Algorithmica (2001) 29: 442–467
DOI: 10.1007/s004530010057

Algorithmica

On Two Class-Constrained Versions of the Multiple Knapsack Problem[1]

H. Shachnai[2] and T. Tamir[2]

Abstract. We study two variants of the classic knapsack problem, in which we need to place items of *different types* in multiple knapsacks; each knapsack has a limited capacity, and a bound on the number of different types of items it can hold: in the *class-constrained multiple knapsack problem* (*CMKP*) we wish to maximize the total number of packed items; in the *fair placement problem* (*FPP*) our goal is to place the same (large) portion from each set. We look for a *perfect placement*, in which both problems are solved optimally. We first show that the two problems are NP-hard; we then consider some special cases, where a perfect placement exists and can be found in polynomial time. For other cases, we give approximate solutions. Finally, we give a nearly optimal solution for the CMKP. Our results for the CMKP and the FPP are shown to provide efficient solutions for two fundamental problems arising in multimedia storage subsystems.

Key Words. Knapsack, Packing, Approximation algorithms, Resource allocation, Fairness, Utilization, Multimedia on-demand.

1. Introduction

1.1. *Problem Statement.* In the well-known *multiple knapsack problem* (*MKP*) [18], M items of different sizes and values have to be packed into N knapsacks with limited volumes. In this paper we study two variants of the MKP, in which items of M *distinct types* have to be packed into N knapsacks, each having a limited volume and a limited number of compartments; items of different types cannot be placed in the same compartment. Specifically, the input is a universe U, which consists of M distinct types of items, given as the subsets $U_1, \ldots, U_M$; there are $|U_i|$ items of type i, $1 \leq i \leq M$, and $U = U_1 \cup U_2 \cdots \cup U_M$; all items have the same (unit) size and the same value, that is, for all $u \in U$, $s(u) = w(u) = 1$. There are N knapsacks: the jth knapsack, K_j, has the volume V_j, and a limited number of compartments, C_j, in which the items can be placed, $1 \leq j \leq N$. Thus, in the jth knapsack we can place items of at most C_j different types. The output of our optimization problems is a *placement*, which specifies for each knapsack K_j to which types of elements K_j allocated compartments, and how many items of each type are placed in K_j. A placement is *legal* if K_j allocated at most C_j compartments, and the overall size of the items placed in K_j does not exceed V_j, for

[1] A preliminary version of this paper appeared in *Proceedings of FUN with Algorithms*, Isola d'Elba, Italy, June 1998.

[2] Department of Computer Science, The Technion, Haifa 32000, Israel. {hadas, tami}@cs.technion.ac.il.

Received June 1, 1998; revised December 5, 1998. Communicated by N. Megiddo.
Online publication October 6, 2000.

all $1 \leq j \leq N$. A placement determines a subset $U' = U'_1 \cup U'_2 \cup \cdots \cup U'_M$ of U, such that $|U'_i|$ is the number of items packed from U_i.

The two optimization problems studied in this paper are:

The class-constrained multiple knapsack problem (CMKP), in which our objective is to maximize the total size of the packed elements, given by $\sum_{i=1}^{M} |U'_i|$.

The fair placement problem (FPP), where the objective is to maximize the value of $0 < c \leq 1$ such that, $\forall 1 \leq i \leq M,\ |U'_i| \geq c \cdot |U_i|$.

Throughout the paper we assume that $\sum_{i=1}^{M} |U_i| = \sum_{j=1}^{N} V_j$, that is, the total number of items in U equals the total sum of the knapsack volumes. In particular, we look for *a perfect placement*, in which both problems are solved optimally. Indeed, such a placement yields the maximal utilization of the knapsack capacities, i.e., the total occupied volume is $V = \sum_{j=1}^{N} V_j$, and maximal fairness, i.e., $c = 1$.

The assumption $|U| = V$ simplifies the presentation of our results; moreover, any input for the storage management problem that motivated our study, satisfies this assumption. It is important to note, however, that our results hold for general inputs for the CMKP and the FPP, i.e., for any relation between $|U|$ and V. We elaborate on that in the Appendix.

1.2. *Storage Management in Multimedia Systems*. Our two variants of the knapsack problem are motivated by two fundamental problems arising in storage management for multimedia-on-demand (MOD) systems. MOD services are becoming common in library information retrieval, entertainment, and commercial applications. MOD systems are expected to manage with the enormous storage and bandwidth requirements of multimedia data. In addition, MOD servers should support strict timing requirements: each user can choose a program he wishes to view and the time he wishes to view it. The service should be provided within a small latency and guaranteeing an almost constant transfer rate of the data.

In an MOD system a large database of M video program files is kept on a centralized server. Each program file is associated with a popularity parameter, given by $p_i \in [0, 1]$, where $\sum_{i=1}^{M} p_i = 1$. The files are stored on N shared disks. Each of the disks is characterized by (i) its storage capacity, that is the number of files that can reside on it, and (ii) its load capacity, given by the number of data streams that can be read simultaneously from that disk. Assuming that $\{p_1, \ldots, p_M\}$ are known, we can predict the expected load generated by each of the programs at any time.

We wish to define a static allocation of storage and load to each file, so that the load generated due to access requests to the file can be satisfied. Our allocation should enable simultaneous transmissions of as many video programs as possible. Indeed, it should reflect the popularities of the programs, by allowing many transmissions of popular programs, and only few transmissions to the less popular ones. In other words, we would like to achieve fair allocation of the storage and load capacity. Another objective is to maximize the utilization of the load capacity of the system.

The problem of assigning files to disks, so as to maximize utilization (fairness), can be formulated as an instance of the CMKP (FPP): a disk j with load capacity L_j and storage capacity C_j will be represented by a knapsack K_j with volume L_j and C_j compartments. A file i will be represented by a set U_i with size $|U_i|$, which is proportional to the file

popularity. Specifically, $|U| = \sum_{j=1}^{N} L_j$ and $|U_i| = p_i|U|$.[3] A solution to any of our two variants of the knapsack problems will induce a legal static assignment.

1.3. *Related Work.* Previous work on the MKP and other knapsack related problems assume that (i) all items of the same type have to be placed in the same knapsack, and (ii) there is no limit on the number of different types of items that can be placed in one knapsack (see, e.g., [3], [7], [15], [20] and detailed surveys in [18] and [19]).

The special case of the MKP where $N = 1$, known as the classic 0–1 *Knapsack problem*, admits a fully polynomial approximation scheme (FPAS). That is, for any $\varepsilon > 0$, a $(1 - \varepsilon)$-approximation to the optimal solution can be found in $O(n/\varepsilon^2)$, where n is the number of items [6], [7]. In contrast, the MKP is NP-hard in the strong sense, therefore it is unlikely to have an FPAS, unless $P = NP$ [19].

The CMKP is closely related to the *fractional knapsack problem*: this problem can be optimally solved in polynomial time (by a simple greedy algorithm [5]). Indeed, the sets $U_1, \ldots, U_M$ can be replaced by M items of the sizes $|U_1|, \ldots, |U_M|$, where each item can split among several knapsacks. In our generalized version of the fractional knapsack, each knapsack has a limited capacity and a limit also on the number of items it can hold. We show below that this problem is NP-hard.

Other related work deals with *multiprocessor scheduling* [9]–[11], also known as the *minimum makespan* problem: given n processors and m jobs with designated integral processing times, the goal is to schedule the jobs on the processors, such that the overall completion time of the schedule is minimized. We can represent a knapsack by a processor, and each set of items of size k by a job requiring k units of processing time. Hence, there are $n = N$ processors and $m = M$ jobs. The compartment constraint can be represented in the scheduling problem by allowing at most C_j jobs to be scheduled on processor $j, \forall 1 \leq j \leq N$. Note, that minimizing the makespan is equivalent to maximizing the utilization of the knapsack volumes. Previous research on the scheduling problem assumes no bound on the number of jobs which can be allocated to each processor, i.e., $C_i = M, \forall 1 \leq i \leq M$ (a survey appears in [12]). In this case the makespan problem admits a polynomial time approximation scheme [9].

MOD systems have been studied intensively in recent years [1], [8], [16], [17], [21]. However, the assignment problem received only little attention in this context. Specifically, most of the previous work discussed the problem of load balancing on disks, in which the goal is to have the total load on the system distributed evenly among N disks. The first solution proposed for the load balancing problem was disk striping (see, e.g., [2] and [4]), in which the data of each file is distributed over multiple disks. Thus, the heavy load caused by a popular program is shared among these disks.

In [22] dynamic algorithms were suggested for balancing the load in the system. The paper also addresses the problem of determining the number of copies of each file that should be kept in the system; the goal is to have the total storage capacity allocated to f_i reflect its popularity. This criterion can yield poor results when used for solving our optimization problems: intuitively, the algorithm will allocate multiple copies to a popular file, however, these copies may be stored on disks, whose load capacities are

[3] For simplicity, we assume that $p_i|U|$ is an integer (otherwise we can use a standard rounding technique [14]).

small. Consequently, these disks will be overloaded and the system will often reject requests for that file. This is due to the fact that the placement of files on the disks uses as parameters only the file popularities and storage capacities of the disks, while the load capacities are ignored (a detailed example is given in Section 2.3.2).

