Multiple-type, two-dimensional bin packing problems: Applications and algorithms

Bernard T. Han[a], George Diehr[b] and Jack S. Cook[c]

[a]*Department of Management and Systems, College of Business and Economics, Washington State University, Pullman, WA 99164-4726, USA*

[b]*Center for High Technology Management, College of Business Administration, California State University, San Marcos, CA 92096, USA*

[c]*Jones School of Business, 1 College Circle, State University of New York – Geneseo, Geneseo, NY 14454, USA*

In this paper we consider a class of bin selection and packing problems (BPP) in which potential bins are of various types, have two resource constraints, and the resource requirement for each object differs for each bin type. The problem is to select bins and assign the objects to bins so as to minimize the sum of bin costs while meeting the two resource constraints. This problem represents an extension of the classical two-dimensional BPP in which bins are homogeneous. Typical applications of this research include computer storage device selection with file assignment, robot selection with work station assignment, and computer processor selection with task assignment. Three solution algorithms have been developed and tested: a simple greedy heuristic, a method based on *simulated annealing* (SA) and an exact algorithm based on *Column Generation* with Branch and Bound (CG). An LP-based method for generating tight lower bounds was also developed (LB). Several hundred test problems based on computer storage device selection and file assignment were generated and solved. The heuristic solved problems up to 100 objects in less than a second; average solution value was within about 3% of the optimum. SA improved solutions to an average gap of less than 1% but a significant increase in computing time. LB produced average lower bounds within 3% of optimum within a few seconds. CG is practical for small to moderately-sized problems – possibly as many as 50 objects.

1. Introduction

In the classical two-dimensional bin packing problem, we are given a set of objects each of which has two resource requirements. The problem is to select the minimum number of bins which will support a packing of the objects without violating the resource constraints. Bins are homogeneous–each has the same two resource capacities. Brief literature review for BPP appears in section 2.

The problems we consider differ from the classical problems in two important ways. First, the potential bins are not homogeneous. Their resource capacities differ

and their costs differ. Second, the resource demands of an object on a bin of one type typically differ from its demands on a bin of another type. We refer to this problem as the *multi-type two-dimensional bin packing problem* (M2BP). Examples follow.

COMPUTER PROCESSOR SELECTION WITH JOB ASSIGNMENT

Consider a finite number of real-time computer jobs which are to be assigned to a group of processors. The jobs must all run simultaneously and, for efficiency (e.g. fast response time) all must be memory-resident at all times. Each processor has processing and memory resource constraints, and each job is characterized by its resource demands for CPU time and memory which, in general, are different for each type of processor. The problem is to select a minimum cost mix of processors and make a feasible job assignment.

ROBOT SELECTION WITH WORKSTATION ASSIGNMENT

Consider workplace layout for an automated manufacturing system in which a number of workstations is to be served by a number of different types of robots. Each robot is constrained by its available processing time and work envelope, and each workstation has a known processing time demand and space requirement for each type of robot [4]. The problem is to find a minimum cost mix of robots and assign each workstation to a single robot within the resource constraints [17, 44].

FILE PLACEMENT FOR A MULTI-DEVICE STORAGE SYSTEM

Consider a computer system in which several types of on-line storage devices are being considered (e.g. conventional magnetic disks, various optical devices, and various tape-based systems) for storage of a number of files. Each device is constrained by its I/O and storage capacities, and each file has a known I/O load (i.e. device time) and file size for each device type. The problem is to minimize total cost by selecting a mix of devices and assigning files to devices such that the files on each device do not violate either the space or I/O capacity of the device [32, 33].

Such problems are becoming more common as information system users ranging from executives and decision support staff to conventional transaction processing applications demand on-line access to increasingly large volumes of data [35, 42]. The common solution is to select a hierarchy of storage devices ranging from slow access tape devices for data which is used only infrequently to central memory for data which requires micro-second access. While the size and access characteristics of some files will dictate the appropriate device, in many cases the optimum choice of storage devices and appropriate assignment of files to devices is not obvious. While a moderately fast device (e.g. optical disk) may have the space to store a particular set of files, and be able to provide adequate response time for a subset of the files, in the aggregate the retrieval and update requirements may swamp the device. The alternative of moving all files to a faster magnetic disk may increase costs.

The "bins" in this problem are clearly heterogeneous–storage capacities of various devices range from hundreds of megabytes for conventional magnetic disks to terabytes for devices such as optical jukeboxes. The performance capacity–e.g. random access time–also varies widely across these devices: magnetic disks have access times of milliseconds and can support files used in intensive real-time applications; optical jukeboxes, on the other hand, require several seconds to load the selected disk and access a record. Thus, the resource requirements of a given file are clearly not the same for each device type. Even the storage space required by a file is not constant across device types due to the different file structures which different devices dictate.

Our research objective was to develop computer algorithms which can solve M2BP problems of a practical size with acceptable computation times and provide optimum solutions or near-optimum solutions with sharp lower bounds. We have developed three solution algorithms and a lower-bounding scheme. One solution algorithm is a simple greedy heuristic which uses an opportunity cost concept to select new bins and assign objects to bins. This algorithm is also used as a front-end to the other methods. A second approximation method uses a global search technique based on *simulated annealing*. The third solution method is based on *column generation* [26–28] combined with branch-and-bound. The lower bounding (LB) scheme relies on a feasible solution from the heuristic to generate a set of potential bin configurations, then assigns objects to bins using linear programming. The result is a lower bound. The algorithms were evaluated by solving randomly generated problems which are based on realistic applications from the storage device selection-file assignment domain.

The remainder of the paper is organized as follows. Section 2 is a literature review. Section 3 presents mathematical formulations of the problem required for the CG method. Section 4 describes the algorithms. Computational results are presented in section 5 with summary and concluding remarks in section 6.

2. Review of literature

The conventional one-dimensional bin packing problem (BPP) is to find the minimum number of bins each of size 1 to pack a given collection of items with sizes in (0, 1] such that the sum of the sizes of all items packed into any given bin does not exceed 1 [36]. BPP is a well-known NP-hard problem [3]. In general, algorithms reported in the literature can be classified into two major types: those designed for *on-line* versus *off-line* applications. On-line algorithms are designed to pack a group of objects *as they are presented*, in contrast, off-line algorithms are given the full set of objects which are then packed *in any order*. Either type of algorithm has real world applications [7]. Recent reported research on off-line problem include Kampke [39] who used simulated annealing and Glover and Hubscher [30] who used tabu search.

An extension to the one-dimensional problem is the two-dimensional BPP which has also been extensively studied over the past two decades. The most comprehensive review of BPP appears in Coffman et al. [16]. One of the most recent reviews is by Dowsland and Dowsland [20].

For the two-dimensional BPP, two specific ways of packing are considered. The first way, called *box-packing*, is to treat the bin packing process as assigning rectangles (i.e. two-dimensional objects) into an open-ended bin which has a unit width and an infinite height. During the packing process, object rotation may or may not be allowed [6]. However, the common objective is to minimize the height of this open-ended bin. The second way, called *vector-packing*, is to treat each object and bin as a two-dimensional normalized vector, and the packing goal is to minimize the use of bins such that all objects packed into a bin have normalized component-wise sums ≤ 1. Our problem is of this second class. As discussed by Coffman et al. [16], each method of packing represents several realistic applications.

