $v$  and set  $S$ ,  $P(S \cup \{v\}) - P(S)$  can be calculated efficiently in  $O(nM)$  time, and we prove below that the profit function  $P$  fulfills the requirements in [17].

**Claim 2.2** *The profit function  $P$  is nondecreasing, polynomially computable, and sub-modular.*

**Proof:** Clearly,  $P$  is nondecreasing since by adding more movies to the stored set, clients might be assigned to more profitable movies. Also,  $P$  is computable in  $O(nM)$  by summing the most profitable available movie for each of the  $n$  clients. To show that  $P$  is sub-modular, assume w.l.o.g. that  $\min(L, n) = L$  (the proof for the other case is identical). Consider two sets  $S, T$ . Assume that out of the  $L$  clients contributing to  $P(S \cup T)$ ,  $n_S$  clients are watching a movie in  $S \setminus T$  and their total profit is  $P_S$ ;  $n_T$  clients are watching a movie in  $T \setminus S$  and their total profit is  $P_T$ ; and  $n_2$  clients are watching a movie in  $S \cap T$  and their total profit is  $P_2$ . Clearly,  $n_S + n_T + n_2 = L$ . This implies that out of the  $L$  values contributing to  $P(S \cap T)$ ,  $n_2$  values are the corresponding  $n_2$  values contributing to  $P(S \cup T)$ , whose total profit is  $P_2$ . Similarly, out of the  $L$  values contributing to  $P(S)$ ,  $n_S + n_2$  values appear also in  $P(S \cup T)$  and their total profit is  $P_S + P_2$ , and out of the  $L$  values contributing to  $P(T)$ ,  $n_T + n_2$  values appear also in  $P(S \cup T)$  and their total profit is  $P_T + P_2$ .

Summing up we get that  $P(S) + P(T) = P_S + P_T + 2P_2 + P'$ , where  $P'$  is the profit from the remaining  $(2L - 2n_2 - n_S - n_T) = L - n_2$  values in  $P(T)$  and  $P(S)$  that do not contribute to  $P(S \cup T)$ . Also,  $P(S \cup T) + P(S \cap T) = (P_S + P_T + P_2) + (P_2 + P'')$ , where  $P''$  is the profit from the remaining  $L - n_2$  values in  $P(S \cap T)$  that do not contribute to  $P(S \cup T)$ . Since  $S \cap T \subseteq S$  and  $S \cap T \subseteq T$ , each value contributing to  $P''$  is a potential value in composing  $P'$ . In other words, when composing  $P'$  clients have a wider collection of movies, so  $P'$  might include higher values achieved by selecting movies from  $S \setminus T$  and  $T \setminus S$ . Thus  $P' \geq P''$ , implying  $P(S) + P(T) \geq P(S \cup T) + P(S \cap T)$ . ■

**Corollary 2.3** *The greedy algorithm achieves  $(1 - 1/e)$ -approximation to the single disk storage allocation problem with unit-size movies.*

### 2.3 Storage costs

Another version of the problem is when there is a cost associated with storing movies on the disk. In this section we consider two possible ways to measure this cost. First, the case in which different movie files have different sizes, and the goal is to maximize the profit from a disk having  $C$  storage units. Formally, let  $s_i$  denote the storage requirement of movie  $i$ , then the goal is to select a subset  $S$  of movies such that  $\sum_{i \in S} s_i \leq C$  and  $P(S)$  is maximized. The paper [21] presents a general greedy algorithm for subset selection problems in which (i) elements are weighted, (ii) there is a constraint on their total weight, and (iii) the profit function is nondecreasing, polynomially computable, and sub-modular. The algorithm considers all possible subsets of three elements, extends each such subset greedily, and picks the best output.

In our setting, in each iteration the greedy algorithm adds the file  $f_i$  for which  $(P(S \cup \{f_i\}) - P(S))/s_i$  is maximal. Note that the sub-modularity proof given in Claim 2.2 is independent of the file sizes and is therefore valid also here. Also,  $P(S)$  is nondecreasing and polynomially computable.

Another way to consider storage costs is when there is a cost  $c_i$  associated with each movie  $i$ . The goal is to maximize the profit from the movies minus the total storage cost. I.e., to find the most profitable subset of movies  $S$ , where the profit is defined by

$$P(S) = \sum_{j \text{ is serviced}} b_{S,j} - \sum_{i \in S} c_i \quad (1)$$

This function is still submodular since the second term is modular:  $cost(S) + cost(T) = cost(S \cup T) + cost(S \cap T)$ . The first term is submodular as before. However, in order to apply the greedy algorithm from [21] we need to assume that the storage costs are relatively low so that  $P(S)$  is nondecreasing and it worth using the whole storage capacity. Finally, it is possible to combine storage costs with variable movie-file sizes. We get the following result.

**Theorem 2.4** *The greedy algorithm from [21] achieves  $(1 - 1/e)$ -approximation to the single disk storage allocation problem with variable size movies and/or storage costs for which  $P(S)$  given in Equation (1) is nondecreasing.*

### 2.4 $k$ -round Broadcasting

In previous sections we assumed that every client is watching a single movie. In the  $k$ -round problem, there are  $k$  synchronized rounds and the goal is to maximize the total profit. We assume that each client  $j$  provides the payment vector  $b_{i,j}$ , and is willing to watch any subset  $T$  of the movies in arbitrary order during the  $k$  rounds. The profit from client  $j$  for this service is  $\sum_{i \in T} b_{i,j}$ .

In the following we show that the greedy algorithm from [21] can be applied here. In particular we show that the profit function defined over feasible sets of movies can be calculated efficiently and is sub-modular. Our results hold when the number of rounds  $k$  is a constant or  $O(M)$ , and also when movie-files have variable sizes or storage costs.

