Available online at www.sciencedirect.com
ScienceDirect
Discrete Optimization 4 (2007) 206–212
DISCRETE OPTIMIZATION
www.elsevier.com/locate/disopt

Algorithms for the bounded set-up knapsack problem

Laura A. McLay[a,*], Sheldon H. Jacobson[b]

[a] *Department of Statistical Sciences and Operations Research, Virginia Commonwealth University, P.O. Box 843083, 1001 West Main Street, Richmond, VA 23284, United States*

[b] *Department of Computer Science, University of Illinois, Thomas M. Siebel Center, 201 N. Goodwin Avenue (MC-258) Urbana, IL 61801-2302, United States*

Received 20 July 2005; received in revised form 9 October 2006; accepted 8 November 2006
Available online 13 December 2006

Abstract

The Bounded Set-up Knapsack Problem (BSKP) is a generalization of the Bounded Knapsack Problem (BKP), where each item type has a set-up weight and a set-up value that are included in the knapsack and the objective function value, respectively, if any copies of that item type are in the knapsack. This paper provides three dynamic programming algorithms that solve BSKP in pseudo-polynomial time and a fully polynomial-time approximation scheme (FPTAS). A key implication from these results is that the dynamic programming algorithms and the FPTAS can also be applied to BKP. One of the dynamic programming algorithms presented solves BKP with the same time and space bounds of the best known dynamic programming algorithm for BKP. Moreover, the FPTAS improves the worst-case time bound for obtaining approximate solutions to BKP as compared to using FPTASs designed for BKP or the 0-1 Knapsack Problem.
Published by Elsevier B.V.

Keywords: Knapsack problems; Dynamic programming; Fully polynomial-time approximation schemes

1. Introduction

The Bounded Knapsack Problem (BKP) is defined by a knapsack capacity and a set of n item types, each having a positive integer value, a positive integer weight, and a positive integer bound on its availability. The objective of BKP is to select the number of each item type (subject to its availability) to add to the knapsack such that their total weight is within capacity and their total value is maximized. The 0-1 Knapsack Problem (KP) is a particular case of BKP in which only one copy of each item type may be added to the knapsack. Martello and Toth [13] and Kellerer et al. [9] summarize research related to KP and many of its variations.

This paper addresses a generalization of BKP called the Bounded Set-up Knapsack Problem (BSKP). BSKP generalizes BKP by including a non-negative *set-up weight* and a *set-up value* associated with each item type. The set-up weight (value) is included in the knapsack (objective function) if any copies of that item type are in the knapsack. The Integer Knapsack Problem with Set-up Weights (IKPSW) [15] is a particular case of BSKP in which each item type's bound is unlimited and set-up value is zero. McLay and Jacobson [15] provide dynamic programming

* Corresponding author.
E-mail addresses: lamclay@vcu.edu (L.A. McLay), shj@uiuc.edu (S.H. Jacobson).

1572-5286/$ - see front matter Published by Elsevier B.V.
doi:10.1016/j.disopt.2006.11.002

algorithms, a Greedy heuristic, and a FPTAS for IKPSW. Both BSKP and IKPSW are motivated by applications in the area of aviation security [16,14].

This paper presents three dynamic programming algorithms that solve BSKP in pseudo-polynomial time using the characteristics of BSKP. In addition, a fully polynomial-time approximation algorithm (FPTAS) is presented that obtains solutions whose values are within an arbitrary level ε of the optimal solution value. A key contribution of this paper is that the algorithms and heuristics presented also provide efficient methods for solving and approximating BKP. Algorithms for BKP are given by Martello and Toth [12], Ingargiola and Korsh [6], Pferschy [18], Pisinger [19] and Kellerer et al. [9]. One of the dynamic programming algorithms for BSKP has the same time and space complexity for solving BKP as the best dynamic programming algorithm for BKP [18]. The FPTAS for BSKP also obtains approximate solutions to BKP with a better worst-case time bound as compared to those obtained from using a FPTAS designed for KP to obtain approximate solutions to BKP.

BSKP has similarities to several other knapsack variations in addition to BKP. The Bounded Knapsack Problem with Setups is a particular case of BSKP that does not include set-up values but only set-up weights. Süral et al. [20] present a branch and bound algorithm for this problem. The Set-up Knapsack Problem (SKP) is a variation of KP that considers partitioning the items into families that are known a priori [2]. There is an item associated with each family that must be added to the knapsack before adding any items in its family, and the weight and value of this item can take on real numbers that may be negative, rather than non-negative integers as in BSKP.

BSKP is a particular instance of precedence-constrained knapsack problems [5,7,1,17]. To see this, construct a BKP instance with two item types for every item type in BSKP. In particular, for each item type in BSKP, first create an item type with weight equal to the sum of the item type's weight and set-up weight, value equal to the sum of the item type's value and set-up value, and a bound of one. Then create a second BKP item type with the same weight and value of the BSKP item type, and with bound one less than that of the BSKP item type. The set of precedence constraints guarantees that each item with the set-up weight and set-up value is added before the other items of that type.