Many studies have been conducted on two-dimensional BPP with *homogeneous* bins. Most of these studies present approximation algorithms and their average or worst case performance analysis: Garey et al. [23], Yao [49], Baker et al. [5,6], Coffman et al. [15], Hofri [34], Golan [31], Fernandez de la Vega and Lueker [22], Brown et al. [11], Chung et al. [12], Karmarkar and Karp [40], Baker and Schwarz [7], Chazelle [13], Bartholdi et al. [9], Coppersmith and Raghavan [18], and Rhee and Talagrand [43].

None of these methods is directly applicable to the M2BP problems addressed here – i.e. BPP with two resource constraints, heterogeneous bins, and object resource requirements which depend on bin type. In the next section we formally define the problem.

3. Problem definition and formulation

A definition of the M2BP problem is:

> We are given N, 2-dimensional objects, each of which has "size" (p'_{ik}, q'_{ik}), where p'_{ik} and q'_{ik} represent the two-dimensional resource demands of an object i on a bin of type k. There are K different bin types, numbered 1 to K; a type k bin has a two-dimensional resource constraint (P_k, Q_k), and incurs a fixed charge, F_k, if selected. The problem is to select bins and pack all objects into these bins such that the total costs is minimized and the resource constraints are met.

Additional notation follows:

$\mathcal{I}$: the index set for the collection of objects, $\mathcal{I} = \{1, 2, 3, \ldots, N\}$.

$\mathcal{K}$: the index set for all bin types, $\mathcal{K} = \{1, 2, \ldots, K\}$.

$p_{ik} = p'_{ik}/P_k$: the normalized *dimension*-1 resource demand of object i on bin type k, $i \in \mathcal{I}$, $k \in \mathcal{K}$.

$q_{ik} = q'_{ik}/Q_k$: the normalized *dimension*-2 resource demand of object i on bin type k, $i \in \mathcal{I}$, $k \in \mathcal{K}$.

m_k : an upper bound on the number of type k bins in an optimal solution.

$\mathcal{J}_k$: the index set for each type k bin where $\mathcal{J}_k = \{1, 2, \ldots, m_k\}$, $k \in \mathcal{K}$.

$$y_{jk} : \begin{cases} 1 & \text{if the } j\text{th bin of type } k \text{ is used;} \\ 0 & \text{otherwise.} \end{cases}$$

$$x_{ijk} : \begin{cases} 1 & \text{if the object } i \text{ is packed into the } j\text{th bin of type } k; \\ 0 & \text{otherwise.} \end{cases}$$

The optimization problem is:

$$\textbf{(M2BP)} \qquad \text{minimize } Z = \sum_{k \in \mathcal{K}} F_k \left(\sum_{j \in \mathcal{J}_k} y_{jk} \right)$$

$$\text{subject to} \quad \sum_{k \in \mathcal{K}} \sum_{j \in \mathcal{J}_k} x_{ijk} = 1 \qquad \forall i \in \mathcal{I}, \tag{3.1}$$

$$\sum_{i \in \mathcal{I}} x_{ijk} p_{ik} \leq y_{jk} \qquad \forall j \in \mathcal{J}_k, \forall k \in \mathcal{K}, \tag{3.2}$$

$$\sum_{i \in \mathcal{I}} x_{ijk} q_{ik} \leq y_{jk} \qquad \forall j \in \mathcal{J}_k, \forall k \in \mathcal{K}, \tag{3.3}$$

$$x_{ijk}, y_{jk} \in \{0, 1\}, \qquad \forall i \in \mathcal{I}, \forall j \in \mathcal{J}_k, \forall k \in \mathcal{K}.$$

Constraint (3.1) ensures that each object is packed into exactly one bin. Constraints (3.2) and (3.3) ensure that bin capacities are not exceeded.

The formulation is a pure integer programming problem. Since it is a generalization of the 2-dimensional BPP with homogeneous bins, it is clearly NP-hard [5]. To our knowledge, no exact optimization algorithm has been developed for M2BP.

Potential exact algorithms for M2BP include methods based on Lagrangian relaxation [21, 25], branch and bound, and column generation [26–28]. Experience reported by other researchers on similar or simpler problems suggests that none of these methods would be acceptable for problems of 100+ objects [19]. Nevertheless, we needed an exact method to assist in validating the performance of our approximate methods and the lower-bounding method. CG was selected over other potential exact methods for the very practical reason that we had developed a CG-based method for a similar problem and its performance was at least acceptable [32]. In the next section we transform M2BP to a set covering problem which is necessary for the CG algorithm.

SET COVERING PROBLEM FORMULATION

Let Ç denote the column index set and $|Ç|$, its cardinality. M2BP can be written as a *set covering problem* (SCP) by using an $N \times |Ç|$ matrix **A**, in which elements of **A** are denoted by α_{ij}; $\alpha_{ij} = 1$ implies object i is packed in bin j. The following resource constraint conditions must hold for each column of **A**:

$$\sum_{i \in \mathcal{I}} p_{it_j} \alpha_{ij} \leq 1,$$

$$\sum_{i \in \mathcal{I}} q_{it_j} \alpha_{ij} \leq 1,$$

where t_j gives the bin type for column j. The cost of column j is F_{t_j}, the cost of the bin type used for the column.

The SCP formulation is:

$$\textbf{(SCP)} \qquad \underset{X}{\text{minimize}} \; Z = \sum_{j \in Ç} F_{t_j} x_j$$

$$\text{subject to} \qquad \mathbf{A}X = \mathbf{1},$$

where X is an $N \times 1$ column vector with elements $x_j \in \{0,1\}$, and $\mathbf{1}$ is an $N \times 1$ column vector of 1's.

Of course, if all possible feasible sets of objects (i.e. columns) are considered, the SCP formulation can lead to extremely large problems*. However, by relaxing the integrality constraint on the x_j and employing a column generation approach which implicitly considers all possible feasible columns, it is possible to obtain an optimum or a near-optimum solution with sharp lower bound.

4. Description of algorithms

This section describes the three algorithms: a simple greedy heuristic called *First Fit by Ordered Deviation* (FFOD), simulated annealing (SA), and column generation (CG).

4.1. GREEDY HEURISTIC ALGORITHM–FFOD

The FFOD heuristic serves two purposes: to generate a solution which is an end in itself or to provide an incumbent feasible solution whose value provides an upper bound, Z_{UB}, and a starting point for the SA or CG algorithm.

*A combinatorially large number of columns exist in even a moderate size problem. For example, consider a problem with $n = 40$. If, on the average, every bin has a capacity of 6, then there are $\binom{40}{1} + \binom{40}{2} + \ldots + \binom{40}{6}$ columns (> 4.5 million) in SCP formulation.

The FFOD sequentially considers objects and either assigns an object to an existing bin or opens a new bin and assigns the object to that bin. The assignment is based on minimum "opportunity cost", defined as follows:

$\mathcal{J}_j$: the index set for all objects stored in bin j;
t_j : the type of bin j;
$P_j = \sum_{i \in \mathcal{J}_j} p_{it_j}$: the total dimension-1 resource demand on bin j;
$Q_j = \sum_{i \in \mathcal{J}_j} q_{it_j}$: the total dimension-2 resource demand on bin j;
δ_{ij} : the adjusted resource demand if object i is assigned to bin j, i.e.,

$$\delta_{ij} : \begin{cases} \text{maximize}[p_{it_j}, q_{it_j} - (P_j - Q_j)] & \text{if } P_j > Q_j; \\ \text{maximize}[q_{it_j}, p_{it_j} - (Q_j - P_j)] & \text{if } Q_j \geq P_j. \end{cases}$$

The opportunity cost, c_{ij}, for assigning object i to bin j is the product of its adjusted resource demand, δ_{ij} and F_{t_j}, the fixed charge associated with the jth bin:

$$c_{ij} = F_{t_j} \delta_{ij}.$$

This definition of opportunity cost gives incentive to tightly pack objects into bins while also maintaining the balance of resource loads on each bin.