1.4. *Our Results*. We now summarize the results presented in this paper:

- The CMKP and the FPP are NP-hard.
- For some instances a perfect placement always exists and can be found in polynomial time. Three simple conditions for the existence of a perfect placement are given. For each condition, we show how the CMKP and the FPP can be optimally solved, when this condition is satisfied.
- When the conditions are not met, we derive approximate solutions for our two knapsack problems. The approximation ratio depends on the "uniformity" of the knapsacks. Specifically, given $r > 0$ and $\alpha \geq 1$ such that, $\forall j,\ r \leq V_j/C_j \leq \alpha \cdot r$, we give an algorithm which achieves $(1/\alpha)$-approximation for both the CMKP and the FPP.
- We show that if the number of compartments in each knapsack is at least b, for some $b \geq 1$, i.e., $C_j \geq b$, $\forall 1 \leq j \leq N$, then the CMKP can be approximated to within factor $b/(b+1)$.

The rest of the paper is organized as follows. The hardness results are given in Section 2.4. In Section 3 we discuss several cases in which a perfect placement exists and can be found in polynomial time. A nearly optimal solution for the CMKP is given in Section 4. In Section 5 we describe how our theoretical results can be applied to storage management in MOD systems, and in particular to *heterogeneous* disk subsystems. In Section 6 we give possible directions for future work.

2. Preliminaries. Given N knapsacks with the volumes $V_1, \ldots, V_N$, the *packing potential* of the knapsacks, denoted by V, is the total number of items that can be placed in the knapsacks, i.e., $V = \sum_{j=1}^{N} V_j$. For a universe U of unit size items, partitioned to the sets $U_1, \ldots, U_M$, a solution to the CMKP or the FPP can be represented as two $M \times N$ matrices:

1. The *indicator matrix*, I, a $\{0, 1\}$-matrix, $I_{i,j} = 1$ iff a compartment of K_j was allocated to items of type U_i.
2. The *quantity matrix* Q, $Q_{i,j} \in \{0, 1, \ldots, V_j\}$, $Q_{i,j}$ is the number of items of U_i that are placed in K_j.

A legal placement has to satisfy the following conditions:

- $I_{i,j} = 0 \Rightarrow Q_{i,j} = 0$. This condition reflects the fact that items of U_i can be placed in K_j only if a compartment of K_j was allocated to items of type i.
- For each knapsack K_j, $\sum_i Q_{i,j} \leq V_j$, that is, the total number of items placed in K_j does not exceed its capacity.
- For each knapsack K_j, $\sum_i I_{i,j} \leq C_j$, that is, the number of different types of items placed in K_j does not exceed the number of compartments in K_j.

The matrices I and Q determine a subset of items $U' = U'_1 \cup \cdots \cup U'_M$ which is placed into the knapsacks.

DEFINITION 2.1. Given a solution for the CMKP (FPP), the *packed quantity of* U_i, denoted by Q_i, is the total number of items packed from U_i. Thus, $Q_i = |U'_i| = \sum_{j=1}^{N} Q_{i,j}$.

2.1. *Utilization of a Placement.* Our first measure for the quality of a placement is utilization:

DEFINITION 2.2. The *utilization of a placement* is $\sum_{i=1}^{M} Q_i$.

The maximal possible utilization of a placement is V, meaning that all the packing potential of the knapsacks is exploited. Since $\sum_{i=1}^{M} |U_i| = V$, it also means that exactly $|U_i|$ items from the set U_i are packed. Other placements may utilize only part of the overall packing potential:

DEFINITION 2.3. A placement is *c-utilized* if its utilization equals $c \cdot V$, for some $c \in [0, 1]$.

Our main questions here are: "Can the maximal possible utilization be found in polynomial time?" "Can we find an efficient approximation?"

The CMKP aims at maximizing the utilization of the packing potential: in Section 4 we present a *dual approximation algorithm* for the CMKP. The notion of dual approximation was introduced in [13]. It involves approximating the *feasibility* of a solution for a given problem, rather than its optimality; traditional approximation algorithms seek feasible solutions that are suboptimal, where the performance of the algorithm is measured by the degree of suboptimality allowed.

In a dual approximation algorithm the objective is to find an infeasible solution that is superoptimal; the performance of the algorithm is measured by the degree of infeasibility allowed. The general relationship between traditional (or primal) approximation algorithms and dual approximation algorithms is discussed in Chapter 9 of [12]. The dual approximation algorithm we present in Section 4 is superoptimal for the CMKP. Our algorithm allows a small degree of infeasibility, that is, at most one compartment is added to each of the knapsacks.

2.2. *The Fair Placement Problem.* Our second criterion for measuring the quality of a placement is fairness:

DEFINITION 2.4. A placement is *c-fair*, for some $c \in [0, 1]$, if, for every set U_i, $Q_i \geq c \cdot |U_i|$.

An optimal placement is 1-fair. In a 1-fair placement, for each i, exactly $|U_i|$ items from the set U_i are packed. Since $\sum_{i=1}^{M} |U_i| = V$, it also means that the packing potential of the knapsacks is fully exploited.

Several questions arise when looking for a fair placement: "Does a 1-fair placement exist for any instance of the problem?" "Can we find it efficiently?" "When a 1-fair placement does not exist, can we find (or approximate) an optimal placement efficiently?"

2.3. *Combining Utilization and Fairness*

2.3.1. *The perfect placement problem.* We first explore the relation between the CMKP and the FPP.

DEFINITION 2.5. A *perfect placement* is a placement in which all the items of all the sets are packed, and all the knapsacks are full.

Clearly, any perfect placement is 1-fair and 1-utilized.

We now show that for some instances a perfect placement does not exist. Consider a simple system consisting of two knapsacks, with $C_1 = C_2 = 1$ and $V_1 = V_2 = 10$, and two sets of items: $|U_1| = 15$ and $|U_2| = 5$. The only legal placements are:

1. Each set is packed into a different knapsack. Ten items of U_1 and five items of U_2 are packed.
2. Both compartments are allocated to U_1, or both compartments are allocated to U_2. Clearly, these placements are 0-fair.

Note, that by increasing C_1 to 2 we obtain an instance, for which a perfect placement exists: now we can place items of U_1 into both knapsacks and choose $Q_{1,1} = Q_{2,1} = 5$ and $Q_{1,2} = 10$.

When a perfect placement does not exist, we would like to find the best possible one. However, the two goals of utilization and fairness may conflict. Consider an instance with two knapsacks: $V_1 = 20,\ C_1 = 2;\ V_2 = 10,\ C_2 = 1$; and three sets of items: $|U_1| = 14,\ |U_2| = 14,\ |U_3| = 2$.

A placement which achieves the maximal utilization is presented in Figure 1(a): 28 items are packed, i.e., this placement is $\frac{28}{30}$-utilized. However, it is 0-fair—no element of U_3 is packed. Figure 1(b) presents the best possible placement with respect to fairness. It is $\frac{10}{14}$-fair and $\frac{26}{30}$-utilized. Generally, any c-fair placement is at least c-utilized.

2.3.2. *Simple algorithms.* In this section we show that two simple greedy algorithms are not suitable for the CMKP and the FPP. The first algorithm (presented in [22], in the context of MOD systems) can be used for the FPP: the algorithm attempts to guarantee "fairness" by allocating to each set of items a number of compartments that is proportional to its size. Specifically, using an apportionment procedure, the algorithm first determines the number of compartments, I_i, that will be allocated to U_i, $1 \le i \le M$; then it selects a subset of I_i knapsacks that will store items from U_i. Finally, the volumes of the knapsacks are split among the sets, so as to achieve maximal fairness.

To realize that this algorithm is not suitable for any of our knapsack problems, consider an instance which consists of three sets, $|U_1| = 6$, $|U_2| = |U_3| = 3$, and eight knapsacks, with $C_j = 1$ for $1 \le j \le 8$. The volume of the first knapsack is $V_1 = 5$, and the volumes of the remaining knapsacks are $V_2 = V_3 = \cdots = V_8 = 1$; thus the packing potential of the knapsacks is $V = 12$. The total number of compartments is eight, therefore, by the "number of compartments" criterion, since half of the items belong to U_1, four

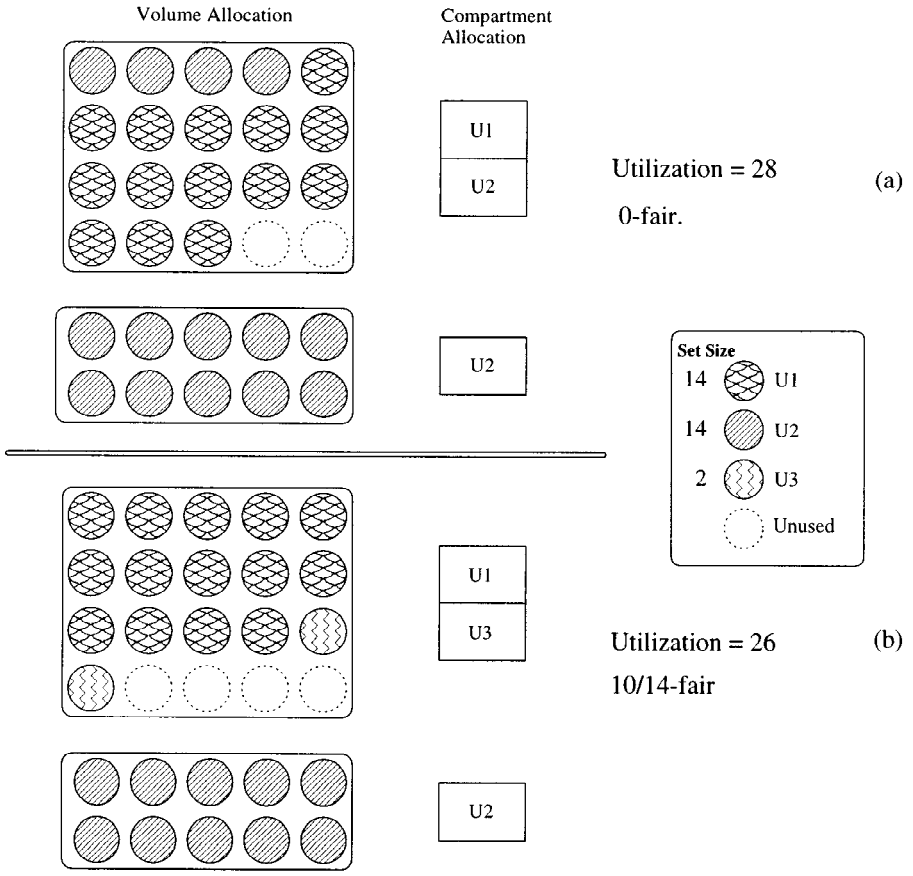

Fig. 1. Maximal utilization versus maximal fairness.

compartments should be allocated to this set, while U_2 and U_3 should be placed into two knapsacks each. Clearly, no fair placement exists under these conditions: the set whose items are stored in the first knapsack is allocated extra volume, while the other two sets are "discriminated."