For a given set  $S$  of movies, let  $P(S)$  denote the total profit it is possible to gain from  $S$  during all  $k$  rounds. In order to use the greedy algorithm we need to be able to calculate  $P(S)$ . Practically, each client is expecting to watch a different movie in each round; therefore, a client can watch at most  $k' = \min(k, |S|)$  movies during the whole broadcast. If  $L \geq n$  then all clients can be serviced

in each round. The profit  $P_j(S)$  gained from client  $j$  is simply the sum of the top  $k'$  values in  $\{b_{i,j} | i \in S\}$ . Therefore, the total profit is  $P(S) = \sum_j P_j(S)$ .

The other case, in which  $L < n$ , is more complicated as it might be that some clients are serviced only in some of the rounds. For  $k = 1$ , it was clear that the clients with the top  $L$  payments will be serviced. However, for  $k > 1$ , one challenge of the algorithm is to select the set of clients serviced in any round.

We present an algorithm, based on dynamic programming that outputs, for a given set  $S$ , which clients are serviced in which round, what movie each of these clients is watching, and the resulting total profit  $P(S)$ . Assuming that clients do not limit the order in which they wish to watch the movies, we can assume w.l.o.g that in the  $i$ th round in which it is serviced, a client is watching its  $i$ th top selection (in  $S$ ). This implies that for a given set  $S$  of movies, the algorithm only needs to determine how many transmissions will be allocated to each client.

Given the set  $S$ , we consider for each client  $j$  only the top  $k'$  values in  $\{b_{i,j} | i \in S\}$ . For any  $\ell \in \{1, \dots, k'\}$ , let  $H_{\ell,j}$  be the sum of the highest  $\ell$  payments of client  $j$  for movies from  $S$ . Define  $P_{\ell,z}$  to be the maximal possible profit from a total of  $\ell$  transmissions assuming only clients  $1..z$  are in the system. For  $z = 1, \dots, n$ , initialize  $P_{0,z}$  to 0 and  $P_{1,z}$  to be the maximal single payment of any of the first  $z$  clients. Finally, initialize  $P_{\ell,1} = H_{\ell,1}$ , that is, the sum of top  $\ell$  payments of client 1, for  $\ell = 1, \dots, k'$ .

Next, calculate the values of  $P_{\ell,z}$  for  $z = 2, \dots, n$ , and  $\ell = 2, \dots, kL$ , using the formula

$$P_{\ell,z} = \max_{a=1, \dots, \min(\ell, k')} (P_{\ell-a, z-1} + H_{a,z}).$$

In the above formula, we combine for any number  $a$ ,  $1 \leq a \leq \min(\ell, k')$ , the transmission of  $a$  movies to client  $z$ , with the best allocation of  $\ell - a$  transmissions to the first  $z - 1$  clients. The optimal solution is given by  $P(S) = P_{kL, n}$ . For any  $\ell, z$ , the value of  $P_{\ell,z}$  can be calculated in  $O(k')$ , thus, the whole table is calculated in time  $O(Lkk') \leq O(Lk^2)$ . In order to determine the actual transmission schedule, the number of transmissions per client should be traced for each entry of the table  $P$ .

Once it is determined in how many rounds each client is going to be serviced, the schedule is done greedily: in each round, the  $L$  clients with the maximal number of remaining views are serviced. Since each client gets at most  $k'$  broadcasts and the total number of broadcasts is  $kL$ , it is possible to show (using exchange arguments) that this greedy approach always terminates with a valid transmission schedule.

**Corollary 2.5** *Given a feasible subset of movies  $S$ , it is possible to determine in time  $O(Lk^2)$  what is the profit  $P(S)$  that can be gained from storing  $S$  and how to achieve this profit.*

Finally, in order to use the greedy algorithm, we also need to show that the profit function is submodular and nondecreasing. In fact, the proof of Claim 2.2 is valid, only replace the number  $L$  of transmissions with  $\min(kL, k'n)$ .

**Claim 2.6** *The profit function  $P$  in  $k$ -round broadcasting is sub-modular.*

As in the single round problem, the following is valid also for variable size movie-files having storage costs (assuming that  $P(S)$  is non-decreasing).

**Theorem 2.7** *The greedy algorithm from [21] achieves  $(1 - 1/e)$ -approximation to the single disk storage allocation problem for  $k$ -round broadcasting.*

## 2.5 Experiments for a Single Disk

We have implemented the greedy algorithm described in Section 2.2 and compared it with some natural heuristics. Our main challenge was to create instances reflecting real life scenarios. As suggested by Zipf [27], human behavior and preferences tend to have a power-law distribution; that is, there are a few movies that are very popular and clients are willing to pay significantly more for watching them. A recent study of user behavior in VoD systems [25] confirmed this suggestion. In order to create such instances we used Zipf distribution functions. In a Zipf distribution over  $M$  items with parameter  $0 \leq \theta \leq 1$ , the probability of the  $i$ -th item is  $p_i = c/(i^{1-\theta})$ . The normalization constant  $c$  ensures that all probabilities sums up to 1, that is  $c = 1/(\sum_{1 \leq i \leq M} 1/(i^{1-\theta}))$ . When  $\theta = 1$  the result is a uniform distribution, while as  $\theta$  decreases and approaches 0, the distribution is more skewed.

Two Zipf functions over  $M$  elements were used in our experiments. The first one was used to create the values  $b_{i,j}$  associated with a client  $j$  whose total budget is given (a higher  $\theta$  value implies more uniform payments). A second Zipf was used to define, for each  $1 \leq i \leq M$ , what is the probability that movie  $i$  is the client's top preference, and was used to create the client's ranking. The first distribution was used to match the  $i$ -th ranked movie to the  $i$ th largest  $b_{i,j}$  value.

We compared the greedy algorithm with some intuitive heuristics. The first, denoted *Greedy Profit*, selects the  $C$  movies with the highest total payments (given by  $\sum_j b_{i,j}$ ). The second, denoted *First Choice*, selects the  $C$  movies that were ranked as first by most clients. It can be shown that this algorithm maximizes the number of clients that get their top preference movie. For small values of  $M$ , for which it was reasonable to calculate an optimal solution, we compared the algorithms with the optimal solution.