The algorithm, followed by explanatory comments, appears below. $\mathcal{J}$ is the index set of opened bins.

Step 1. Initialize the solution upper bound $Z_{UB} \leftarrow \infty$

Step 2. For $k' = 1$ to K

(2.1) Create vector $\boldsymbol{R}$ with elements $|p_{ik'} - q_{ik'}|/(p_{ik'} + q_{ik'})$ for $i = 1$ to N

(2.2) Sort elements of $\boldsymbol{R}$ into *nondecreasing* order

(2.3) Reindex objects from 1 to N based on the element sequence in array $\boldsymbol{R}$

(2.4) Find k^* such that $c_{1k^*} = \min_{k \in \mathcal{K}} c_{1k}$

$TC \leftarrow F_{k^*}$

$\mathcal{J} \leftarrow \{1\}$; $\mathcal{J}_1 = \{1\}$; $t_1 = k^*$

(2.5) For i = 2 to N

(2.5.1) $j^* \leftarrow 0$; Find j^* such that

$$c_{ij^*} = \min_{j \in \mathcal{J}} \{c_{ij} \,|\, (1 - P_j) \geq p_{it_j}, (1 - Q_j) \geq q_{it_j}\}$$

(2.5.2) Find k^* such that $c_{ik^*} = \min_{k \in \mathcal{K}} \{c_{ik} | \, 1 \geq p_{ik}, 1 \geq q_{ik}\}$

(2.5.3) If $j^* = 0$ or $c_{it_{j^*}} > c_{ik^*}$ then

$TC \leftarrow TC + F_{k^*}$

$j' = |\mathcal{J}| + 1$; $\mathcal{J} \leftarrow \mathcal{J} \cup \{j'\}$; $\mathcal{J}_{j'} \leftarrow \{i\}$; $t_{j'} = k^*$

else $\mathcal{J}_{j^*} \leftarrow \mathcal{J}_{j^*} \cup \{i\}$

(2.6) If $TC < Z_{UB}$ then $Z_{UB} \leftarrow TC$

Step 3. Stop.

Steps 2.1 and 2.2 determine the packing order by sorting objects into nondecreasing order of $|p_{ik} - q_{ik}|/(p_{ik} - q_{ik})$ (for given k), which reflects the balance between the two resource demands. Thus, objects with higher balance of resource demands are assigned first. To increase the chance of finding a good solution, the packing is repeated K times, once for each bin type (step 2). To further increase the probability of improving the solution value, the entire algorithm is repeated sorting objects into *nonincreasing* order.

Step 2.4 finds the "best" bin to open for the first object and assigns it to that bin.

Step 2.5.1 determines the cost of assigning object i to an open bin (if an open bin exists with sufficient capacity).

Step 2.5.2 determines the cost of assigning object i to a newly opened bin.

Step 2.5.3 determines whether it is cheaper to assign object i to an existing bin or open a new bin. If a new bin is opened, the total cost is updated, the index set $\mathcal{J}$ is expanded, and object i is assigned to that bin. If an existing bin is "cheaper", object i is assigned to the cheapest existing bin, j^*.

Step 2.6 saves the best feasible solution value and, although not shown, records the best solution.

The algorithm has a computational complexity $O(cN)$, where c is a constant bounded above by the number of potential bins.

4.2. LOWER-BOUNDING ALGORITHM AND PROBLEM SIZE REDUCTION–LB

There are several objectives for this algorithm. First, there is the obvious benefit of having a lower bound to solutions obtained by the approximate methods. In addition, the efficiency of the SA search is sensitive to the size of the solution space which is a function of the total number of bins of each type which must be considered. In developing the lower bound, the LB algorithm also finds tight upper bounds on the number of bins of each type.

Upper bounds on bins

Define U_k as the number of bins of each type k. Using the solution upper bound (Z_{UB}) found by the heuristic, *initial* upper bounds are given by:

$$U_k = \lfloor Z_{UB}/F_k \rfloor \quad \forall k \in \mathcal{K}.$$

The U_k are now used in a linear relaxation of the mathematical program for M2BP:

(M2BP/LP)

$$\text{minimize } Z = \sum_{k \in \mathcal{K}} \left(F_k \sum_{j \in \mathcal{J}_k} y_{jk} \right)$$

$$\text{subject to } \sum_{k \in \mathcal{K}} \sum_{j \in \mathcal{J}_k} x_{ijk} = 1 \qquad \forall i \in \mathcal{I}, \tag{4.1}$$

$$\sum_{i \in \mathcal{I}} x_{ijk} p_{ik} \leq y_{jk} \qquad \forall j \in \mathcal{J}_k, \forall k \in \mathcal{K}, \tag{4.2}$$

$$\sum_{i \in \mathcal{I}} x_{ijk} q_{ik} \leq y_{jk} \qquad \forall j \in \mathcal{J}_k, \forall k \in \mathcal{K}, \tag{4.3}$$

$$\sum_{j \in \mathcal{J}_k} y_{jk} \leq U_k \qquad \forall k \in \mathcal{K}, \tag{4.4}$$

$$\sum_{k \in \mathcal{K}} \sum_{j \in \mathcal{J}_k} F_k y_{jk} < Z_{UB}, \tag{4.5}$$

$$0 \leq x_{ijk}, y_{jk} \leq 1 \qquad \forall i \in \mathcal{I}, \forall j \in \mathcal{J}_k, \forall k \in \mathcal{K},$$

where $\mathcal{J}_k = \{1, 2, \ldots, U_k\}$, $\forall k \in \mathcal{K}$, is an index set for type k bins. Constraint (4.4) puts an upper limit on the number of bins of each type. The importance of constraint (4.5) will become clear in the following discussion.

To (potentially) reduce a U_k, the objective function is replaced by max $Z = \sum_{j \in \mathcal{J}_k} y_{jk}$. The resulting "optimum" Z is then truncated to an integer to redefine U_k; this new U_k is the maximum number of bins of type k in an optimum solution. The revised U_k replaces its previous value in (4.4). The process is repeated for each value of k. M2BP/LP is solved a final time using its original objective function, with all of the new U_k values, to obtain an *initial* lower bound, Z_{LB}.

Based on computational experience, this algorithm typically reduces the original problem size (i.e. $\sum_{k \in \mathcal{K}} U_k$) by 25% to 43%.

Lower bounding

After problem reduction, the following are available: (1) a feasible solution with upper bound, Z_{UB}; (2) a solution lower bound, Z_{LB}; and (3) upper limits, U_k, on the number of bins of each type. Using this information, the set of *potential bin configurations*—those configurations with cost less than Z_{UB}—is determined. Each potential configuration is then examined to determine if it is linearly feasible—that is, if it will allow (potentially fractional) assignments of objects to bins without violation of capacity constraints. The minimum cost of the feasible bin configurations is the new lower bound.