In the FPP our goal is to find placements in which Q_i, the packed quantity of U_i, reflects its size. Note that in the above example, if the number of compartments allocated to each set of items is not determined prior to the placement, then a *perfect* placement exists: for example, we can place the items of U_1 into K_1 and K_2, items of U_2 into K_3, K_4, and K_5; and items of U_3 into K_6, K_7, and K_8. Indeed, items of U_1 are placed into only two knapsacks, but since one of these knapsacks is large (K_1), all the items of U_1 can be packed.

The second algorithm is based on the *Longest Remaining Time First* (*LRTF*) algorithm [10]. LRTF provides a $(\frac{4}{3} - 1/n)$-approximation for the makespan problem, where n is the number of machines. When LRTF is adopted to the FPP, we place items from the largest remaining set into the knapsack with the largest remaining volume.

Note that this algorithm can yield poor, nonfair placements. Consider an instance with two knapsacks: $V_1 = L + 1, C_1 = L$ and $V_2 = L, C_2 = 1$, for some $L > 1$; suppose that there are $L + 1$ sets, $|U_1| = L + 1$, and in the L remaining sets $|U_i| = 1$. The optimal placement packs the L items of the small sets into K_1 and L items of U_1 into K_2. This is an $(L/(L + 1))$-fair and $(2L/(2L + 1))$-utilized placement. The LRTF algorithm first chooses $Q_{1,1} = L + 1$ and one more item from another set is placed in K_2. Then no available compartments are left in K_2. The resulting placement is 0-fair and $((L + 2)/(2L + 1))$-utilized.

Alternatively, consider a variant of the LRTF, which sorts the knapsacks by the ratio V_j/C_j. Then items from the largest remaining set are placed into the knapsack in which the (updated) ratio V_j/C_j is maximal. Note that this algorithm also yields inefficient solutions for the FPP. Consider, e.g., two knapsacks with $V_1 = 7, C_1 = 1$ and $V_2 = 12, C_2 = 2$; and three sets with $|U_1| = 10, |U_2| = 7$, and $|U_3| = 2$. In the optimal placement U_2 is placed in K_1, while U_1 and U_3 are placed in K_2. The above algorithm initially packs seven items of U_1 into K_1, and cannot complete the packing of all the items. In particular, it is 0-fair for U_3. Using a slightly different rule, which places items from the set with the largest remaining "unpacked fraction" into the knapsack with the largest (updated) V_j/C_j ratio, may increase fairness, with a corresponding decrease in utilization. Still, a more complicated algorithm, which combines sorting by the ratios V_j/C_j with other ideas, can be useful in obtaining approximation algorithms for the CMKP (we elaborate on that in Section 4).

2.4. *Hardness of the Perfect Placement Problem.* The next question we consider is whether we can detect efficiently if a perfect placement exists.

THEOREM 2.1. *Given N knapsacks with the volumes $V_1, \ldots, V_N$, and C_j compartments in knapsack j, and the sets of items $U_1, \ldots, U_M$, it is NP-hard to determine if a perfect placement exists for this instance.*

PROOF. We show a reduction from the *partition problem*, which is known to be NP-hard [6]. The partition problem consists of a finite set A, and size $s(a)$ for each $a \in A$. The problem is to determine if there exists a subset A' of A such that $\sum_{i \in A'} s(i) = \sum_{i \in A \setminus A'} s(i)$.

Given an instance for partition, consider the placement problem consisting of two sets $|U_1| = |U_2|$, and $|A|$ knapsacks with $C_j = 1$ and $V_j = s(a_j), \forall 1 \leq j \leq |A|$. For this problem, every perfect placement induces a desired partition and vice versa. □

Any perfect placement is both 1-fair and 1-utilized. Therefore, the above reduction is suitable for the CMKP and the FPP. We conclude that each of these problems is NP-hard.

3. Finding a Perfect Placement

3.1. *Simple Conditions for the Existence of a Perfect Placement.* In this section we present simple conditions for the existence of a perfect placement. The first one considers

inputs in which the "number of compartments" constraint can be ignored. Clearly, if, for all the knapsacks, $C_j \geq V_j$, then a perfect placement exists, and can be found in polynomial time. (Observe, that if $C_j \geq V_j$ for all $1 \leq j \leq N$, then we can greedily place the sets into the knapsacks, until all the items are packed.)

The next simple condition considers the *sizes* of the different sets.

THEOREM 3.1. *Let $0 < \varepsilon \leq 1$ be the maximal number, such that, for all $1 \leq i \leq M$, $|U_i| \geq (\varepsilon \cdot |U|)/M$. If, for all the knapsacks, $C_j \geq (V_j \cdot M)/(\varepsilon \cdot V) + 1$, then there exists a perfect placement which uses at most $M + N - 1$ different compartments.*

PROOF. Consider the simple greedy algorithm, which packs the sets of items by filling the knapsacks one after the other. Specifically, the knapsack K_j is filled until it contains V_j items, we then continue to pack into K_{j+1}, and so on. Since $\sum_i |U_i| = |U| = V = \sum_j V_j$, the algorithm terminates when the last knapsack, K_N, is filled with the last item of U_M. Thus, all the packing potential of the knapsacks is exploited and all the items are packed. Intuitively, the conditions indicate that even the smallest sets are large enough to fill any knapsack. Formally, $|U_i| \geq (\varepsilon \cdot |U|)/M$ implies that any subset of $C_j - 1$ sets includes at least $(C_j - 1)|U| \cdot \varepsilon/M = (C_j - 1)V \cdot \varepsilon/M$ items. Since, $C_j \geq (V_j \cdot M)/(\varepsilon \cdot V) + 1$, the subset's total size is at least V_j. The additional compartment is needed for a small number of items of the first set placed in K_j—for this set, the size is not predicted, since some of its items were already placed in previous knapsacks.

If all the items of one set are placed in one knapsack, we use only one compartment for this set. If it is placed in k different knapsacks, we use k compartments for this set. Since the algorithm makes at most $N - 1$ splits, the total number of compartments used by this algorithm is $M + N - 1$. □

3.2. *Uniform Capacity Ratio.* In this section we present an alternative condition for the existence of perfect placement. We require that the volume and the number of compartments in a knapsack be correlated—knapsacks with high volume should have many compartments and vice versa.

DEFINITION 3.1. For any $1 \leq j \leq N$, the *capacity ratio* of K_j is V_j/C_j. The set of knapsacks $K_1, \ldots, K_N$ has a *uniform capacity ratio* if there exists a constant $r > 0$, such that $V_j/C_j = r, \forall 1 \leq j \leq N$.

Intuitively, the capacity ratio of a knapsack K_j gives the average number of items contained in each compartment when the volume of K_j is totally utilized. We show that, for a set of knapsacks with uniform capacity ratio, if $\sum_{j=1}^{N} C_j \geq M + N - 1$, then a perfect placement exists and can be found efficiently. This holds for *any* distribution on the sizes of the sets $U_1, \ldots, U_M$.

THEOREM 3.2. *If the capacity ratio is uniform and $\sum_{j=1}^{N} C_j \geq M + N - 1$, then a perfect placement exists and can be found in $O(M \cdot N + \max(N \lg N, M \lg M))$ steps.*

PROOF. We present a polynomial time algorithm which yields a perfect placement. The algorithm, denoted by $\mathcal{A}_u$, proceeds by placing exactly V_j items of at most C_j different sets into each of the knapsacks. We assume that the sets are given in a nondecreasing order of their sizes, i.e., $|U_1| \leq |U_2| \leq \cdots \leq |U_M|$.

In each step we keep the remainders of the sets in a sorted list, denoted by R. The list, $R[1], \ldots, R[m]$, $1 \leq m \leq M$, is updated during the algorithm, that is, we remove from R sets that were fully packed, and we move to their updated place sets that were only partially packed. R is the *volume request list*, that is, each entry of R represents the request for volume, which is the number of unpacked items, of some set U_i. The knapsacks are given in a nondecreasing order of the number of compartments they have, i.e., $C_1 \leq C_2 \leq \cdots \leq C_N$.

The main idea in the algorithm is to keep the average size of the requests in R large enough. Specifically, for each $1 \leq k \leq N$, we will show that when the knapsack K_k is filled, either the average request size is at least r, or a trivial greedy algorithm can be applied to pack the remaining requests in the remaining knapsacks.

The algorithm uses two knapsack-filling procedures: some of the knapsacks are filled using the *greedy-filling* procedure; the other knapsacks will be filled using the *moving-window* procedure. We now describe the two procedures.

GREEDY-FILLING. The greedy-filling procedure fills a knapsack with items, starting from the smallest set, $R[1]$, and continuing until the knapsack is *saturated*, that is, until it contains exactly V_j items. The last set may split, that is, only part of its items will be packed into K_j. A formal description of the greedy-filling procedure is given in Figure 2.