Some of our experiments considered systems with variable storage costs. For these systems, in the *Greedy Profit* heuristic, the storage cost is reduced from the movie's profit. We also considered a third heuristic, denoted *Take Cheapest*, that simply selects the  $C$  cheapest movies. The real-life scenario guided us also when determining the storage costs – popular movies are more expensive than unpopular movies.

Finally, the other parameters of our system were as follows: The number of clients,  $n$ , and the load capacity,  $L$ , were both set to 1000. The number of movies,  $M$ , was set to 200, and the storage capacity,  $C$  was varying between 1 and 150. The  $\theta$  parameters of the Zipf functions were varying between 0.2 and 0.8.

**Experimental Results:** Figure ?? shows two sample results of our experiments for a single disk with no storage costs. In all the experiments the profit achieved by the greedy algorithm was always higher than those achieved by the other two heuristics. Figure ??(a) presents one extreme, in which all clients have the same budget and share the same popularity distribution. Moreover, both Zipf functions are highly skewed (with  $\theta = 0.2$ ). In this non-challenging instance, the movies selected by all three algorithms were close to coincide and the achieved profit is similar. Figure ??(b) presents the results for a more challenging instance, in which 10% of the clients are *rich and strange*: the budget of a rich client is 10 times higher than the budget of a non-rich client, and their preferences are totally different – determined by the same Zipf function but applied on a reversed order of the



movies. For such instances the superiority of the greedy algorithm is significant. In general, we found out that the more the sets selected by the three algorithms differ from each other, the higher the gap between the profits. We also found out that the first choice heuristic performs better than the greedy profit one when the preferences are more skewed.

Figure ?? shows two sample results of our experiments for a single disk with storage costs. In both experiments all clients have the same budget and preference distributions that are highly skewed ( $\theta = 0.2$ ). In experiment (b) the storage costs are on average 2.5 times higher than the costs in experiment (a). Once again, the greedy algorithm achieves the best results. It is interesting to note the good performance of the simple *take cheapest* heuristic when costs and available storage become higher.

Finally, we note that for small values of  $M$  (at most 15) we calculated also the optimal profit and find out that the greedy algorithm performs much better than its theoretical bound – within 5% of the optimal profit, for variable values of  $n$  and payments distributions.

## 2.6 Rank-Related Objectives

In this section we consider a variant in which, instead of specifying its potential payments, each client provides an ordered tie-free list of preferences of the movies. That is, a list of length  $M$  in which the first element is the 1<sup>st</sup> choice of the client, etc. In this section we consider several objectives that are suitable for this variant.

### 2.6.1 Lexicographic Profile

The profile of an assignment is a vector in which the  $j$ th element is the number of clients allocated their  $j$ -ranked movie. An assignment is *rank-maximal* if it has the maximal possible lexicographic profile. We show that the problem of finding a rank-maximal assignment is NP-hard and present a greedy algorithm that finds an assignment that maximizes the first element in the profile.

**Theorem 2.8** *Finding a rank-maximal assignment is NP-hard.*

**Proof:** The proof is by reduction from vertex cover of regular graphs that is known to be NP-hard [6]. Given a  $d$ -regular graph  $G = (V, E)$  and the question whether  $G$  has a vertex cover of size  $C$ , construct the following instance of the assignment problem: The VoD system consists of a single disk having  $C$  storage units. There are  $2|E|$  clients, two for each edge, and  $|V|$  movies, one for each vertex. The preference-lists of the two clients originated from an edge  $e = (u, v)$  begin with  $u, v$  and with  $v, u$ . The order of the remaining  $|V| - 2$  vertices in each of these lists is arbitrary.

**Claim 2.9** *The graph  $G$  has a vertex cover of size  $C$  if and only if the profile of the rank-maximal assignment is  $\langle dC, 2|E| - dC, 0, \dots, 0 \rangle$ .*

**Proof:** First, note that for any selection of  $C$  movies, the number of clients who receive their 1<sup>st</sup> choice movie is exactly  $dC$  – since for any  $v$ , there are exactly  $d$  clients for which  $v$  is the 1<sup>st</sup> choice movie. Having  $2|E|$  clients, it means that  $\langle dC, 2|E| - dC, 0, \dots, 0 \rangle$  is the best possible profile for this instance.

Assume that  $G$  has a vertex cover of  $C$  vertices. In the assignment problem, select the  $C$  movies associated with these vertices. We show that the resulting profile is  $\langle dC, 2|E| - dC, 0, \dots, 0 \rangle$ . As

mentioned above, there are  $dC$  clients whose 1<sup>st</sup> choice movie is one of the  $C$  movies selected. Also, for each client,  $e_j$ , one of the two movies associated with the vertices at the endpoints of  $e$  was selected, therefore, each client is allocated one of its top-2 preferences.

Assume now that there is an assignment with profile  $\langle dC, 2|E| - dc, 0, \dots, 0 \rangle$ . It means that each client is allocated one of its top-2 preferences. That is, for each edge  $(u, v)$ , at least one of  $u, v$  is one of the  $C$  selected movies. Clearly this implies that the movies associated with the selected movies form a vertex cover. ■ ■

We note that the greedy algorithm that runs in  $C$  iterations and selects in each iteration the movie that improves the ranking the most (relative to lexicographic order) maximizes the number of clients getting their 1<sup>st</sup>-choice movie.

### 2.6.2 Fairness

For the variant where the input is given as preference lists (ranking) of the movies by the clients, another possible objective is *Fairness*. An assignment is  $k$ -fair if each client get one of its top  $k$  choices. For example, if all the clients get their first choice except for one client that gets its  $(3M/4)$ -th choice, then the assignment is  $3M/4$ -fair. The goal is to find the minimal  $k$  such that a  $k$ -fair assignment exists. In other words, to find a  $k$ -fair assignment with a minimal value of  $k$ .