A *potential bin configuration* is any mix of bin types whose total cost lies between Z_{LB} and Z_{UB}. For example, suppose three bin types, A, B, and C are available with costs \$10, \$8, and \$6, respectively, and the current lower and upper bounds are \$20 and \$25, respectively. Then, one potential bin configuration is two

of type B and one of type C; another potential configuration is one bin of each type. However, one of type A and two of type B is not a potential bin configuration (cost exceeds upper bound) nor is one of type A and one of type B (cost falls below bound). Let the quadruplet (n_1, n_2, n_3, TC) denote a specific bin configuration with n_1 type 1 bins, n_2 type 2 bins, and n_3 type 3 bins, and it incurs a total cost TC. If we also assume that an upper limit of two bins of each type has been established, then the potential bin configurations are limited to the following five:

1. (0, 1, 2, \$20)
2. (2, 0, 0, \$20)
3. (0, 2, 1, \$22)
4. (1, 0, 2, \$22)
5. (1, 1, 1, \$24)

Typically, the number of potential bin configurations is less than 8% of the total possible configurations based on the U_k (i.e. $\prod_{k\in\mathcal{K}} (U_k+1)-1$).

The next step tests each configuration in order of increasing cost to determine if it allows a feasible solution to M2BP/feasible (see below). The lowest cost potential configuration which yields a feasible solution redefines Z_{LB}.

(M2BP/feasible)

$$\sum_{k\in\mathcal{K}} \sum_{j\in\mathcal{J}_k} x_{ijk} = 1 \qquad \forall i \in \mathcal{I},$$

$$\sum_{i\in\mathcal{I}} x_{ijk} p_{ik} \le 1 \qquad \forall j \in \mathcal{J}_k, \forall k \in \mathcal{K},$$

$$\sum_{i\in\mathcal{I}} x_{ijk} q_{ik} \le 1 \qquad \forall j \in \mathcal{J}_k, \forall k \in \mathcal{K},$$

$$x_{ijk} \le 1 \qquad \forall i \in \mathcal{I}, \forall j \in \mathcal{J}_k, \forall k \in \mathcal{K}.$$

This step typically reduces the difference in upper and lower bound to about half of its initial value. The increased lower bound also reduces search time in SA which uses a stopping rule based on the difference between solution value and lower bound. Of course, the lower bound can also be used to decide if the heuristic solution is acceptable without further search. Finally, LB occasionally produces an integer solution at value Z_{LB} which is, therefore, the optimum.

4.3. SIMULATED ANNEALING ALGORITHM–SA

Simulated annealing is a well-known global search technique for optimization [41]. The following elements must be specified for SA. They are detailed in subsequent subsections.

(1) Solution space, Ω, and solution state, S.

(2) A state generation mechanism and the neighborhood size, N_B.
(3) An objective (energy) function, $C(S)$.
(4) A cooling schedule, which includes:
  (a) a starting temperature, T_0;
  (b) a temperature decrement function, $g(T)$;
  (c) the final (frozen) temperature, T_f, and/or other stopping criteria.

Solution space (Ω) *and solution state* (S)

The solution space, Ω, is the set of all *possible* partitions of the N objects into less than or equal to $\sum_{k\in\mathcal{K}} U_k$ subsets and all possible assignments of bin types to each subset within a partition. The partitions of Ω are *not* constrained to those partitions which allow only feasible subsets of objects (i.e. subsets of objects which can be assigned to at least one of the available bin types without violating either resource constraint). A solution state, S, is any partition. A subset of S is denoted by s.

State generation mechanism and neighborhood structure

A *one-way transfer scheme* is used to generate a new solution state. Consider a solution state S. The one-way transfer generates a new state by moving an object from one bin to another or by moving an object to a new bin. All possible one-way transfers from a given state define the neighborhood structure. Note again that no feasibility checking is performed on the resultant packing.

This generation mechanism is adopted because of its simplicity and because it yields a tractable neighborhood size. In a problem with N objects and a maximum of B potential bins, the neighborhood size is $N_B = N(B-1)$.

Objective (energy) function

To incorporate the capacity constraints, two penalty terms are added to the objective function. The resulting energy function is:

$$\min_{S\in\Omega} C(S) = \min_{S\in\Omega} \sum_{s\in S} \left\{ F_{t(s)} + \alpha \max\left(1 - \sum_{i\in s} p_{it(s)},\ 1 - \sum_{i\in s} q_{it(s)}, 0\right) + \beta \max\left(\sum_{i\in s} p_{it(s)} - 1,\ \sum_{i\in s} q_{it(s)} - 1,\ 0\right)\right\},$$

where α and β are non-negative penalty parameters and $t(s)$ gives the bin type for subset s.

The term, $\alpha\max(\bullet)$, penalizes unused resources in each bin and the term $\beta\max(\bullet)$ penalizes capacity violations. Replacing the constraints by penalty terms

leads to a smoothing of the objective function "landscape", making it easier for SA to escape a local optimum. As reported by other researchers [37, 38], selection of weighting factors is empirical and very problem dependent. "Tuning" led to setting β to 0.30; the best choice for α seems to be 0.0 – that is, no penalty for underfilling a bin.

Cooling schedule

A cooling schedule includes: (1) the starting temperature, T_0; (2) a temperature decrement function $g(T)$; and (3) the final (i.e. frozen) temperature and/or some other stopping criterion. The selection of cooling schedule impacts the performance of SA. Considerable theoretical work has been done on cooling schedules [47, 48]. We adopt the cooling schedule by Aarts and van Laarhoven [2] for the following reasons:

(1) It uses a simple mechanism to derive the starting temperature, T_0.
(2) It uses fixed-length Markov chains (i.e. number of state transitions per temperature) to attain quasi-equilibrium at each temperature, which results in polynomial-time convergence.
(3) It employs a stochastic temperature decrement function, $g(T)$, which has proven effective in locating near-optimum solutions.

Details of the cooling schedule follow.

Starting temperature – T_0: The starting temperature is chosen to support a high acceptance of all possible state transitions. T_0 is determined by a search process. We begin with an arbitrary value, t, and carry out $2N_B$ state transitions which yield t^+ cost-increasing transitions and t^- cost-decreasing transitions. Let ΔC^+ represent the average cost increment over t^+ moves. The acceptance ratio, χ, is approximated by:

$$\chi \approx \frac{[t^+ e(-\Delta C^+/t) + t^-]}{t^+ + t^-}.$$

Experience suggests that the initial temperature should yield an acceptance ratio of about 0.90. Thus, if the computed c exceeds 0.90, T_0 is decreased; otherwise, T_0 is increased. A binary search is used to determine the desired starting temperature (see [1] for details).

Temperature decrement function – $g(T)$: The temperature is updated by:

$$g(T) = \frac{T}{1 + T\dfrac{\ln(1+\delta)}{(3\sigma_T)}},$$

where σ_T is the standard deviation of the objective function values over the number of state transitions generated at temperature T and δ is a *distance parameter* used to determine the cooling rate. In general, choosing small δ (around 1) leads to a small decrement in T and results in a better quality solution. Selecting large δ (around 10) leads to rapid cooling and, usually, freezes with a poorer final solution. We experimented with a range of δ values finding that values close to 1 produced unacceptably high solution times without significant improvement in solution quality. Values near 10 produced quick, but poor, solutions. We finally settled on a value of 5 which produced good solutions in acceptable time.