MOVING-WINDOW. The moving-window procedure fills a knapsack, K_j, with the first sequence of C_j sets whose total size is at least V_j. We search the list R using a moving window of size C_j. Initially, the window covers the set of the smallest C_j sets. In every iteration we replace the smallest set in the window by the next set in R, until the number of items that are contained in the window is large enough to saturate K_j. If the subset of the C_j largest sets includes less than V_j items, then we cannot saturate K_j. However, we

Greedy-Filling(j)
$i \leftarrow 1$
repeat
    Allocate one compartment of K_j for items of the set U_k corresponding to $R[i]$.
    Place items of U_k in K_j: $Q_{k,j} = \min\{R[i], V_j\}$
    $V_j \leftarrow V_j - Q_{k,j}$
    $i \leftarrow i + 1$
until $V_j = 0$
Remove from R the sets that were fully packed into K_j.
Update the size of the new smallest set that was only partially packed into K_j.
Remove K_j from the knapsack list.

Fig. 2. The greedy-filling procedure.

Moving-Window(j)
$i \leftarrow 1$
$a = R[1] + \cdots + R[C_j]$.
while ($a < V_j$ and $|R| \geq C_j + i$)
    $a \leftarrow a - R[i] + R[C_j + i]$
    $i \leftarrow i + 1$
Fill K_j:
    Allocate compartments of K_j to all the sets corresponding to the requests $R[i], \ldots, R[i + C_j - 1]$.
    For each k, $i \leq k \leq i + C_j - 2$
        Place in K_j $R[k]$ elements of the corresponding set.
    $Q_{k',j} = V_j - (R[i] + \cdots + R[i + C_j - 2])$.
    Place in K_j $Q_{k',j}$ elements of the set $U_{k'}$ corresponding to $R[i + C_j - 1]$.
Update the list R:
    Remove from R the sets $R[i], \ldots, R[i + C_j - 2]$.
    If the set $U_{k'}$ corresponding to $R[i + C_j - 1]$ is not completely packed,
        Move $R[i + C_j - 1]$ to its appropriate new place in the sorted list R.
Remove K_j from the knapsack list.

Fig. 3. The moving-window procedure.

show below that this never happens. A formal description of the moving-window procedure is given in Figure 3. Note that the window advances until it covers C_j sets, such that $V_j > R[i-1] + \cdots + R[i + C_j - 2]$ and $V_j \leq R[i] + \cdots + R[i + C_j - 1]$ (see Figure 4). At this stage we can clearly place in K_j all the items in the sets $R[i], \ldots, R[i + C_j - 2]$, and saturate K_j by adding some of the items in the set $R[i + C_j - 1]$.

THE ALGORITHM $\mathcal{A}_u$. The algorithm proceeds in iterations. In the jth iteration we fill K_j by the following rules:

1. If there are less than C_j sets, or if the subset of the C_j smallest sets contains more than V_j items, then fill K_j using the greedy procedure.
2. If the subset of the C_j smallest sets contains at most V_j items, then fill K_j using the moving-window procedure.

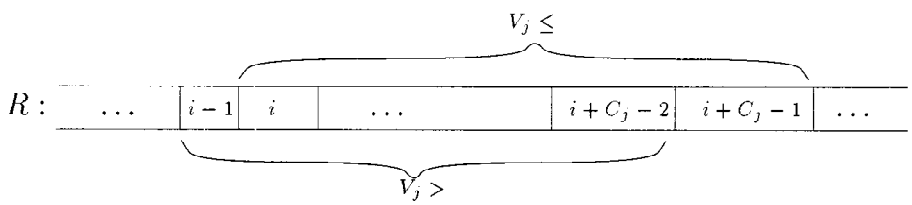

Fig. 4. The list R.

We now show that the algorithm terminates with a perfect placement. We distinguish between two stages in the execution of $\mathcal{A}_u$:

1. In the first stage, each knapsack K_j that is filled contains $C_j - 1$ or C_j sets of items. This stage includes executions of the moving-window procedure and some executions of the greedy procedure.
2. The second stage starts when, for the first time, the number of sets placed in K_j, for some $1 \leq j \leq N$, is smaller than C_j. This can clearly happen only when the greedy procedure is used.

Note that for some inputs, one of the stages may not occur. For each of the two stages, we show that any knapsack K_j that is filled during this stage will contain exactly V_j items of at most C_j different sets.

We use the following notation:

- C^k is the total number of compartments that are available after k knapsacks are filled.
- M^k is the number of sets that are not fully packed after k knapsacks are filled.
- N^k is the number of empty knapsacks after k knapsacks are filled ($N^k = N - k$).
- r is the capacity ratio, that is, $\forall j,\ V_j/C_j = r$.

We now consider the first stage.

LEMMA 3.3. *Each of the knapsacks that are filled during the first stage of $\mathcal{A}_u$ will contain exactly V_j items that belong to at most C_j different sets.*

PROOF. We show the following invariant for the ratio between the number of remaining sets, knapsacks, and compartments, during the first stage. Let K_b be the first knapsack filled in the second stage: b can be any number between 1 and N.

CLAIM 3.4. *For every $0 \leq k < b$, $C^k \geq M^k + N^k - 1$.*

PROOF. The proof is by induction on k, the number of knapsacks that were already filled.

Base. $k = 0$. It is given that $\sum_{j=1}^{N} C_j \geq M + N - 1$. Thus, using our notation, $C^0 \geq M^0 + N^0 - 1$.

Induction Step. Consider the kth iteration, in which K_k is filled.

1. After this iteration K_k is no longer available, therefore $C^k = C^{k-1} - C_k$.
2. By the definition of the first stage, we fully pack in this iteration $C_k - 1$ or C_k sets. Therefore $M^k = M^{k-1} - (C_k - 1)$ or $M^k = M^{k-1} - C_k$. Hence, $M^k \leq M^{k-1} - (C_k - 1)$, that is, $M^{k-1} \geq M^k + C_k - 1$
3. Clearly, $N^k = N^{k-1} - 1$.

By the induction hypothesis, $C^{k-1} \geq M^{k-1} + N^{k-1} - 1$, therefore

$$\begin{aligned} C^k &= C^{k-1} - C_k \\ &\geq M^{k-1} + N^{k-1} - 1 - C_k \\ &\geq M^k + C_k - 1 + N^k + 1 - 1 - C_k \\ &= M^k + N^k - 1. \end{aligned}$$

□

CLAIM 3.5. *For every $1 \leq k < b$, the average set size when we fill K_k is at least r.*

PROOF. Using Claim 3.4, after $k-1$ knapsacks are filled, the average set size is

$$\frac{\text{number of remaining items}}{\text{number of remaining sets}} \geq \frac{\text{remaining packing potential}}{\text{number of remaining sets}}$$

$$= \frac{r \cdot C^{k-1}}{M^{k-1}} \geq \frac{r \cdot (M^{k-1} + N^{k-1} - 1)}{M^{k-1}} \geq r.$$

Recall that at the beginning $\sum_{i=1}^{M} |U_i| = \sum_{j=1}^{N} V_j$. Therefore, at any stage, the number of nonpacked items is at least the remaining packing potential. Also, $N^{k-1} \geq 1$ since at least K_k is still empty after $k-1$ knapsacks are filled. □

We conclude that, for every $k < b$, the largest C_k sets after $(k-1)$ iterations contain at least $r \cdot C_k$ items, which is equal to V_k. This means that if the moving-window procedure is applied, the window never reaches the end of R (the largest sets) without saturating K_k. Knapsacks that are filled by the greedy procedure are clearly saturated, since the greedy procedure is applied when even the smallest C_k sets are large enough to saturate K_k. □

We now analyze the second stage of the algorithm. Note that we reach the second stage when the smallest $C_b - 1$ sets together contain more than V_b items.

LEMMA 3.6. *Each of the knapsacks that are filled during the second stage of $\mathcal{A}_u$ will contain exactly V_j items that belong to at most C_j different sets.*

PROOF. Let R_b be the set list at the end of the first stage. We first show that, for every $j \geq b$, K_j can be saturated by any subset of $C_j - 1$ sets from R_b.

CLAIM 3.7. *At the beginning of the second stage, for every $j \geq b$, any subset of $C_j - 1$ sets contains more than V_j items.*

PROOF. By definition, K_b is the first knapsack filled in the second stage. Therefore less than $C_b - 1$ sets are packed into K_b. This can happen only if the subset of the $C_b - 1$ smallest sets contains more than $V_b = r \cdot C_b$ items. Therefore the average size of the smallest $C_b - 1$ sets is larger than $(r \cdot C_b)/(C_b - 1)$. Consider a knapsack K_j, $j > b$. The knapsacks are sorted such that $C_j \geq C_b$, and the sets are sorted in a nondecreasing

order of their sizes, therefore the average size of the smallest $C_j - 1$ sets is larger than $(r \cdot C_b)/(C_b - 1)$. In addition, since $C_j \geq C_b$,

$$\frac{r \cdot C_b}{C_b - 1} \geq \frac{r \cdot C_j}{C_j - 1}.$$

This means that the total number of items in the smallest $C_j - 1$ sets is larger than

$$(C_j - 1) \cdot \frac{r \cdot C_j}{C_j - 1} = r \cdot C_j = V_j.$$

Clearly, if the smallest $C_j - 1$ sets in R_b contain more than V_j items, then any $C_j - 1$ sets in R_b contain more than V_j items. □

We now use Claim 3.7 to show that any knapsack filled after K_b contains at most $C_j - 1$ different sets, and is saturated by additional items packed from one set of R_b.