We first show that the fairness problem is NP-hard for any  $k > 1$ . Note that for  $k = 1$  the problem is simply to determine if the set of first-choice movies is of size at most  $C$ , which can be done in linear time.

**Theorem 2.10** *The fairness problem is NP-hard*

**Proof:** The proof is by reduction from vertex cover. Given a graph  $G = (V, E)$  and the question whether  $G$  has a vertex cover of size  $C$ , construct the following instance for the fairness problem: There are  $|E|$  clients, one for each edge, and  $|V|$  movies, one for each vertex. The preference-list of client  $j$  originated from the edge  $e_j = (v_{j1}, v_{j2})$  begins with  $v_{j1}, v_{j2}$ . The order of the remaining  $|V| - 2$  vertices is arbitrary. It is easy to verify that a vertex cover of size  $C$  exists if and only if a 2-fair assignment exists when  $C$  movies can be selected. ■

Next, we show that no algorithm that packs  $C$  movies can be better than  $(M - C + 1)$ -fair. We will do so by describing an instance for which every solution is at most  $M - C + 1$ -fair. The instance consists of  $n = M!$  clients whose preference lists are the  $M!$  possible permutations of  $1 \dots M$ .

For each selection of  $C$  of movies there are at least  $C!$  clients for which these movies form the tail of their preference list. Each of these clients will be  $M - C + 1$ -fair at most.

On the other hand, the following naive algorithm achieves at least  $M - C + 1$ -fairness for all instances.

**Algorithm 2.1 (naive algorithm for fairness)**

1. Initialize  $S \leftarrow \emptyset$ .
2. For  $i : 1 \rightarrow C$  do
  - 2.1. Add to  $S$  a movie that improves the fairness the most.

**Theorem 2.11** *The greedy algorithm achieves  $M - C + 1$ -fairness.*

**Proof:** In fact, the following analysis is valid for any selection of  $C$  different movies: Consider the preference list of any client. Since  $C$  different movies are selected by the algorithm, in the worst case these are the  $C$  movies forming the tail of its list, thus at least one of its top  $M - C + 1$  movies is selected. ■

### 3 Multiple Disks

We turn to consider the more general case where the system consists of  $N$  identical disks, each having a limited storage capacity,  $C$ , and a limited load capacity,  $L$ . We first present an approximation algorithm based on selecting a-priori the movie to be seen by each client, such that these selections are guaranteed to be granted and their total profit is within ratio  $(C - 1)(e - 1)/Ce$  from the optimal profit. Next, we propose algorithms for solving the problem in two stages. In the first stage a storage allocation of movies to the disks is determined. In the second stage, given the storage allocation, the problem is to allocate the bandwidth, that is, to decide which of the clients will be serviced by which disk. We present two heuristics for the first task and an optimal algorithm for the second task.

Throughout this section we assume that  $NL \geq n$ , that is, the system has enough streaming capacity to service all clients. This assumption is valid in real VoD systems, and enable us to focus on the problem of best exploiting this streaming capacity.

#### 3.1 A $(C - 1)(e - 1)/Ce$ -Approximation Algorithm

The paper [18] considers systems in which each client is interested in a single movie and the goal is to maximize the number of serviced clients. The *Moving Window* algorithm (MW) presented in [18] assigns movies to the disks in a way that satisfies all clients whenever  $C \cdot N \geq M + N - 1$ . This algorithm is used as a subroutine in our algorithm for the problem with client preferences.

**Algorithm  $\mathcal{A}_{MW}$ :**

1. Let  $M' = N \cdot (C - 1) + 1$ . Use the greedy algorithm (Section 2.2) to select  $M'$  movies, assuming a single disk with storage capacity  $M'$ . Let  $S_{M'}$  be the set of selected movies.
2. For each client  $j$ , let  $i_j$  be the movie in  $S_{M'}$  for which  $b_{i,j}$  is maximal. In other words,  $i_j$  is the movie achieving the maximal profit from client  $j$  out of the movies in  $S_{M'}$ .
3. Define an input for the MW algorithm: The single request of client  $j$  is for the movie  $i_j$ .
4. Run the MW algorithm. Since  $M'$ , the total number of different movies requested, fulfills  $C \cdot N \geq M' + N - 1$ , the MW algorithm terminates with an assignment in which each client  $j$  is allocated a transmission of  $i_j$ .

In the first step the algorithm selects the movies whose copies will be stored. Next, the algorithm calculates, based on client preferences, how many transmissions from each of these movies will be required. The result is an input to the MW algorithm, that is guaranteed to halt with a storage enabling all these transmissions. Recall that that  $NL \geq n$  so all clients can be serviced.

**Theorem 3.1** *Algorithm  $\mathcal{A}_{MW}$  provides  $(C-1)(e-1)/Ce$ -approximation to the optimal profit on multiple disks.*

**Proof:** In the first step the greedy algorithm is used to select  $M' = N \cdot (C-1) + 1$  movies. By Theorem 2.3, the profit from these movies is within  $(1-1/e)$ -ratio from the optimal profit from any set of  $M'$  movies. The moving window algorithm guarantees that each client will get his top choice from the  $M'$  selected movies. An optimal solution can pack at most  $C \cdot N$  movies. Therefore, the approximation ratio of our algorithm is at least

$$\frac{alg}{opt} \geq \frac{N \cdot (C-1) + 1}{N \cdot C} \cdot \frac{e-1}{e} \geq \frac{(C-1)(e-1)}{Ce}$$

■

### 3.2 Optimal Bandwidth Allocation for a Given Storage Allocation

In this section we present an algorithm for finding an optimal *bandwidth allocation* for a given storage allocation. A bandwidth allocation is also denoted *client assignment*, since by assigning a client to a disk it is granted one stream of unit bandwidth. It is easy to see that this problem is not trivial in a sense that a greedy algorithm cannot do the job. We present a pseudo-polynomial time algorithm based on reducing the problem to a min-cost max-flow problem. The graph in the resulting flow problem has a special structure, that enables simple detection of negative cost cycles by dynamic programming. In our experiments, the min-cost max-flow problem was solved using this method.