Final temperature and stopping criterion: Simulated annealing terminates at freezing temperature, T_f, defined by first occurrence of:

$$\frac{T_f}{\overline{C}(T_0)} \frac{|\overline{C}(T_f) - \overline{C}(T_{f-1})|}{T_{f-1} - T_f} < \varepsilon,$$

where $\overline{C}(T_0)$ is the average objective function value over the transitions generated at T_0; $\overline{C}(T_f)$ and $\overline{C}(T_{f-1})$ are moving averages of the objective function values over the last 30 state transitions at temperatures T_f and T_{f-1} respectively; ε, the stopping parameter, is set to 0.0005.

Unfortunately, the objective function is relatively bumpy making the stopping condition somewhat ineffective. Therefore, one more stopping criterion was added: SA terminates if the difference in upper and lower bounds, $(Z_{UB} - Z_{LB})/Z_{LB}$, is less than 5%.

Simulated annealing search

An outline of SA algorithm follows:

Step 1. Start with the best solution, S, with value, Z_{UB}, from the FFOD heuristic.
Step 2. Determine the initial temperature T_0 and cooling function $g(T)$.
Step 3. $T \leftarrow T_0$; $S^* \leftarrow S$.
Step 4. Do until the temperature is frozen *or* the stopping criterion is met
(4.1) Repeat N_B times
(1) Generate a random neighbor S' of S, and compute $\Delta = C(S') - C(S)$
(2) **If** $\Delta < 0$ **then** $S \leftarrow S'$
If S feasible **then** $S^* \leftarrow S$, $Z_{UB} \leftarrow C(S^*)$ **end if**
else
If Random ~ $[0, 1) \leq \exp(-\Delta/T)$ **then** $S \leftarrow S'$ **end if**
end if
(4.2) $T \leftarrow g(T)$.
Step 5. Output S^* and Z_{UB}.

4.4. COLUMN GENERATION ALGORITHM–CG

The exact algorithm uses the set covering formulation of section 3.2 with *column generation* followed by *branch-and-bound*. The algorithm is outlined below followed by details.

Step 1. *Column generation phase*

(1.1) Generate the initial set covering problem from the FFOD solution – the "Restricted Master Program", RMP. Solve RMP as a relaxed set-covering problem.

(1.2) Repeat until no column can be generated with non-negative reduced cost:

(1.2.1) For each bin type, use dual prices to generate a column – add best to RMP.

(1.2.2) Solve expanded RMP

Step 2. If solution is all integer, terminate; else, perform branch and bound.

Generate initial restricted master program

The initial columns (i.e. bins) are determined by feasible solution from the FFOD heuristic. A binary column, a_j, of type t_j, is created with cost coefficient F_{t_j} for $j = 1, 2, \ldots, |Ç|$.

Solve the relaxed RMP

The relaxed RMP is:

$$\textbf{(SCP)} \qquad \underset{X}{\text{minimize}}\ Z = \sum_{j \in Ç} F_{t_j} x_j$$

subject to *dual multiplier*

$$\sum_{j \in Ç} \alpha_{ij} x_j \geq 1, \qquad \pi_i \qquad \forall i \in \mathcal{I},$$

$$0 \leq x_j \leq 1 \qquad \forall j \in Ç.$$

Solution of the (relaxed) SCP yields two results. First, it may generate an integer solution which then becomes the new incumbent or may "suggest" a better incumbent (e.g. by rounding fractional results). Second, it produces dual multipliers which are used for column generation.

Generate "best" column

The dual multiplier of row i (i.e. π_i) represents the storage "price" of object i given the current set of bins. To determine if a column not in the RMP should

be added, its *reduced cost* is computed which is the difference between total prices of the objects represented by the potential column and the fixed charge of its bin type. The "best" potential columns – i.e. those with positive reduced cost – can be determined by solving a series of two-constraint knapsack problems as follows:

Let

Π_k^* : the maximum reduced cost over all feasible packings for a bin of type k.

$$\omega_i : \begin{cases} 1 & \text{if object } i \text{ is assigned to the bin;} \\ 0 & \text{otherwise.} \end{cases}$$

The best column (bin) is determined by solving the following problem, KP, for each bin type. The decision variables, $\omega_i, \forall i \in \mathcal{I}$, from the best solution define the column which will be added to the RMP.

$$\textbf{(KP)} \qquad \Pi_k^* = \text{maximize} \quad \sum_{i \in \mathcal{I}} \pi_i \omega_i - F_k$$

$$\text{subject to} \quad \sum_{i \in \mathcal{I}} p_{ik} \omega_i \leq 1,$$

$$\sum_{i \in \mathcal{I}} q_{ik} \omega_i \leq 1,$$

$$\omega_i \in \{0,1\} \qquad \forall i \in \mathcal{I}.$$

The fixed charge, F_k, is constant. Therefore, this problem is a multi-constraint knapsack problem which can be solved by an algorithm developed by Gavish and Pirkul [24].

If Π_k^* is non-positive, $\forall k \in \mathcal{K}$, then there are no potential added columns and we are done – the continuous solution to the RMP is a lower bound to the incumbent. If the solution is all-integer it is also the optimum; otherwise, a standard branch-and-bound algorithm (LINDO) is used to solve the problem to optimality.

5. Computational results and analysis

The algorithms were coded in FORTRAN using LINDO to solve LP subproblems The computer was an IBM 3090.

TEST PROBLEM GENERATION

Test problems were generated based on computer storage device (bin) selection and file (object) assignment problems described in the introduction to this paper and detailed by Han and Diehr [33]. The actual cost and parameter generation is rather involved; actual problem sets are available from the first author. General characteristics of test problems are outlined below.

(1) *Device characteristics*: Four storage types were considered: standard magnetic disk (MD), erasable optical disk (EOD), and write-once/read-many (WORM) optical disk with 30 and 60 day reorganization times. Amortized fixed costs per day (the F_k) are \$10, \$8, \$7, and \$6, for MD, EOD, WORM(30), and WORM(60), respectively.

Each device type has the same nominal storage capacity of 1. Performance capacities, however, differ across the four device types due to differences in random access times and transfer rates.

(2) *File characteristics*: Each file has two resource requirements, space and I/O, both characterized by the fraction of device space and I/O capacity the file requires. File space requirements, as a fraction of magnetic disk storage capacity, was randomly generated from a uniform distribution over the range 0 to S ($S < 1$). I/O requirement was randomly generated from an exponential distribution with mean μ (again, where μ is a fraction of total magnetic disk I/O capacity).

Resource requirements imposed by a given file on other, non-MD, device types are, in general, greater than imposed on the MD. For example, since the WORM does not support update in place, files stored on WORMs require additional space for updated records. The greater the update rate for a file, the greater the space which must be set aside on a WORM. Consider a file with size equal to 5% of MD capacity and update rate of 1% per day. If stored on a WORM with 60 days between reorganization, the file will grow by roughly 60%. Thus, the WORM space requirement will be 8% of total WORM capacity. For the MD and EOD only 5% of device capacity is required since both of these device types support update in place.

The I/O load imposed by a file also differs across devices. For example, random access time on an EOD is on the order of 50 milliseconds versus only 20 milliseconds on an MD. Furthermore, to update a record stored on an MD typically requires one disk read and one write. On the EOD, updates require at least one extra rotation due to separate erase and write operations. However, transfer rates on an EOD are comparable to the MD so that times for sequential file scans on the two devices are similar. Therefore, the I/O load on each device is a function of device characteristics and file I/O pattern (i.e. random versus sequential access and update rate).