By the greedy-filling procedure, K_b is filled until V_b items are packed. The last set packed into K_b may include some more unpacked items. The fraction left from the split set is now the smallest set in R_b (since even before the split it was part of the smallest available set). In other words, the fraction is $R[1]$. We now turn to fill K_{b+1}. Consider the smallest C_{b+1} sets. These sets are composed of $R[1]$ and additional $C_{b+1} - 1$ sets. By Claim 3.7 the additional $C_{b+1} - 1$ sets include more than V_{b+1} items, therefore (no matter what the size of the fraction $R[1]$) the smallest C_{b+1} sets include together more than V_{b+1} items, and the greedy-filling procedure is used: V_{b+1} items of at most C_{b+1} sets are placed in K_{b+1}. Again, the last set may be only partially packed. The same argument holds until we reach a knapsack K_l, such that less than C_l sets are left. Since $\sum_{i=1}^{M} |U_i| = \sum_{j=1}^{N} V_j$ at the beginning, and since all the knapsacks before K_l were saturated, the total number of nonpacked items equals the total left volume. Hence, we can fill K_l using the greedy procedure. This argument holds for all the remaining knapsacks, until K_N is saturated and no sets are left. □

Combining Lemmas 3.3 and 3.6 we conclude that each of the knapsacks K_j, $1 \leq j \leq N$, is filled by exactly V_j items of at most C_j different sets. Thus, the algorithm terminates with a perfect placement.

The algorithm is polynomial: each of the filling procedures takes $O(M)$ steps. Adding the preprocessing complexity of sorting the lists, the total complexity of the algorithm is $O(N \cdot M + N \log N + M \log M)$. □

3.3. *Approximating a Perfect Placement.* When the knapsacks do not have uniform capacity ratio, the degree of nonuniformity is measured by the minimal α such that, for some $r > 0$,

$$\forall j, \qquad r \leq \frac{V_j}{C_j} \leq \alpha \cdot r. \tag{1}$$

(When $\alpha = 1$ we have uniform capacity ratio, in any other case $\alpha > 1$.) In this section we show how a perfect placement can be approximated. The approximation ratio is proportional to α.

THEOREM 3.8. *Let $\alpha \geq 1$ be the minimal number satisfying* (1). *If $C \geq M + N - 1$, then a $(1/\alpha)$-utilized placement can be found in polynomial time.*

PROOF. Consider the following instance, I', of the placement problem: for each knapsack K_j, $1 \leq j \leq N$, the volume of K_j in I' is $V'_j = \lceil r \cdot C_j \rceil$. Let $V' = \sum_{j=1}^{N} V'_j$. For each set, U_i, the set size of U_i in I', denoted by $|U'_i|$, is determined by solving the following MAX-MIN problem:

$$(2) \qquad \text{Maximize} \quad \min \frac{|U'_i|}{|U_i|}, \quad \text{such that} \quad \sum_{i=1}^{M} |U'_i| = V', \qquad |U'_i| \text{ integer.}$$

First, note that any legal placement of I' induces a legal placement for the original instance. This follows from the next claim:

CLAIM 3.9. *For each $1 \leq j \leq N$, $V'_j \leq V_j$.*

PROOF. For each $1 \leq j \leq N$, V_j is an integer. Therefore, for the knapsack achieving the minimal capacity ratio, $V_j = r \cdot C_j = \lceil r \cdot C_j \rceil = V'_j$. For each knapsack K_j such that $V_j/C_j > r$, there exists some $\varepsilon > 0$ such that $V_j = (r + \varepsilon) \cdot C_j = \lceil (r + \varepsilon) \cdot C_j \rceil \geq \lceil r \cdot C_j \rceil = V'_j$. □

Next, we show that a perfect placement exists, and can be found efficiently for I'. Note that the knapsacks in I' do not necessarily have uniform capacity ratio, however, the nonuniformity is small enough to show that the algorithm $\mathcal{A}_u$ presented in Section 3.2 is suitable for I'. We follow the proof of Theorem 3.2 to show that $\mathcal{A}_u$ fills each knapsack with exactly V'_j items of at most C_j sets. It is easy to verify that Claims 3.4 and 3.5 hold for I'. Let K_k be a knapsack filled during the first stage of $\mathcal{A}_u$. By Claim 3.5, the average set size when we fill K_k is at least r. Indeed, V'_j may be larger than $r \cdot C_j$, however, since the number of items included in the window is an integer and since $V'_j = \lceil r \cdot C_j \rceil$ is the smallest integer not smaller than $r \cdot C_j$, we conclude that whenever the moving-window procedure is applied, the window does not reach the end of R without saturating K_k. Similarly, for the second stage of $\mathcal{A}_u$, Claim 3.7 holds for I' and we conclude that all the knapsacks are saturated. By Claim 3.9, a legal placement in I' is a legal placement for the original instance. Since $V' \geq (1/\alpha)V$, a perfect placement in I' induces a placement that is $(1/\alpha)$-utilized for the original instance I. □

REMARK 3.1. For solving the CMKP, any choice of set sizes that satisfies, $\forall i$, $|U'_i| \leq |U_i|$ and $\sum_{i=1}^{M} |U'_i| = V'$ can be applied here. The solution of (2) also solves efficiently the FPP: for each set U_i, $|U'_i| \geq (1/\alpha)|U_i| - 1$.

In order to achieve a good approximation we would like to have α as close as possible to 1. In Section 5 we show that if compartments can be moved among the knapsacks, then $\alpha = C_k/(C_k - 1)$, where K_k is the knapsack having the maximal capacity ratio. This model is relevant to MOD storage subsystems, in which storage resources can be moved among the disks.

4. Approximation Algorithm for the CMKP. In this section we present a dual approximation algorithm for the CMKP. Specifically, we show that it is sufficient to add one compartment to each knapsack, in order to eliminate the gap between the performance of an optimal, probably exponential time algorithm, and a polynomial time algorithm. Recall that a *dual approximation algorithm* finds an infeasible solution that is superoptimal. Its performance is measured by the degree of infeasibility allowed. The proposed algorithm is allowed to place items of $C_j + 1$ (instead of C_j) different sets into K_j, $1 \le j \le N$.

THEOREM 4.1. *Given an instance I of the CMKP, by adding a single compartment to each knapsack, we can find in polynomial time a placement, whose utilization is at least the maximal possible utilization for I.*

PROOF. Let I be an instance of the CMKP. Denote by I^+ the instance generated from I by adding one compartment to each knapsack. We present a polynomial time algorithm which finds a legal placement in I^+. The algorithm, denoted by $\mathcal{A}_r$, proceeds by filling K_j with at most V_j items of at most $C_j + 1$ different sets, $\forall 1 \le j \le N$. The total number of items packed from I^+ by $\mathcal{A}_r$ is at least the total number of items packed from I by an optimal algorithm.

As in the uniform-ratio case (Section 3.2), we assume that the sets of items are given in nondecreasing order of their sizes, i.e., the sets satisfy $|U_1| \le |U_2| \le \cdots \le |U_M|$. We keep the remaining sets in a sorted list, denoted by R, which is updated during the algorithm. This algorithm also uses the two knapsack-filling procedures: greedy-filling and moving-window introduced in Section 3.2. However, the moving-window procedure is slightly changed: now the window covers $C_j + 1$ sets (instead of C_j). Also, if the subset of the $C_j + 1$ largest sets includes less than V_j items, then we cannot saturate K_j, and we do the best we can: we fill K_j with these $C_j + 1$ sets (in the uniform-ratio case this never happens).

The knapsacks are given in a nonincreasing order by their *capacity ratio*, i.e., $V_1/C_1 \ge V_2/C_2 \ge \cdots \ge V_N/C_N$. The sorted knapsacks are kept in a list denoted by L'. Another list of knapsacks, denoted by L'', may be created during the execution of the algorithm.

THE ALGORITHM $\mathcal{A}_r$. Generally, the algorithm uses the moving-window procedure to fill the knapsacks according to their order in L', that is, in the jth iteration we fill K_j and remove it from L'.

Note that the moving-window procedure places in K_j items of exactly $C_j + 1$ sets. Sometimes it is not possible to pack from that amount of sets (see below): in such cases we move K_j to L'' or look for another knapsack for which the moving-window procedure can be applied. Knapsacks that are moved from L' to L'' are filled using the greedy procedure, after L' is empty.

The following rules are used when $\mathcal{A}_r$ examines $K_j \in L'$.

- If $K_j = \emptyset$, i.e., L' is empty, fill sequentially all the knapsacks in L'', using the greedy-filling procedure.
- If there are less than $C_j + 1$ sets in R, move K_j to the end of L''.

- If the subset of the C_j smallest sets in R contains at most V_j items, fill K_j with items from $C_j + 1$ different sets, using the moving-window procedure.
- If the subset of the C_j smallest sets in R contains more than V_j items, look for the first knapsack $K_k \in L'$ for which the subset of the smallest C_k sets includes at most V_k items. If such a knapsack exists, fill K_k using the moving-window procedure; K_j will be examined again in the next iteration. If there is no such knapsack in L', fill sequentially all the knapsacks in L' and L'' using the greedy-filling procedure.