BSKP can also be transformed into a particular instance of the Multiple-Choice Knapsack Problem (MCKP). In MCKP, the set of items are partitioned into classes, and exactly one item from each class must be added to the knapsack. To transform BSKP into an instance of MCKP, a class is created for each item type. In each class, a set of items are created to account for all possible multiplicities of the item type, up to the given bound (i.e., the MCKP value is a multiple of the value, and the MCKP weight is the sum of the set-up weight and a multiple of the weight), including an item with weight and value equal to zero. The size of the MCKP instance depends on the size of the bounds. Note that the transformation to MCKP allows for the negative set-up weights and set-up values.

This paper is organized as follows. Section 2 introduces BSKP and formulates BSKP as an integer program. Section 3 describes three pseudo-polynomial time dynamic programming algorithms for solving BSKP. Section 4 presents two approximation algorithms for BSKP, including a Greedy heuristic and a FPTAS for BSKP, which obtain solutions to BSKP that are within a factor of $1/2$ and $\varepsilon \in (0, 1/2]$ of the optimal solution value, respectively. Section 5 analyzes the performance of the dynamic programming algorithms and the FPTAS in Sections 3 and 4 when applied to BKP. Section 6 provides concluding comments and directions for future research.

2. Problem formulation

BSKP generalizes IKPSW by including a bound for the availability of each item type and a set-up value associated with each item type that is added to the objective value if any copies of that item type are added to the knapsack. BSKP is given by n item types, values $v_i \in Z_0^+$, set-up values $u_i \in Z_0^+$, weights $w_i \in Z^+$, set-up weights $s_i \in Z_0^+$, and bounds $b_i \in Z^+$ corresponding to each item type $i = 1, 2, \ldots, n$, and knapsack capacity $c \in Z^+$. Without loss of generality, assume that $s_i + b_i w_i \leq c$, $i = 1, 2, \ldots, n$ (i.e., the bounds are defined such that b_i copies of item type $i = 1, 2, \ldots, n$ can fit into the knapsack) and that $\sum_{i=1}^{n}(s_i + b_i w_i) > c$ to ensure a nontrivial solution. Note that this implies that $b_i \leq \lfloor (c - s_i)/w_i \rfloor$, $i = 1, 2, \ldots, n$. BSKP is trivially NP-hard since it is a generalization of IKPSW, which itself is NP-hard [15].

BSKP can be formulated as an integer programming (IP) model, where the integer decision variables x_i indicate how many items of type $i = 1, 2, \ldots, n$ are added to the knapsack, and the binary decision variables y_i indicate if any

copies of item type $i = 1, 2, \ldots, n$ are in the knapsack.

$$\max \sum_{i=1}^{n} v_i x_i + \sum_{i=1}^{n} u_i y_i \tag{1}$$

$$\text{subject to } \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} s_i y_i \leq c \tag{2}$$

$$\frac{1}{b_i} x_i - y_i \leq 0, \quad i = 1, 2, \ldots, n \tag{3}$$

$$y_i \leq x_i, \quad i = 1, 2, \ldots, n \tag{4}$$

$$x_i \in \{0, 1, \ldots, b_i\}, \quad i = 1, 2, \ldots, n, \tag{5}$$

$$y_i \in \{0, 1\}, \quad i = 1, 2, \ldots, n. \tag{6}$$

In (1), the objective of BSKP is to maximize the total value of the items in the knapsack, including set-up values. The first constraint, (2), ensures that the weight of the items in the knapsack, including their set-up weights, do not exceed the knapsack capacity. The second and third sets of $2n$ constraints, (3) and (4), guarantee that $y_i = 1$ if any items of type i are in the knapsack (i.e., $x_i > 0$) and that $y_i = 0$ otherwise, $i = 1, 2, \ldots, n$. The fourth set of n constraints, (5), indicates that x_i is a non-negative integer within its bounds, and the final set of n constraints, (6), indicates that y_i is binary, $i = 1, 2, \ldots, n$.

3. Dynamic programming algorithms

This section introduces three dynamic programming algorithms for BSKP that obtain the optimal solution set and its value. The details of these algorithms are given in [14].