The system is represented by a complete bipartite  $G = (U \cup V, E)$  where each client is represented by a vertex in  $U$ , and each disk is represented by a vertex in  $V$ . Let  $M_k$  denote the set of movies stored on disk  $k$ . For each  $j \in \{1, \dots, n\}$  and  $k \in \{1, \dots, N\}$ , the edge  $(u_j, v_k)$  has weight  $w_{j,k} = \max_{i \in M_k} b_{i,j}$ . That is, the weight represents the profit gained if client  $j$  is assigned to disk  $k$  - and will naturally watch the most profitable movie among those stored on  $d_k$ . A valid assignment is a set of edges  $E'$  that forms a *semi-matching* [10] with the following properties:

- (P<sub>1</sub>) Each  $u_j \in U$  is an endpoint of at most one edge in  $E'$ . These constraints imply that each client is watching at most one movie.
- (P<sub>2</sub>) Each  $v_k \in V$  is an endpoint of at most  $L$  edges in  $E'$ . These constraints imply that each disk is servicing at most  $L$  clients - as required by its load capacity.

The goal is to find a set  $E'$  with a maximal total weight - reflecting the resulting profit of the assignment. This problem can be represented as a min-cost max-flow problem (see Figure 1): The network consists of the bipartite  $G$ , a source  $s$ , connected to all the vertices of  $U$ , and a sink  $t$  connected from all the vertices of  $V$ . The capacities of each edge  $(s, u_i)$ , or  $(u_j, v_k)$  is 1. The capacity of each edge  $(v_k, t)$  is the load capacity  $L$ . Edges of type  $(s, u_j)$  or  $(v_k, t)$  have cost 0, while edges of type  $(u_j, v_k)$  have negative cost (= revenue) of value  $-w_{j,k}$ .

Note that the value of the max-flow in this network is  $n = \min\{n, NL\}$ . The following claim proves the reduction to the flow problem.

**Claim 3.2** *A flow having cost  $-C$  corresponds to client assignment achieving profit  $C$ , and a client assignment achieving profit  $C$  corresponds to a valid flow in the network whose cost is  $-C$ .*

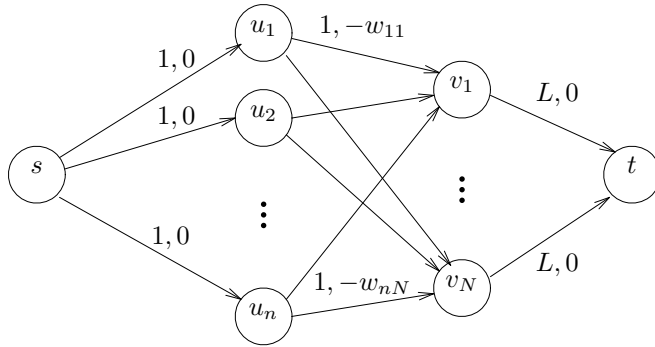


Figure 1: A min-cost max-flow problem representing the bandwidth allocation problem. Each edge is labeled by its capacity and cost.

**Proof:** Given a flow, since all  $(s, U)$ -edges have capacity one, then the flow on all  $(s, U)$ - and  $(U, V)$ -edges must be 0 or 1 (all capacities are integers in  $\{0, 1\}$ ). The flow cost is exactly  $\sum_{j,k} -w_{j,k}$  summing over all  $j, k$  pairs for which one unit of flow travels from  $u_j$  to  $v_k$ . This flow induces the following assignment: if the flow on the edge  $(s, u_j)$  equals 1, then client  $j$  is serviced by the disk  $k$  for which the edge  $(u_j, v_k)$  carries a flow of value 1. This disk transmits to  $j$  the most profitable movie for  $j$  among the movies stored on  $d_k$ . Thus, this client contributes  $w_{j,k}$  to the total profit. The edge  $v_k, t$  has capacity  $L$  guaranteeing that disks do not exceed their load capacity. The other direction is similar - given a client assignment achieving profit  $C$ , determine the flow to be 1 on an edge  $(s, u_j)$  iff client  $j$  is serviced, and on an edge  $(u_j, v_k)$  iff it is serviced by  $d_k$ . The flow on a  $(v_k, t)$ -edges equals the number of clients serviced by  $d_k$ . Since in a valid assignment, each disk is servicing at most  $L$  clients, this flow is at most  $L$ . It is easy to verify that the above flow is valid and has cost  $-C$ . ■

In particular, the min-cost max-flow induces an optimal assignment of clients to disks in which  $n$  clients are serviced. In the following we describe the technique we used in order to solve the flow problem. This technique is tailored for bipartite graphs and is simpler than the general method of detecting negative cost cycles [15]. We also find it more efficient in our experiments.

For a given semi-matching  $E'$ , we are looking for negative cost cycles alternating between vertices of  $V$  and  $U$ , that is, of type  $P = (\{v_1, u_1\}, \{u_1, v_2\} \dots \{u_\ell, v_1\})$  with  $v_i \in V, u_i \in U$  and  $\{v_i, u_i\} \in E'$  for each  $i$ . Each such cycle corresponds to migrating the service of client  $u_j$  from disk  $v_j$  to disk  $v_{j+1} \pmod{\ell}$ . Note that the load on each of the disks is preserved. If the total profit of applying these migrations is positive, then the corresponding cycle has negative cost and is therefore a *profit-increasing cycle* (PIC). A similar method was used in [10] for finding an optimal semi-matching in a bipartite. However, in [10], edges have uniform costs and the goal is to find cost reducing paths (not cycles) for which only the load on the endpoint nodes is changing.

**Optimal Bandwidth-Allocation Algorithm:** (i) find an initial client assignment having properties  $(P_1)$  and  $(P_2)$  above, (ii) improve it by finding *profit increasing cycles* and apply them on the graph until no profit increasing cycle exists. The correctness of this algorithm follows from the correctness of the general method of removing negative cost cycles [15].