Optimality. Denote by $G = \{K_{g_1}, K_{g_2}, \ldots, K_{g_n}\}$ the set of knapsacks that are not saturated by $\mathcal{A}_r$, and denote by $w_1, w_2, \ldots, w_n$ the resulting waste of volume in each non-saturated knapsack, i.e., the volume of K_{g_i} is not fully exploited and only $V_{g_i} - w_i$ items are placed in K_{g_i}. We show that there is no legal placement of I in which the total utilization exceeds $\sum_{j=1}^{N} V_j - (w_1 + w_2 + \cdots + w_n)$. To prove this, we distinguish between four stages of $\mathcal{A}_r$:

1. Knapsacks from L' are filled by their order in L', using the moving-window procedure; some of the knapsacks may be moved to L''.
2. Knapsacks from L' are filled using the moving-window procedure, but not necessarily according to their order in L'; some of the knapsacks may be moved to L''.
3. Knapsacks from L' are filled, using the greedy-filling procedure (when no knapsack from L' can be filled by the moving-window procedure).
4. Knapsacks from L'' are filled using the greedy-filling procedure (when L' is empty).

For the first two stages we show that the total waste of volume for I^+ is at most the total waste of volume in an optimal placement of I. For the last two stages we show that there is no waste, and all the knapsacks filled during these stages are saturated.

For simplicity, assume that whenever the moving-window procedure is executed, the list R is scanned from left to right, that is, the smallest set, $R[1]$, is the leftmost set and the largest set is the rightmost in R. During execution of the moving-window procedure, the window moves from left to right. There are two possible scenarios:

1. **Saturating:** in which K_j is saturated, meaning that there exists some i, which is the index in R of the smallest set in the window, such that
   (i) ($i > 1$ and $V_j > R[i-1] + \cdots + R[i + C_j - 1]$) or ($i = 1$ and $V_j > R[1] + \cdots + R[C_j]$);
   (ii) $V_j \leq R[i] + \cdots + R[i + C_j]$.
   At this stage, we can clearly place in K_j the C_j sets $R[i], \ldots, R[i + C_j - 1]$, and saturate K_j by adding some of the items of $R[i + C_j]$. The sets $R[i], \ldots, R[i + C_j - 1]$ are removed from R, and the fraction left from the set $R[i + C_j]$ is moved to its updated place in the list R. Note that since some of the items of this set are packed, the position of that fraction in R is left to its original position.
2. **Nonsaturating:** in which the window reaches the rightmost position in R, but the subset of the $C_j + 1$ largest sets covered by the window contains less than V_j items. All items in the sets $R[i], \ldots, R[i + C_j]$ are packed, and these sets are removed from R.

In both cases we can consider the removed sequence of sets as a *hole* in R. In a saturating execution, K_j creates a hole of C_j sets in R, and one additional set (the fraction left from the last set) is moved left of the hole. In a nonsaturating execution, K_j creates a hole of $C_j + 1$ sets in R.

We examine the sequence of holes created in R during the first stage of the algorithm. We first show that every nonsaturated knapsack creates, at the right end of R, a hole which is the union of all the holes in R.

CLAIM 4.2. *Every $K_{g_j} \in G$ filled during the first stage of $\mathcal{A}_r$ unites the holes existing in R into a single hole positioned at the right end of R.*

PROOF. The proof is by induction on j, the index of K_{g_j} in G.

Base. K_{g_1} is the first nonsaturated knapsack in the execution of $\mathcal{A}_r$. The knapsacks are sorted in nonincreasing order by their capacity ratios. Therefore, for every $k < g_1$, the average number of items from each set packed into K_k is larger than the average number of items from each set packed into K_{g_1}. In particular, the largest set in the hole created by K_k is larger than the smallest set in the hole created by K_{g_1}. Thus, the hole created by K_{g_1} starts left of the hole created by K_k. In addition, since K_{g_1} is not saturated, the hole it creates includes the largest $C_{g_1}+1$ available sets, and in particular the rightmost one. Therefore it unites all the holes that were created by previously filled knapsacks into one hole.

Induction Step. Let K_{g_j} be the jth nonsaturated knapsack filled during the first stage of the algorithm $\mathcal{A}_r$. By the induction hypothesis, $K_{g_{j-1}}$ unites all the holes created by $K_1, K_2, \ldots, K_{g_{j-1}}$. In other words, $K_{g_{j-1}}$ divides R into two parts: the hole at the right and the remaining sets at the left. The knapsacks are sorted in nonincreasing order by their capacity ratios. Therefore, as in the base case, for every $g_{j-1} < k < g_j$ the average number of items in each set placed in K_k is larger than the average number of items in each set packed into K_{g_j}, thus the hole created by K_{g_j} starts left of the hole created by K_k. Since it also includes the largest available set, it unites the holes created by $K_{g_{j-1}+1}, \ldots, K_{g_j-1}$ and the hole at the right created by $K_1, \ldots, K_{g_{j-1}}$ into one hole. □

The way the holes are created implies that $\mathcal{A}_r$ is optimal for the knapsacks filled during the first stage:

LEMMA 4.3. *Let K_j be the last knapsack filled by $\mathcal{A}_r$ during the first stage, then the total number of items placed in $K_1, \ldots, K_j$ is at least the number of items placed into $K_1, \ldots, K_j$ under an optimal algorithm for the instance I.*

PROOF. Let K_{g_j} be the last nonsaturated knapsack filled during the first stage. The knapsacks that are filled after K_{g_j} are saturated. Thus, it is sufficient to prove that the lemma holds for $K_1, \ldots, K_{g_j}$. By Claim 4.2, the hole created by K_{g_j} unites all the holes created by $K_1, \ldots, K_{g_j}$. For every $k \leq g_j$ the hole created by K_k consists of C_k or C_k+1 sets, therefore the combined hole consists of at least the largest $C_1 + C_2 + \cdots + C_{g_j}$ sets. The total size of the hole is the total size of undivided sets placed in $K_1, \ldots, K_{g_j}$, which is at most $V_1 + V_2 + \cdots + V_{g_j} - (w_1 + w_2 + \cdots + w_j)$. We conclude that the sum of

the largest $C_1 + C_2 + \cdots + C_{g_j}$ sets is at most $V_1 + V_2 + \cdots + V_{g_j} - (w_1 + w_2 + \cdots + w_j)$, meaning that no algorithm, and in particular an optimal one, can place more than $V_1 + V_2 + \cdots + V_{g_j} - (w_1 + w_2 + \cdots + w_j)$ items in $K_1, \ldots, K_{g_j}$.

Note that fractions of sets that were created by $K_1, \ldots, K_{g_j-1}$ should not bother us: if a fraction of a set is packed later, it means that the original size of that set is contained in the united hole. If the fraction is not packed, then the total number of items packed by $\mathcal{A}_r$ is in fact larger than $V_1 + V_2 + \cdots + V_{g_j} - (w_1 + w_2 + \cdots + w_j)$, meaning that $K_1, \ldots, K_{g_j}$ cannot be filled better even by more than the largest $C_1 + C_2 + \cdots + C_{g_j}$ sets. □

Recall that during the first stage knapsacks can be moved from L' to L''. These knapsacks will be filled during the fourth stage, and the first stage continues. The first stage continues until we find a knapsack K_k, for which the subset of the C_k smallest sets contains more than V_k items.

We now prove the optimality of the second stage. The proof is similar to the proof for the first stage. We show that the holes created during the second stage are united whenever a knapsack is not saturated; then we conclude that the placement is optimal.

We first prove that the holes created during the second stage always "spread to the left" in R.

CLAIM 4.4. *Let K_{k_1} and K_{k_2} be two knapsacks that are filled in successive iterations during the second stage, then the hole created by K_{k_2} starts left of the hole created by K_{k_1}.*

PROOF. We consider separately two cases:

1. $k_1 < k_2$, meaning that the capacity ratio of K_{k_1} is higher than the capacity ratio of K_{k_2}. In this case, as in the first stage, it is clear that the hole created by K_{k_2} starts left of the hole created by K_{k_1}.
2. $k_1 > k_2$. By the algorithm, when we examined K_{k_2} the subset of the C_{k_2} smallest sets contained more than V_{k_2} items, and the filling of K_{k_2} was delayed. Since we finally fill K_{k_2} during the second stage, new small sets are used. These small sets are fractions created by knapsacks saturated after K_{k_2} was rejected. Since we examine K_{k_2} again after each iteration, and it is finally filled right after K_{k_1}, it means that a fraction created by K_{k_1} is used. Recall that, for $i \geq 1$, any remainder of a set which returns to R at the end of iteration i, will be positioned left of the hole created in R during this iteration. Thus the hole created by K_{k_2} starts left of the hole created by K_{k_1}. □

We now conclude that nonsaturated knapsacks unite the holes in R.

CLAIM 4.5. *Every $K_{g_j} \in G$ filled during the second stage of $\mathcal{A}_r$ unites the holes in R into a single hole; this hole forms the right end of R.*

PROOF. The proof is by induction on j, the index of K_{g_j} in G. We follow the steps of the proof of Claim 4.2. Indeed, we cannot assume that the knapsacks are filled in a nonincreasing order of their capacity ratio. However, by Claim 4.4, if K_{g_j} is filled after K_k, then the hole it creates starts left of the hole created by K_k, and since it also

includes the rightmost available set, it unites all the holes created by $K_k, \ldots, K_{g_j}$, and the induction in the proof of Claim 4.2 can be applied. □

As in the proof of optimality for the first stage, we conclude from the way the holes are created, that the placement is optimal.

COROLLARY 4.6. *Let K_j be the last knapsack filled by $\mathcal{A}_r$ during the second stage, then the total number of items packed into $K_1, \ldots, K_j$ is at least the number of items packed into $K_1, \ldots, K_j$ under an optimal algorithm for the original instance I.*

For the first and second stages we have shown that the total volume wasted in knapsacks filled during these stages does not exceed the waste of volume in these knapsacks under an optimal placement for I. We complete the proof of optimality by showing that all the knapsacks which are filled during the third and the fourth stages of $\mathcal{A}_r$ are saturated.