(3) *Problem sets*: Problem sets were characterized by number of files, average file size (i.e. $S/2$), I/O load (value of μ), and update rate as follows:

(a) *Problem sizes* were 30, 40, 50, 75, and 100 files.

(b) Average *file sizes* and *average I/O loads* were 0.05, 0.1, and 0.2–approximately, an average of 20, 10, or 5 files per MD device (results in tables 1, 2, and 3, respectively.

(c) Daily file *update rates* were 1%, 2%, 3%, 4%, and 5%. As noted above, this parameter differentially impacts space requirements and I/O requirements across device types.

Of the 75 possible combinations of these parameters, problems for 55 combinations were generated with 10 problems for each set of parameters.

COMPUTATIONAL RESULTS

Tables 1–3 summarize results on 550 test problems for the four algorithms. Each table corresponds to a different bin capacity/object size ratio–for example, the files generated for table 2 have average (MD) file sizes equal to one-tenth of the capacity of an MD. Problem sets within each table are organized by number of objects (files) and within each size by file update rate. The column "*M* (bins)" gives the average number of potential bins which were considered by the FFOD heuristic. Data reported include CPU seconds on an IBM 3090 and "Gap". For the heuristic and SA, Gap is defined as the percentage difference in best solution value and the optimum if known. If the optimum is not known, the best solution is compared to the lower bound. For LB, the gap is compared to the optimum (if known) and to

Table 1

Solution quality and time: capacity/object size = 5.
(CPU time is IBM 3090 seconds)

Problem size		Update	FFOD algorithm		SA algorithm		LP bound		CG algorithm	
N (Objects)	*M* (Bins)	rate	CPU time	Gap (%)	CPU time	Gap (%)	CPU time	Gap (%)	CPU time	Range
30	8.2	**1%**	0.07	0.00	23.18	0.00	2.86	4.10	96	347
	9.9	**2%**	0.07	1.10	40.55	0.00	4.23	5.22	84	264
	11.0	**3%**	0.07	2.20	49.65	0.30	4.76	5.28	70	233
	11.0	**4%**	0.07	4.70	50.21	0.60	5.78	3.74	68	256
	11.6	**5%**	0.07	5.60	66.58	1.20	6.10	4.18	77	293
Average	10.3		0.07	2.72	46.03	0.42	4.75	4.50	79	278
40	11.3	**1%**	0.11	0.58	78.31	0.30	8.37	5.28	326	3706
	16.2	**2%**	0.11	2.63	134.64	0.50	9.75	6.09	1013	3405
	14.7	**3%**	0.11	1,67	103.39	0.50	15.08	4.98	340	1298
	14.6	**4%**	0.11	3.76	74.79	1.20	14,24	4.02	243	1815
	13.3	**5%**	0.11	2.27	53.84	0.70	18.74	3.46	124	864
Average	14.0		0.11	2.18	88.99	0.64	13.24	4.77	409	2218
50	17.2	**1%**	0.16	0.00	183.10	0.00	19.6	5.78	4790	10412
	20.2	**2%**	0.16	0.38	239.23	0.40	18.12	6.73	4200	7458
	19.0	**3%**	0.16	0.37	232.76	0.00	27.39	5.61	1944	5120
	19.8	**4%**	0.16	2.54	232.47	0.40	18.73	5.86	996	5328
	19.8	**5%**	0.16	2.51	181.17	0.00	26.45	4.32	1168	5376
Average	19.2		0.16	1.16	213.75	0.16	22.03	5.66	2620	6739

Note: Solution gaps were computed using the optimum (Z^*) obtained from column generation (i.e. $(Z_{UB}-Z^*)/Z^*$), if available. Otherwise, the linear programming bound is used (i.e. $(Z_{UB}-Z_{LP})/Z_{LP}$).

Table 2

Solution quality and time: capacity/object size = 10.
(CPU time is IBM 3090 seconds)

Problem size		Update	FFOD algorithm		SA algorithm		LP bound		CG algorithm	
N (Objects)	*M* (Bins)	rate	CPU time	Gap (%)	CPU time	Gap (%)	CPU time	Gap (%)	CPU time	Range
30	4.6	**1%**	0.06	5.00	2.20	0.70	0.60	0.00	115	295
	4.6	**2%**	0.06	3.30	2.00	0.00	0.80	0.63	103	290
	4.8	**3%**	0.06	2.20	6.97	0.00	1.00	3.75	123	463
	4.6	**4%**	0.06	2.20	6.25	0.60	1.00	2.93	71	148
	4.5	**5%**	0.06	3.10	7.72	0.60	1.10	1.09	31	155
Average	4.6		0.06	3.16	5.03	0.38	0.90	1.68	89	270
40	5.8	**1%**	0.10	1.90	5.42	0.50	1.70	0.50	977	3351
	6.4	**2%**	0.10	2.90	6.18	1.50	2.50	0.90	128	1934
	6.6	**3%**	0.10	2.80	6.02	1.40	2.80	3.00	1122	2326
	6.8	**4%**	0.10	4.70	3.05	1.40	2.90	1.70	1038	3114
	6.8	**5%**	0.10	5.30	5.59	1.50	2.90	1.70	700	1816
Average	6.5		0.10	3.52	5.25	1.26	2.56	1.56	793	2508
50	6.4	**1%**	0.14	4.00	24.65	1.80	3.10	1.90	9705	13500
	8.0	**2%**	0.15	5.80	16.78	2.40	3.70	2.30	6575	9536
	7.8	**3%**	0.15	4.90	10.08	1.10	4.80	2.20	7244	9642
	7.9	**4%**	0.15	6.40	16.41	0.80	5.20	1.40	8058	10786
	7.4	**5%**	0.15	4.60	15.25	0.80	4.90	1.50	3558	9191
Average	7.5		0.15	5.14	16.63	1.38	4.34	1.86	7028	10531
75	10.7	**1%**	0.29	4.10	37.69	4.10	11.50			
	12.3	**2%**	0.30	5.30	64.04	5.30	13.70	Same		
	12.4	**3%**	0.30	5.40	52.50	4.80	14.10	as		
	13.0	**4%**	0.30	6.30	75.39	4.90	15.20	SA		
	12.8	**5%**	0.30	6.60	59.66	4.00	15.60	Gap		
Average	12.2		0.30	5.54	57.86	4.62	14.02			
100	17.3	**1%**	9.53	5.50	120.45	5.47	16.60			
	17.3	**2%**	0.53	4.60	62.48	4.57	20.20	Same		
	16.9	**3%**	0.54	4.80	131.61	4.68	24.10	as		
	16.9	**4%**	0.54	5.95	129.07	5.21	20.90	SA		
	16.7	**5%**	0.54	5.30	144.64	4.71	24.50	Gap		
Average	17.0		0.54	5.23	117.65	4.93	21.26			

the best known solution otherwise. CPU time for CG is shown as a range–the second lowest time to second highest time over the 10 problems in each group. This approach is used instead of an average because of the high variation in solution time.