(i) **Finding an Initial Feasible Assignment:** This step is done greedily by assigning in each step the next client  $j$  to the disk  $k$  enabling the highest  $w_{j,k}$  value.

(ii) **Finding Profit-Increasing Cycles:** Our algorithm for finding profit increasing cycles is based on dynamic programming. This technique ensures that every potential sub-path is considered only once, and therefore the whole detection is practically efficient. Define  $\delta_{i,j}^k$  as the maximal possible change in the total profit achieved when one unit of load is transmitted from disk  $i$  to disk  $j$  along a path in which  $k$  clients are re-assigned. In other words,  $\delta_{i,j}^k$  measures the change in the profit achieved by an alternating path of  $2k$  edges that starts at vertex  $d_i$  and ends at vertex  $d_j$ . Note that the  $\delta$  values might be negative - if any such path results in decreasing the total profit.

Initially, the values of  $\delta_{i,j}^1$  are calculated for all pair  $i, j$  of disks. For each pair, all  $n$  clients are considered. Formally, recall that  $M_j$  denotes the set of movies stored on disk  $j$ , then

$$\delta_{i,j}^1 = \max_{\ell=1}^n \{ \max_{z \in M_j} b_{\ell,z} - \max_{z \in M_i} b_{\ell,z} \}.$$

For  $k > 1$ , the values of  $\delta_{i,j}^k$  are calculated using the following recursion:

$$\delta_{i,j}^k = \max_{h=1}^N \{ \delta_{i,h}^{k-1} + \delta_{h,j}^1 \}$$

That is, every path from  $i$  to  $j$  in which  $k$  clients are reassigned can be viewed as a concatenation of a path from  $i$  to some disk  $h$  in which  $k - 1$  clients are reassigned, and a path from  $h$  to  $j$  in which one client is reassigned.

In order to detect a profit increasing cycle, the values  $\delta_{i,i}^k$ , that lay in the diagonal of the  $\delta$ -table, are considered after  $\delta_{i,j}^k$  is calculated for  $k > 1$ . If a positive value is found, a profit-increasing cycle is induced (and can be retrieved by additional  $O(1)$  book-keeping along the calculation). Else, the algorithm proceeds to calculate the table  $\delta_{i,j}^{k+1}$ . The maximal length of a PIC is  $2N$  edges, therefore, if no positive  $\delta_{i,i}^k$  value is detected for  $k = N$ , the algorithm terminates with an optimal assignment.

The time complexity of calculating the  $N \times N$  table  $\delta_{i,j}^1$  is  $O(N^2n)$ , the time complexity of calculating each entry of the  $N \times N$  table  $\delta_{i,j}^k$  is  $O(N)$ , therefore the whole calculation of  $\delta_{i,j}^k$  takes  $O(N^3)$ . Assume that the shortest PIC has  $2k$  edges, then the total time of detecting this path is  $O(N^2n) + O(kN^3)$ . If no PIC exists, the algorithm terminates after  $O(N^2n) + O(N^4)$  operations. We note that this time complexity is higher than the time complexity required to detect a cost-reducing path in the semi-matching problem [10] because now the profit from the internal disks in the path is changed, and in the semi-matching only the load on the endpoint vertices is affected. Due to this crucial difference, the cost of using any sub-path of any length must be considered. We also note that in practice we found the above algorithm faster than our implementation of the general method of detecting negative cost cycles from [15].

### 3.3 Heuristics for Initial Storage Allocation

In this section we describe the two heuristics we have used to determine an initial storage as a preprocessing for the optimal bandwidth allocation algorithm described in Section 3.2. The goal is to determine which movie files will be stored on each disk, in a way that achieves high profit

when combined with an optimal bandwidth allocation. The first heuristic is based on a round robin algorithm, the second heuristic uses a variant of the moving window algorithm [18].

Our experiments show that both algorithms are doing a very good job and have similar performances, with a slight advantage to the second heuristic.

**Weighted Round Robin Assignment:** The Weighted Round Robin assignment algorithm is a simple and fast method for determining an initial storage allocation. It first decides what are the  $NC$  copies of movies that will be stored and then distributes them on the disks using round robin. Assume that  $n_i \leq N$  copies of movie  $i$  need to be stored, then  $n_1$  copies of  $f_1$  are stored on  $d_1, \dots, d_{n_1}$ ;  $n_2$  copies of  $f_2$  are stored on  $d_{n_1+1}, \dots, d_{n_1+n_2}$ , etc., where the calculation is modulo  $N$ . In order to determine the value of  $n_i$  we first use the greedy algorithm (Section 2.2) to select  $NC$  movies, assuming a single disk with storage capacity  $NC$ , and then each client selects its top preference from this set. We get a vector of 'votes' that sums up to  $n$ . This vector serves as input for the *apportionment problem* [11], whose solution determines, for each  $i$ , the value of  $n_i$  in a way that it is proportional to the number of votes for  $f_i$ , and  $\sum_i n_i = NC$ .

**Initial Assignment using the Moving Window Algorithm:** In section 3.1 we showed that if we run the Moving Window algorithm with  $M' = N \cdot (C - 1) + 1$  then we get at least  $(C - 1)/C$ -fraction of greedy's profit on a single disk having storage  $M'$ . When the value of  $M'$  increases then clients have a larger variety of movies but on the other hand there is no guarantee that the MW algorithm will succeed in granting all requests for these movies. We suggest the following heuristic: Initially, as in the round-robin solution, select  $M' = NC$  movies, and let each client select its top preference among this set. Then run the MW algorithm for the resulting set of requests. Indeed, the MW algorithm might allocate to some movies less transmissions than their client demand. However, as our experiments reveal (see Section 3.4) this heuristic performs very good in practice.