LEMMA 4.7. *Each of the knapsacks that are filled during the third stage of $\mathcal{A}_r$ will contain exactly V_j items, which belong to at most $C_j + 1$ different sets.*

PROOF. The third stage consists of successive executions of the greedy-filling procedure. $\mathcal{A}_r$ reaches the third stage if, for every knapsack $K_j \in L'$, the subset of the C_j smallest sets contains more than V_j items. Clearly, if the C_j smallest sets are too large, then any C_j sets are too large for K_j. We show that the greedy algorithm never uses more than $C_j + 1$ sets to fill K_j:

Let K_c be the first knapsack filled in the third stage. Clearly, at most C_c sets are used. The last set from which items are packed into K_c may split. The fraction that is left is now the smallest set (since even before the split it was part of the smallest available set). In other words, the fraction is $R[1]$. We now turn to fill K_{c+1}. Consider the subset of the $C_{c+1} + 1$ smallest sets. It consists of $R[1]$ and additional C_{c+1} sets. The subset of the additional C_{c+1} sets includes more than V_{c+1} items, therefore, no matter what the size of the fraction $R[1]$, the smallest $C_{c+1} + 1$ sets include together more than V_{c+1} items, and we can pack into K_{c+1} exactly V_{c+1} items from at most $C_{c+1} + 1$ sets. Again, the last set may split. The same argument holds for all the knapsacks that remain in L'. That is, every knapsack K_j will contain exactly V_j items that belong to at most $C_j + 1$ sets. □

LEMMA 4.8. *Each of the knapsacks filled during the fourth stage of $\mathcal{A}_r$ will contain exactly V_j items that belong to at most $C_j + 1$ different sets.*

PROOF. In the fourth stage we fill the knapsacks in L''. Recall that a knapsack K_j is moved to L'' if there are less than $C_j + 1$ available sets at the time it is examined by $\mathcal{A}_r$. Since we do not add sets along the execution of the algorithm, clearly there are less than $C_j + 1$ available sets when we fill K_j in the fourth stage, using the greedy procedure. In order to realize that the knapsacks are saturated, note that at the beginning $\sum_{i=1}^{M} |U_i| = \sum_{j=1}^{N} V_j$, that is, the total number of items is equal

to the total available volume. Since no knapsack is filled by more than V_j items, the sum of the sizes of the remaining sets always exceeds the total remaining volume. □

By combining Corollary 4.6 with Lemmas 4.7 and 4.8 we conclude that the total amount of wasted volume for the instance I^+ does not exceed the total amount of wasted volume under an optimal placement of I. In particular, if there exists a perfect placement of I, then our algorithm finds a perfect placement of I^+.

The algorithm is polynomial: each filling procedure has complexity $O(M)$. In the worst case (during the second stage) it takes $O(M \cdot N)$ steps to choose the next knapsack to be filled. Therefore the total complexity of $\mathcal{A}_r$ is $O(M^2 \cdot N^2)$. □

The dual approximation scheme yields the following approximation algorithm for the CMKP:

1. Find an optimal solution, assuming $C_j = C_j + 1, \forall 1 \leq j \leq N$. Let I', Q' be the resulting indicator and quantity matrices.
2. Let $I = I'$, $Q = Q'$.
3. For each knapsack K_j, $j = 1, \ldots, N$: if $\sum_i Q_{i,j} = C_j + 1$ (i.e., $C_j + 1$ compartments are used), let U_s be the set from which the quantity placed in K_j is minimal, that is, $Q_{s,j} = \min_{1 \leq i \leq M} Q_{i,j} > 0$, then $Q_{s,j} = 0$.

In other words, we turn the infeasible placement into a feasible one, by using only the C_j fullest compartments in each knapsack. Note that this way we omit from the subset of packed items at most $V_j/(C_j + 1)$ items.

COROLLARY 4.9. *Given an instance I of the CMKP, let U_{opt} be the utilization obtained by an optimal placement of the items, then we can find in $O(M^2 \cdot N^2)$ steps a placement which achieves utilization $U = U_{\text{opt}} - \varepsilon$ for $\varepsilon = \sum_{j=1}^{N}(V_j/(C_j + 1))$.*

In particular, if the number of compartments in each knapsack is at least b, for some $b \geq 1$, that is, $C_j \geq b$ for all $1 \leq j \leq N$, then the above approximation algorithm achieves utilization $(1 - \alpha)U_{\text{opt}}$ for $\alpha = 1/(b + 1)$.

5. Application to MOD Systems. In this section we show how our results for the CMKP and the FPP apply to storage management in multimedia systems. Consider a system having N disks: the storage capacity of disk j is C_j, and its load capacity is L_j, $1 \leq j \leq N$. The database associated with the MOD system contains M video program files $\{f_1, \ldots, f_M\}$, with the corresponding popularities $\{p_1, \ldots, p_M\}$. The popularity parameter of f_i reflects the portion of the total load generated due to access requests to f_i. Knowing these popularities and the total load capacity of the system, we can determine the average load generated by each of the files.

As mentioned in Section 1.2, the problem of assigning files to disks can be formulated as an instance of our packing problems, with the disks represented by knapsacks, and the files by sets of items. The popularities of the files determine the set sizes, such that $\sum_{i=1}^{M} |U_i| = \sum_{j=1}^{N} L_j$. When our objective is to maximize utilization, we need to solve

the CMKP; when the goal is to maximize fairness, we need to solve the FPP. A solution for any of our two variants of the knapsack problems will induce a legal static assignment. In terms of the matrices I and Q:

- $I_{i,j} = 1$ iff a copy of the file i is stored on the disk j.
- $Q_{i,j} \in \{0, 1, \ldots, L_j\}$ is the total load that file i can create on the disk j.

Thus, our results in Section 3 yield efficient algorithms for finding a *perfect* assignment of files to the disks, in which the load capacity of the system is totally utilized, and the requests to each of the files can be satisfied.

5.1. *Approximating Uniform Capacity Ratio.* We now consider a slightly different model, in which the storage subsystem consists of N *disk arrays*, $D_1, \ldots, D_N$: D_j has a fixed load capacity, L_j, and, in addition, there is a limit, C, on the *total* number of storage units that can be allocated to the disk arrays. We would like to find an allocation of the storage units to the disk arrays. That is, for any $1 \leq j \leq N$ we need to determine C_j, the storage capacity of D_j, such that $\sum_{j=1}^{N} C_j = C$. This model reflects the situation in which several disk arrays are used for storing the files. The storage capacity of a disk array is the sum of the storage capacities of the individual disks. Thus, the storage capacity of a disk array (with a fixed load capacity) can vary, depending on the storage capacities of the disks composing the array. We show below that, in such a system, the overall storage capacity can be distributed among the disk arrays, so as to achieve a nearly uniform capacity ratio. This enables us to find an almost perfect assignment of the files to the disks.

Let $L = \sum_{j=1}^{N} L_j$ be the total load capacity of the system. Indeed, it is easy to determine the storage capacity of each disk array so as to obtain a "uniform capacity ratio": in particular, we can choose $C_j = C \cdot L_j / L$. This yields a uniform capacity ratio with $r = L/C$. However, since we require that each disk array holds an integral number of files, we need to round the C_j's in a way that minimizes the violation of uniformity. Formally, we need to solve the following integer programming problem:

$$\text{(3)}\quad \text{Minimize} \quad \alpha = \frac{\max_{1 \leq j \leq N}(L_j/C_j)}{\min_{1 \leq j \leq N}(L_j/C_j)} \quad \text{such that} \quad \sum_{j=1}^{N} C_j = C, \qquad C_j \text{ integer.}$$

This problem can be optimally solved in $O(N \log N + N \log C)$ steps, by using, e.g., the algorithm "SOLVE-FAIR" (see Chapter 6 of [14]). The solution provides an allocation of the storage units to the disks such that, for some $r > 0$,

$$\text{(4)}\qquad \forall j, \qquad r \leq \frac{L_j}{C_j} \leq \alpha \cdot r,$$

and α is minimized. By Theorem 3.8 we have:

COROLLARY 5.1. *If storage units can be distributed among the disks, a* $(1/\alpha)$*-utilized assignment can be found in* $O(N \cdot M + N \log N + N \log C)$ *steps, where* α *is the value of the optimal solution of* (3).

To observe that α is small (that is, close to 1), note that, in any optimal solution of (3), the capacity ratio of D_j satisfies

$$\left\lfloor \frac{L}{C} \right\rfloor \leq \frac{L_j}{C_j} \leq \left\lceil \frac{L}{C} \right\rceil, \qquad \forall 1 \leq j \leq N.$$

This immediately yields a bound of 2 on α. More accurately, $\alpha \leq \lceil L/C \rceil / \lfloor L/C \rfloor$.

5.2. *Achieving Almost Optimal Utilization.* In Section 4 we presented a dual approximation algorithm for the CMKP. In terms of MOD systems, it means that we can achieve the optimal utilization of a system by adding one storage unit to each disk. Indeed, for a given MOD system, such changes in configuration may be impossible, however, we can use the approximation algorithm derived from algorithm $\mathcal{A}_r$, and the result in Corollary 4.9. This implies that if each of the disks can store at least b files, for some $b \geq 1$, that is, $C_j \geq b$ for all $1 \leq j \leq N$, then the above approximation algorithm achieves utilization $(1 - \alpha)U_{\text{opt}}$, with $\alpha = 1/(b + 1)$.

6. Discussion. We have studied two variants of the knapsack problem, namely, the CMKP and the FPP. We have shown that both problems are NP-hard; for some instances an optimal polynomial time algorithm exists. We also proposed an approximation algorithm for the CMKP. Finally, we have shown how our results for the CMKP and the FPP can be used for efficient resource allocation in multimedia storage subsystems.