Table 3

Solution quality and time: capacity/object size = 20.
(CPU time is IBM 3090 seconds)

Problem size		Update	FFOD algorithm		SA algorithm		LP bound		CG algorithm	
N (Objects)	*M* (Bins)	rate	CPU time	Gap (%)	CPU time	Gap (%)	CPU time	Gap (%)	CPU time	Range
30	4.9	**1%**	0.06	3.04	2.26	0.00	0.33	0.90	61.2	124.8
	4.4	**2%**	0.06	2.20	5.54	0.80	0.49	0.80	0.0	210.0
	4.4	**3%**	0.06	0.77	3.16	0.80	0.04	0.00	0.2	193.3
	4.3	**4%**	0.06	5.71	3.40	0.70	0.80	0.00	0.1	153.6
	4.6	**5%**	0.06	7.71	3.81	0.00	0.90	0.75	0.5	169.8
Average	4.5		0.06	3.89	3.63	0.46	0.51		12.4	170.3
40	4.1	**1%**	0.10	0.71	3.34	0.71	0.76			
	4.1	**2%**	0.10	2.86	4.98	0.71	1.25	Same		
	4.5	**3%**	0.10	4.95	16.74	3.50	1.35	as		
	4.9	**4%**	0.10	7.58	18.82	3.50	1.61	SA		
	4.9	**5%**	0.10	4.58	18.18	3.30	1.73	Gap		
Average	4.5		0.10	4.14	12.41	2.34	1.34			
50	4.6	**1%**	0.14	4.46	23.42	3.21	2.50			
	5.4	**2%**	0.14	8.25	28.08	4.82	2.74	Same		
	6.0	**3%**	0.14	6.83	19.88	1.80	2.67	as		
	5.8	**4%**	0.14	5.98	13.86	1.70	2.79	SA		
	5.6	**5%**	0.14	8.15	27.29	3.38	2.91	Gap		
Average	5.5		0.14	6.73	22.51	2.98	2.72			

FFOD heuristic performance

The quality of the solution is quite good; the overall average solution deviated by about 3% from the optimum (where available) and by about 5.4% from the lower bound when the optimum was unavailable. Performance tends to be better on problems where the bin capacity/object size ratio is smaller.

The heuristic is very fast–no problem took over 1 second. Solution time increases with (slightly less than) the square of the number of objects. Recall that one "run" of the heuristic involves repeating the bin selection and file assignment process $2K$ times. The fast solution time suggests that the optimum might be found more often if the process were repeated a larger number of times using alternative orderings of the objects. Initial experimentation suggests that this is a very good idea.

Simulated annealing performance

SA was able to significantly improve the FFOD solution in the vast majority of problems. The average gap versus the optimum was reduced to about 0.6%; the

gap versus the lower bound, when the optimum was not available, was 3.7%. As might be expected, SA performance tends to be better on problems where the bin capacity/object size ratio is smaller. On problems where the optimum is known, SA found the optimum in all but 17 of the 150 problems solved in table 1 where capacity/size ratio is 5; in table 2, with capacity/size ratio 10, the optimum was not found in 28 of 150 problems. We suspect that in problems with relatively larger number of bins it is easier for SA to open (and close) bins thus making it less likely that SA will become stuck on a local optimum.

The better solutions came at significant increase in computation time over the FFOD heuristic, ranging from about 50 over 1000 times longer. As expected, increased number of objects increases solution time; solution time increases with roughly the cube of N. While the capacity/size ratio has almost no impact on FFOD solution time, it has an impact on solution time for SA. While problems of equal number of objects in table 2 and 3 were solved at about the same speed, further decrease in bin size (table 1) resulted in a dramatic increase in solution time.

If we consider the relationships between capacity/size ratio, solution time, and solution quality a clear pattern emerges: on problems where SA spent relatively more time, that is, on the problem sets with lower capacity/size ratio, better solutions were found. This suggests that better solutions *may* be available on the problems with larger bins if (say) the cooling schedule were modified for more gradual cooling. Such speculation must, however, be confounded by the fact that the starting solution from FFOD was generally better with smaller capacity/size ratio.

LB performance

The average gap over all problems solved to optimality is just under 3%. However, the sharpness of the bound is clearly a function of problem characteristics: the problems in table 1, where more bins are required due to low capacity/size ratio, have the poorest bounds–an average of about 5%. In table 2, where fewer bins are used, the average gap was only 1.7%. We believe that the poorer bounds in table 1 problems are easily explained by the larger number of bins which allow for more fractional assignments of objects.

Bounding time is, of course, a function of N, the number of bins in each feasible configuration, and the number of feasible configurations which must be solved. Thus, table 1 problems took more time than comparable problems in table 2 which, in turn, took longer than problems in table 3.

CG performance

Our objective in using the column-generation/branch-and-bound algorithm was primarily to validate solutions of FFOD and SA and the bounds from LB. Problems as large as 50 objects were solved to optimality by CG but at significant computing cost in many instances. In addition, solution time had very high variance

as tabled results show. Since we began with the FFOD solution and the LP lower bound, problems in which FFOD's solution was already verified optimal by the lower bound did not require any time in CG. That fact accounted for the very low times seen at the low-end of the CPU time range for a few problems.

6. Summary and concluding remarks

(1) Of the methods tested, the simple FFOD heuristic coupled with the linear programming-based lower bounding scheme appear to be the most useful and most promising. The heuristic is so fast that it can easily be applied repeatedly, using different orderings of object, to increase the chance of finding the optimum or a near-optimum solution. Experimentation is underway to evaluate the impacts of repeatedly running FFOD with random orderings of objects and other variations in the algorithm. Results are promising.

The lower bound is tight enough to provide a satisfactory "comfort level" with the heuristic solution for most practical problems.

(2) Simulated annealing results are mixed. While it typically provides improved solutions over a few iterations of the FFOD heuristic, the computation time runs to minutes instead of seconds. In addition, if SA is run without invoking the "gap < 5%" stopping rule, an even higher proportion of better solutions (versus FFOD) are found but at an even greater computing cost. There may exist other cooling schedules, neighborhood definitions, or "energy functions" which could improve the performance of SA and/or reduce its solution time.

Whether or not to use SA on a real problem comes down to an issue of cost-effectiveness–cost of computation versus savings possible in an improved solution. For those problems where a reduction in cost of a percent or so means large dollar savings, then we would recommend that SA be run unless the FFOD:LB gap is quite small. In situations where "quick-and-dirty" solutions are needed rapidly and frequently, or where solutions are needed for *very* large problems, the heuristic is probably the best choice.

Further research is underway on this special problem. In addition to modifications to the heuristic and simulated annealing methods, we are also developing an algorithm based on tabu search [29]. Other research includes algorithms based on a "Boltzmann machine" (i.e. simulated neural net) [1]. We also find the subproblem of section 4.2, that is, determining if a given bin configuration allows a feasible assignment of objects, quite intriguing. Are there quick methods, for example, which can be used to establish that a given set of bins does not support a feasible assignment?

Acknowledgements

This research was supported in part by a CBE Summer Research Stipend provided by College of Business and Economics at Washington State University.

We would like to thank Dr. Hasan Pirkul for allowing us to use his optimization code for solving multi-constraint knapsack problems. Two anonymous referees provided very careful and comprehensive reviews and many helpful suggestions. Their contributions are highly appreciated.