### 3.4 Experiments for Multiple Disks

Our experiments compare the performance of all the algorithms and heuristics suggested for multiple disks. As described in Section 2.5 we used two Zipf distributions in order to build the client preferences and payment vectors in a way that reflects a real life scenario in which few movies are very popular, and clients are willing to pay significantly more for their top choice movies. In our experiments we tried to compare the various algorithms and to find out which algorithm performs better for specific system parameters and payment vector distributions.

For the algorithms that are based on two phases (finding a storage allocation and then assign the bandwidth to clients), the challenge was to isolate the contribution of each phase to the overall performance. In order to do this we combined two strategies: (i) we replaced the min-cost max-flow algorithm with a simple greedy algorithm that for a given storage allocation iterates over the clients and in each step allocates the next client to the most profitable (and available) disk. (ii) we run each of the two storage allocation algorithms with the above greedy algorithm and with the min-cost max-flow algorithm. Therefore, in each experiment we tested 5 algorithms:

1. The  $(C - 1)(e - 1)/Ce$ -approximation algorithm, that runs the MW algorithm with  $M' = C(n - 1) + 1$ .
2. Storage allocation by weighted round-robin, and optimal bandwidth allocation.

3. Storage allocation by weighted round-robin, and greedy bandwidth allocation.
4. Storage allocation by MW algorithm with  $M' = CN$ , and optimal bandwidth allocation.
5. Storage allocation by MW algorithm with  $M' = CN$ , and greedy bandwidth allocation.

The other parameters in our experiments were as follows: The number of clients,  $n$ , was set to 1000. The number of movies,  $M$ , was set to 200, the number of disks,  $N$ , was varying between 4 to 20. The load capacity  $L$  was defined to be  $n/N$ . That is, all clients can be serviced independent of the number of disks. The storage capacity,  $C$ , varies from 1 to 10. Also, in all the experiments all clients have the same budget,  $\forall j, \sum_i b_{i,j} = 1$ . This enables us to isolate the influence of each of the varying parameters on the system's profit.

**Experimental Results:** Figures 2 and 3 present the experimental results for two different popularity and budget distributions ( $\theta$  values), and two different storage capacities. The graphs present the profit achieved as a function of the number of disks  $N$ . In all four experiments the total number of broadcasts is  $n = NL = 1000$ , therefore, a system with many disks is more powerful only since it has more storage capacity. In other words, the bandwidth is better exploited (profit function is increasing), only thanks to the increased storage.

In all four experiments we see that the combination of the MW algorithm with the optimal bandwidth allocation achieves the best results. In particular, the MW algorithm is a better storage allocation algorithm than the round-robin algorithm. We also learn that the greedy bandwidth allocation algorithm achieves fair results. Clearly, it is worse than the optimal one, but the average gap in the profit is only about 10%, and it is much faster ( $O(nN)$  vs.  $O(N^2n) + O(N^4)$  for the optimal). From comparing systems with low and high storage capacities ((a) vs. (b) experiments), we learn that the performance of the MW algorithm with  $M' = N \cdot (C - 1) + 1$  is improved, relative to the other algorithms, when the storage capacity is high.

From comparing systems with different client-preference distributions (Fig. 2 vs. 3), we learn that when  $\theta$  is increased (movie popularity and clients' payments are more uniform) then the total profit is decreased. This can be explained by observing that in high- $\theta$  instances there is no small set of movies that is highly requested and has high payment readiness. Low  $\theta$  values are more challenging, and the results show higher gaps between the different algorithms. This is explained by the fact that any deviation in the allocation might result in large gaps in the profit, while for high  $\theta$ 's the storage allocation is less crucial, since any movie is among the top choices of some clients, and the payments for closely ranked movies are not significantly different. Unlike the single disk case, where solving the problem for low  $\theta$  values was done efficiently even by simple heuristic, for systems with multiple disks we conclude that selecting the right algorithm is crucial as  $\theta$  decreases and as the system's resources are enlarged.

## Acknowledgments

We thank Seffi Naor and Hadas shachnai for valuable discussions and helpful comments.

## References

- [1] A. Bar-Noy and R. E. Ladner. Efficient algorithms for optimal stream merging for media-on-demand. *SIAM Journal on Computing*, 33(5): 1011-1034 (2004)



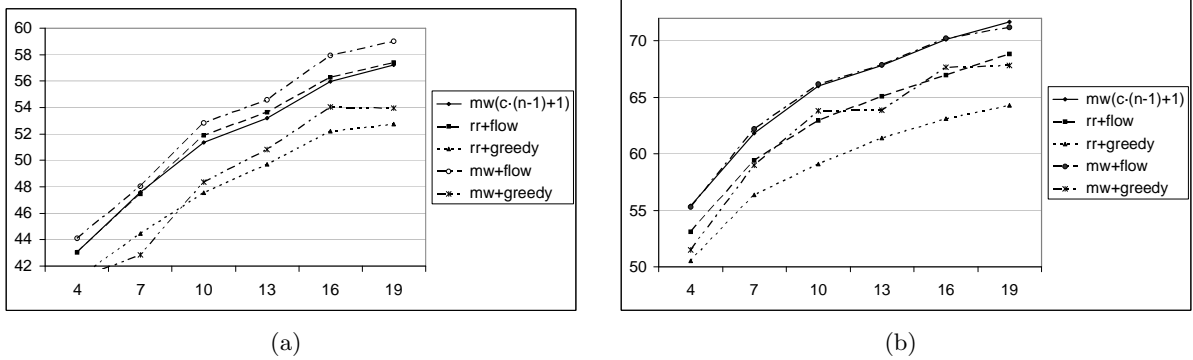


Figure 2: Profit as a function of  $N$ , with fixed  $NL$ , and  $\theta = 0.3$ , (a)  $C = 3$ , (b)  $C = 9$ .