Our paper leaves open several interesting avenues for future work:

- For the special case where $M = 1$, both the CMKP and the FPP are easy to solve. In contrast, both problems are hard to solve, already for the case where $M = 2$, if each knapsack has a single compartment. It is interesting to investigate further how the ratio between M, the number of items classes, and the number of compartments in the knapsacks enables us to find an optimal solution for each of the problems efficiently. Along these lines, it may be possible to formulate weaker versions of the conditions given in Section 3.1.
- We presented a $(1 - \alpha)$-approximation algorithm for the CMKP, where α depends on the input, namely, the capacity ratio of the knapsacks. Can the CMKP (FPP) be approximated to within a factor $1 - \varepsilon$, for any $\varepsilon > 0$, using a (fully) polynomial approximation scheme?
- We considered the case where $s(u) = w(u)$ for any $u \in U$. A natural extension of both the CMKP and the FPP would allow items of different types to have different sizes and different weights.
- We have shown the application of the CMKP and the FPP to the problem of assigning files to disks in multimedia storage subsystems. An underlying assumption in the assignment problem was that the hardware configuration is fixed, and our goal is to make the best use of this configuration, in terms of utilization and fairness. In the dual problem of *system configuration* our objective is to achieve a certain quality of service, and we are allowed to change the hardware configuration. Specifically, given a set of files, we would like to determine the number of disks that need to be used for storing and broadcasting these files; the disks may be of several different types, where each

type is characterized by specific storage and load capacity, and a fixed cost. This gives rise to the following class-constrained version of the fractional bin-packing problem: suppose we have a set U of $|U| = M$ items; each item $u \in U$ has a size $s(u) \in Z^+$. We can pack the items in U (allowing items to split) in a collection of bins that may be of several different types. A bin of type j has volume V_j; it can hold $C_j \geq 1$ items and its cost is F_j. Our objective is to pack all the items in a set of bins at minimal cost.

- The CMKP and the FPP can be applied for the *static* assignment of files to the disks. Static assignment is only the first component of an MOD resource allocation scheme, in which the initial state of the system is defined. It is then followed by a *dynamic phase*, in which customer requests arriving to the system need to be serviced. During the dynamic phase, the popularities of the various files can change. Such changes are natural, e.g., when dealing with video data available on the world-wide-web sites. In response, the MOD system should support operations such as file deletions or replications, as well as reallocations of load. This introduces an on-line version of each of our packing problems, where the sizes of the sets that we would like to pack can change dynamically, and we need to update the placement accordingly. The transition from one placement to another should be done with the minimum number of reallocations of items to compartments.

Acknowledgments. We would like to thank the reviewers, for providing many helpful comments and suggestions.

Appendix. In this appendix we show the validity of our results for general instances of the CMKP and the FPP, in which the total number of items is not necessarily equal to the sum of volumes of the knapsacks, namely, $|U| \neq V$. Note that the definitions of utilization and fairness, as given in Sections 2.1 and 2.2, do not depend on the ratio between $|U|$ and V. This ratio, however, influences the definition of *perfect placement*. When $|U| \neq V$, the maximal possible utilization is $\min(|U|, V)$. When the objective is to maximize fairness, an optimal placement is $\min(1, V/|U|)$-fair.[4] For a general instance, a placement is *perfect* if it is optimal with respect to utilization as well as fairness. Formally,

DEFINITION A.1. A *perfect placement* is a placement that is $\min(1, V/|U|)$-fair, in which one of the following is satisfied: (i) all the items are packed, or (ii) all the knapsacks are full.

We now argue, that all the results presented in Section 3 hold for general inputs:

- If $V > |U|$, then add one *dummy set* of size $V - |U|$ to obtain an instance in which $V = |U|$. After placing the items in the knapsacks, omit the "dummy" items.

[4] Standard rounding techniques [14] can be applied here to determine the exact number of items to be packed from each type.

- If $V < |U|$, a perfect placement is $(V/|U|)$-fair. For each set i, determine $|U_i^*| = (V/|U|)|U_i|$ (rounded to integers, such that $\sum_i |U_i^*| = V$). The resulting instance U^* satisfies $|U^*| = V$.

All the algorithms presented in Section 3 can be applied to the above adjusted instances, to produce perfect placements for the original instances. Special tuning is needed in the proof of Theorem 3.1: when $V > |U|$, the added dummy set may not satisfy the condition $|U_i| \geq (\varepsilon \cdot |U|)/M$. To solve this potential problem, the algorithm has to consider the dummy set first: note that one (arbitrarily small) fraction of a set is allowed in each knapsack, hence, the dummy set can serve as the fraction placed in the first knapsack.

Our results in Section 4 also hold for general inputs, namely, the statements of Theorem 4.1 and Corollary 4.9 remain valid. In fact, the algorithm $\mathcal{A}_r$, used in the proof of Theorem 4.1, can be applied for any instance of the CMKP. It is sufficient to show the validity of Corollary 4.6 and Lemmas 4.3, 4.7, and 4.8. First note that only the proof of Lemma 4.8 assumes that $|U| = V$. We now show how the proof of this lemma can be modified to argue that all the knapsacks filled during the fourth stage are "saturated," also when $|U| \neq V$. We consider separately two cases:

(i) If $|U| > V$, then as in the case where $|U| = V$, the sum of the sizes of the remaining sets always exceeds the total remaining volume.
(ii) If $V > |U|$, then all the knapsacks filled during the fourth stage are saturated until R is empty, or until all the knapsacks are filled. If R is empty (no requests are left), it means that we packed all the elements of U, which is clearly optimal. If we run out of knapsacks, then, as in the case where $|U| \geq V$, it means that all the knapsacks filled in this stage are saturated. Thus, the only waste of volume is the inevitable $w = w_1 + w_2 + \cdots + w_n$ (from the two first stages).

In both cases the utilization achieved by $\mathcal{A}_r$ is at least the maximal possible utilization for the original instance, I.

References

[1] C. Aggarwal, J. Wolf, and P. S. Yu, On optimal piggyback merging policies for video-on-demand systems, in *Proceedings of Sigmetrics*, 1996, pp. 200–209.
[2] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju, Staggered striping in multimedia information systems, in *Proceedings of SIGMOD*, 1994, pp. 79–90.
[3] A.K. Chandra, D.S. Hirschberg and C.K. Wong, Approximate algorithms for some generalized knapsack problems, *Theoret. Comput. Sci.*, **3** (1976), 293–304.
[4] P. Chen, E. Lee, G. Gibson, R. Katz, and D. Patterson, RAID: high performance, reliable secondary storage, *ACM Comput. Surveys*, **26**(2) (1994), 145–185.
[5] T. Corman, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, The MIT Press, Cambridge, MA, 1990.
[6] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA, 1979.
[7] G.V. Gens and E.V. Levner, Computational complexity of approximation algorithms for combinatorial problems, in *Proceedings of the 8th International Symposium on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science, Vol. 74, Springer-Verlag, Berlin, 1979, pp. 292–300.

[8] L. Golubchik, J.C.S Lui, and R.R. Muntz, Adaptive piggybacking: a novel technique for data sharing in video-on-demand storage servers, *ACM Multimedia Systems J.*, **4**(3) (1996), 140–155.
[9] R.L. Graham, Bounds for certain multiprocessing anomalies, *Bell Systems Tech. J.*, **45** (1966), 1563–1581.
[10] R.L. Graham, Bounds on multiprocessing timing anomalies, *SIAM J. Appl. Math.*, **17** (1969), 263–269.
[11] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, *Ann. Discrete Math.*, **5** (1979), 287–326.
[12] D.S. Hochbaum, *Approximation Algorithms for NP-Hard Problems*, PWS, Boston, MA, 1995.
[13] D.S. Hochbaum and D.B. Shmoys, Using dual approximation algorithms for scheduling problems: practical and theoretical results, *J. Assoc. Comput. Mach.*, **34**(1) (1987), 144–162.
[14] T. Ibaraki and N. Katoh, *Resource Allocation Problems - Algorithmic Approaches*, The MIT Press, Cambridge, MA, 1988.
[15] O.H. Ibarra and C.E. Kim, Fast approximation for the knapsack and the sum of subset problems, *J. Assoc. Comput. Mach.*, **22** (1979), 463–488.
[16] M. Kamath, K. Ramamritham, and D. Towsley, Continuous media sharing in multimedia database systems, in *Proceedings of the Fourth International Conference on Database Systems for Advanced Applications*, Singapore, 1995, pp. 79–86.
[17] P.W.K. Lie, J.C.S. Lui, and L. Golubchik, Threshold-based dynamic replication in large-scale video-on-demand systems, in *Proceedings of the Eighth International Workshop on Research Issues in Database Engineering* (*RIDE*), Orlando, FL, February 1998, pp. 52–59.
[18] S. Martello and P. Toth, Algorithms for knapsack problems, *Ann. Discrete Math.*, **31** (1987), 213–258.
[19] D. Pisinger, Algorithms for Knapsack Problems, Ph.D. Thesis, Department of Computer Science, University of Copenhagen, February 1995.
[20] S.S. Skiena, *The Algorithm Design Manual*, Springer-Verlag, New York, 1998.
[21] J.L. Wolf, P.S. Yu, and H. Shachnai, Scheduling issues in video-on-demand systems, in *Multimedia Information Storage and Management* (Soon M. Chung, ed.), Kluwer, Dordrecht, 1996, pp. 183–207.
[22] J.L. Wolf, P.S. Yu, and H. Shachnai, Disk load balancing for video-on-demand systems, *ACM Multimedia Systems J.*, **5** (1997), 358–370.