References

[1] E.H.L. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines* (Wiley, 1989).
[2] E.H.L. Aarts and P.J.M. van Laarhoven, Statistical cooling: A general approach to combinatorial problems, Phillips J. Res. 40(1985)193–226.
[3] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms* (Addison–Wesley, Reading, MA, 1974) chapter 10.
[4] E. Appleton and D.J. Williams, *Industrial Robot Applications* (Wiley, New York, 1987) chapter 3.
[5] B.S. Baker, E.G. Coffman and R.L. Rivest, Orthogonal packings in two dimensions, SIAM J. Comp. 9(1980)846–855.
[6] B.S. Baker, D.J. Brown and H.P. Katseff, A 5/4 algorithm for two-dimensional bin packing, J. Algor. 2(1981)348–368.
[7] B.S. Baker and J.S. Schwarz, Shelf algorithms for two-dimensional packing problems, SIAM J. Comp. 12(1983)508–525.
[8] M. Ball and M. Magazine, The design and analysis of heuristics, Networks 11(1981)215–219.
[9] J.J. Bartholdi III, J.H. Vande Vate and J. Zhang, Expected performance of the shelf heuristic for 2-dimensional packing, Oper. Res. Lett. 8(1989)11–16.
[10] S.P. Bradley, A.C. Hax and T.L. Magnanti, *Applied Mathematical Programming* (Addison–Wesley, 1977) chapter 12.
[11] D.J. Brown, B.S. Baker and H.P. Katseff, Lower bounds for on-line two-dimensional packing algorithms, Acta Inf. 18(1982)207–225.
[12] F.R.K. Chung, M.R. Garey and D.S. Johnson, On packing two-dimensional bins, SIAM J. Alg. Discr. Methods 3(1982)66–76.
[13] B. Chazelle, The bottom-left bin-packing heuristic: An efficient implementation, IEEE Trans. Comp. C-32(1983)697–707.
[14] E.G. Codd, Multiprogramming scheduling, Commun. ACM, Parts 1 and 2(1960)347–350; Parts 3 and 4 (1960) 413–418.
[15] E.G. Coffman, M.R. Garey, D.S. Johnson and R.E. Tarjan, Performance bounds for level-oriented two-dimensional packing algorithms, SIAM J. Comp. 9(1980)808–826.
[16] E.G. Coffman, M.R. Garey and D.S. Johnson, Approximation algorithms for bin-packing–an updated survey, in: *Algorithm Design for Computer System Design*, ed. G. Ausiello, M. Luccertini and P. Serafini (Springer, Vienna, 1984) pp. 49–106.
[17] J. Cook and B.T. Han, Optimal robot selection and work station assignment for a CIM system, IEEE Trans. Robotics and Automation (February, 1994), to appear.
[18] D. Coppersmith and P. Raghavan, Multidimensional on-line bin packing: algorithms and worst-case analysis, Oper. Res. Lett. 8(1989)17–20.
[19] G. Cornuejols, R. Sridharan and J.M. Thizy, A comparison of heuristics and relaxations for the capacitated plant location problem, Eur. J. Oper. Res. 50(1991)280–297.
[20] K.A. Dowsland and W.B. Dowsland, Packing problems, Eur. J. Oper. Res. 56(1992)2–14.
[21] M.L. Fisher, The Lagrangian relaxation method for solving integer programming problem, Manag. Sci. 27(1981)1–18.
[22] W. Fernandez de la Vega and G.S. Lueker, Bin packing can be solved within $1 + \varepsilon$ in linear time, Combinatorica 1(1981)349–355.

[23] M.R. Garey, R.L. Graham, D.S. Johnson and A.C.C. Yao, Resource constrained scheduling as generalized bin packing, J. Comb. Theory (A) 21(1976)257–298.
[24] B. Gavish and H. Pirkul, Efficient algorithms for solving multiconstraint zero–one knapsack problems to optimality, Math. Progr. 31(1985)78–105.
[25] A. Geoffrion and R. McBride, Lagrangian relaxation applied to capacitated facility location problems, AIIE Trans. 10(1978)40–47.
[26] P.C. Gilmore and R.E. Gomory, A linear programming approach to the cutting-stock problem, Oper. Res. 9(1961)849–859.
[27] P.C. Gilmore and R.E. Gomory, A linear programming approach to the cutting-stock problem–Part II, Oper. Res. 11(1963)863–888.
[28] P.C. Gilmore and R.E. Gomory, Multistage cutting stock problems of two and more dimensions, Oper. Res. 13(1965)94–120.
[29] F. Glover, Tabu search: a tutorial, Interfaces 20(1990)74–94.
[30] F. Glover and R. Hubscher, Bin packing with tabu search, Graduate School of Business and Administration, University of Colorado at Boulder (1991).
[31] I. Golan, Performance bounds for orthogonal oriented two-dimensional packing algorithms, SIAM J. Comp. 10(1981)571–582.
[32] B.T. Han and G. Diehr, An algorithm for device selection and file assignment, Eur. J. Oper. Res. 61(1992)326–344.
[33] B.T. Han, Optimal file management for a storage system using magnetic and optical disks, Inf. Dec. Technol., to appear.
[34] M. Hofri, Two-dimensional packing: Expected performance of simple level algorithms, Inf. Control 45(1980)1–17.
[35] W.H. Inmon, EIS and data warehouse, Database Progr. Design (November, 1992)70–73.
[36] D.S. Johnson, Fast algorithms for bin packing, J. Comp. Syst. Sci. 8(1974)272–314.
[37] D.S. Johnson, C.R. Aragon, L.A. McGeoch and C. Schevon, Optimazion by simulated annealing: An experimental evaluation: Part I, Graph partitioning, Oper. Res. 37(1989)865–892.
[38] D.S. Johnson, C.R. Aragon, L.A. McGeoch and C. Schevon, Optimization by simulated annealing: An experimental evaluation: Part II, Graph coloring and number partitioning, Oper. Res. 39(1991) 378–406.
[39] T. Kampke, Simulated annealing: Use of a new tool in bin packing, Ann Oper. Res. 16(1988) 327–332.
[40] N. Karmarkar and R.M. Karp, The differential method of set partitioning, Report UCB/CSD 82/113, Computer Science Division, University of California, Berkeley, California 94720 (1982).
[41] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, Optimization by simulated annealing, Science 220(1983)671–680.
[42] B. O'Connell, Reinventing data management, DEC Professional (February, 1993)40–45.
[43] W.T. Rhee and M. Talagrand, Multidimensional optimal bin packing with items of random size, Math. Oper. Res. 16(1991)490–503.
[44] S.C. Sarin and W.E. Wilhelm, Prototype models for two-dimensional layout design of robot systems, IIE Trans. 16(1984)206–215.
[45] D. Sleator, A 2.5 times optimal algorithm for packing in two dimensions, Inf. Proc. Lett. 10(1980) 37–40.
[46] J.D. Ullman, Complexity of sequencing problems, in: *Computer and Job-shop Scheduling Theory*, ed. E.G. Coffman, Jr. (Wiley, New York, 1975).
[47] P.J.M. van Laarhoven, *Theoretical and Computational Aspects of Simulated Annealing*, Centre for Mathematics and Computer Science, Amsterdam, The Netherlands (1988).
[48] P.J.M. van Laarhoven and E.H.L. Aarts, *Simulated Annealing*: *Theory and Applications* (Kluwer Academic, Boston, MA, 1988).
[49] A.C.C. Yao, New algorithms for bin packing, J. ACM 27(1980)207–227.