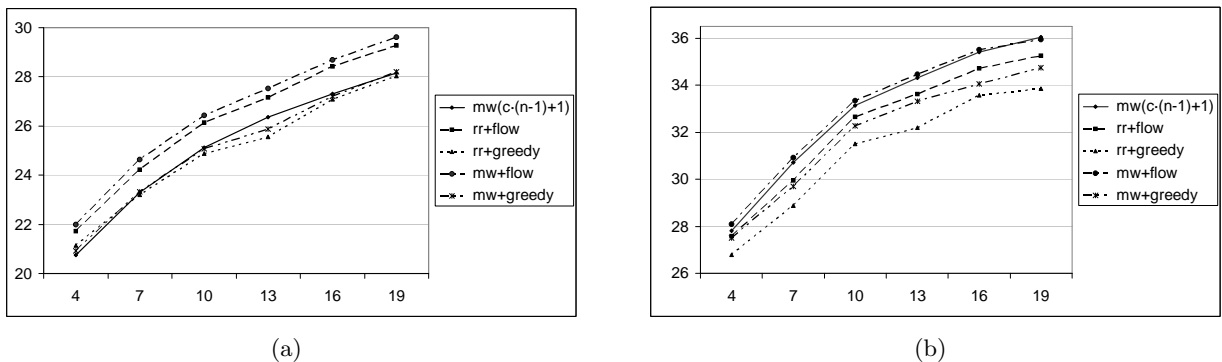


Figure 3: Profit as a function of  $N$ , with fixed  $NL$ , and  $\theta = 0.5$ , (a)  $C = 3$ , (b)  $C = 9$ .

[2] A. Bar-Noy, J. Goshi, and R. E. Ladner. Off-line and on-line guaranteed start-up delay for media-on-demand with stream merging. *Journal of Discrete Algorithms*, 4(1):72–105, 2006.

[3] A. Bar-Noy, R. E. Ladner, and T. Tamir. Scheduling techniques for media-on-demand. In *Proc. of the 14th ACM-SIAM Symposium on Discrete Algorithms*, 791–800, 2003.

[4] C.F. Chou, L. Golubchik, and J.C.S. Lui. A performance study of dynamic replication techniques in continuous media servers. *Proc. of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (IEEE MASCOTS)*, pp. 256–264, 2000.

[5] U. Feige, A Threshold of  $\ln n$  for Approximating Set Cover. *Journal of the ACM*, 45(4), 634–652, 1998.

[6] U. Feige, Vertex Cover is Hardest to Approximate on Regular Graphs. *Technical Report MCS03-15, Computer Science and Applied Mathematics, The Weizmann Institute of Science*, 2003

[7] L. Golubchik, S. Khanna, S. Khuller, R. Thurimella, and A. Zhu. Approximation algorithms for data placement on parallel disks. In *Proc. of SODA*, 223–232, 2000.

[8] M.R. Garey and D.S. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*, W. H. Freeman and Company, San Francisco, 1979.

- [9] S. Ghandeharizadeh and R.R. Muntz. Design and implementation of scalable continuous media servers. *Parallel Computing J.*, 24(1):91–122, 1998.
- [10] N.J.A. Harvey, R.E. Ladner, L. Lovász, and T. Tamir, Semi-Matchings for bipartite graphs and load balancing, *Journal of Algorithms* 59(1):53–78, 2006.
- [11] T. Ibaraki and N. Katoh. *Resource allocation problems - algorithmic approaches*. The MIT Press, 1988.
- [12] S. Kashyap and S. Khuller. Algorithms for non-uniform size data placement on parallel disks. *Journal of Algorithms* vol. 60(2): 144-167, 2006.
- [13] S. Khuller, A. Moss, and J. Naor, The budgeted maximum coverage problem. *Inf. Process. Letters* vol. 70(1):39–45, 1999.
- [14] M. Kamath, K. Ramamritham, and D. Towsley. Continuous media sharing in multimedia database systems. In *Proc. of the 4th Intl. Conf. on Database Systems for Advanced Applications*, 1995.
- [15] M. Klein, A primal method for minimal cost flows, *Management Science* 14:205-220, 1967.
- [16] T.C. Little and D. Venkatesh, Popularity-based assignment of movies to storage devices in a video-on-demand system. *Multimedia Syst.* 2(6): 280-287, 1995.
- [17] G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of the approximations for maximizing submodular set functions. *Mathematical Programming*, vol 14:265–294, 1978.
- [18] H. Shachnai and T. Tamir, On two class-constrained versions of the multiple knapsack problem. *Algorithmica* 29(3): 442–467, 2001.
- [19] H. Shachnai and T. Tamir, Polynomial time approximation schemes for class-constrained packing problems. *J. of Scheduling*, vol. 4(6), pp. 313–338, 2001.
- [20] H. Shachnai and T. Tamir, Approximation schemes for generalized 2-dimensional vector packing with application to data placement. In *proc. of RANDOM-APPROX*, pp. 165–177. 2003
- [21] M. Sviridenko. A Note on maximizing a submodular set function subject to knapsack constraint”, *Operations Research Letters* vol. 32:41–43, 2004.
- [22] J.L. Wolf, P.S. Yu, and H. Shachnai. Scheduling issues in video-on-demand systems. *Multimedia Information Storage and Management*, 183–207, 1996.
- [23] J.L. Wolf, P.S. Yu, and H. Shachnai. Disk load balancing for video-on-demand systems. *ACM Multimedia Systems J.*, 5:358–370, 1997.
- [24] Y. Wang, J.C.L. Liu, D.H.C. Du, and J. Hsieh, Efficient video file allocation schemes for video-on-demand services. *J. Multimedia Systems* vol 5. pp:1432–1882. 2004.
- [25] H. Yu, D. Zheng, B.Y. Zhao, W. Zheng. Understanding User Behavior in Large Scale Video-on-Demand Systems. *Eurosys Conference*, 2006.
- [26] X. Zhou and C.Z. Xu, Optimal video replication and placement on a cluster of video-on-demand servers. *Proc. of the 2002 International Conference on Parallel Processing (ICPP’02)*, pp. 547–555.
- [27] G. K. Zipf. Human behavior and the principle of least effort. *Cambridge, Mass.: Addison-Wesley Press*. 1949.