The first dynamic programming algorithm (labeled DP-W1) uses a nested approach to solve BSKP in $O(nc)$ time and space by extending an algorithm for IKPSW to handle set-up values and bounds [15]. To describe DP-W1, define $z_r(\bar{c})$, $r = 1, 2, \ldots, n$, $\bar{c} = 0, 1, \ldots, c$, as the optimal solution value to the knapsack subproblem defined on the first r item types with capacity $\bar{c}$, with $x_r(\bar{c})$ defined as the corresponding optimal number of copies of item type r. Define $\bar{z}_r(\bar{c})$ as the optimal value over the first $r = 1, 2, \ldots, n$ item types with knapsack capacity $\bar{c} = 0, 1, \ldots, c$ given such that $1 \leq \bar{x}_r(\bar{c}) < b_r$ copies of item type r are present in the knapsack,

$$\bar{z}_r(\bar{c}) = \max\{\bar{z}_r(\bar{c} - w_r) + v_r, z_{r-1}(\bar{c} - w_r - s_r) + v_r + u_r\}.$$

Define $\bar{z}^*_r(\bar{c})$ as the optimal value over the first r item types with capacity $\bar{c}$, given that b_r copies of item type r are in the knapsack. At each step of the recursion, $z_r(\bar{c})$ adds either no items of type r to the knapsack (using $z_{r-1}(\bar{c})$), adds at least one copy of item type r to the knapsack (using $\bar{z}_r(\bar{c})$), or adds b_r copies of item type r to the knapsack (using $\bar{z}^*_r(\bar{c})$), calling at most two other recursions,

$$z_r(\bar{c}) = \max\{z_{r-1}(\bar{c}), \bar{z}_r(\bar{c}), \bar{z}^*_r(\bar{c})\}.$$

If only the optimal value is desired then DP-W1 can be modified to reduce the space bound to $O(n + c)$.

The second dynamic programming algorithm (labeled DP-DC) solves BSKP in $O(nc)$ time and $O(n + c)$ space by applying a storage reduction scheme using a recursive "divide and conquer" approach [3,18]. To solve BSKP, the set of items is divided such that the cardinality of each set is approximately equal, creating two subproblems. BSKP is solved over each subset of items given capacity $0, 1, \ldots, c$. The solutions to both subproblems are combined, and the optimal number of items of one item type from each subproblem is known. Each subproblem is then divided into two more subproblems, and this process is repeated until the optimal solution set is known. Pferschy [18] shows how this approach requires $O(nc)$ time and $O(n + c)$ space.

The final dynamic programming algorithm solves BSKP with lists. List L_r contains a series of m entries over the first r item types

$$L_r = \langle (V_1, W_1, I_1), (V_2, W_2, I_2), \ldots, (V_m, W_m, I_m) \rangle,$$

where V_r denotes the value (including set-up values) of the partial knapsack solution, W_r denotes the corresponding weight (including set-up weights) of the partial knapsack solution, and I_r is the set of items in the knapsack and their multiplicity. The dominated entries are removed, and the lists are sorted in increasing value and weight, resulting in low storage requirements. This results in algorithm DP-L1, described in pseudo-code by Algorithm 1.

Algorithm 1 BSKP Dynamic Programming Algorithm DP-L1

$L_0 = \langle(0, 0, \emptyset)\rangle$
for $r = 1$ to n **do**
  Add item type r (including set-up weights and values) to all elements in L_{r-1}, yielding L'_{r-1}.
  Add item type r (*not* including set-up weights and values) to all elements in L'_{r-1}, keeping between 1 and $b_r - 1$ copies of item type r, yielding L'_{r-1}.
  Add b_r copies of item type r (including set-up weights and values) to all elements in L_{r-1}, yielding L''_{r-1}.
  L_r merges L_{r-1}, L'_{r-1}, and L''_{r-1}.
end for
return the largest state in L_n

There are three lists in DP-L1, L_{r-1} contains the optimal knapsack subproblem solutions over the first $r-1$ item types, L'_{r-1} contains the optimal knapsack subproblem solutions over the first r item types given that there are between 1 and $b_r - 1$ copies of item type r in the knapsack, and L''_{r-1} contains the optimal knapsack subproblem solutions over the first r item types given that there are b_r copies of item type r in the knapsack. When constructing L'_{r-1}, single copies of item type r are added to existing entries in the list from smallest to largest weight and value, which allows multiple copies of each item type to be added to the partial knapsack solutions. In particular, item type r is added to each entry in constant time if L'_{r-1} is a linked list, with new items being added to the beginning of the list. Note that first item type ($r = 1$) can be added to all items in the current list as long as there is room in the knapsack. Additional items of type $r = 1, 2, \ldots, n$ can be added, but the bound b_r must be first checked. Let U be an upper bound on the optimal objective function value. Kellerer et al. [9, p. 52] show how the two lists can be merged in $O(\min\{c, U\})$ time using pointers, and their method is trivially modified to merge three lists. The resulting algorithm requires $O(n \min\{c, U\})$ time and $O(n \min\{c, U\})$ space.

4. Fully polynomial-time approximation scheme

This section presents a FPTAS for BSKP. To describe this FPTAS, define a *reward* as an item type $i \in R$ with bound $b_i > 1$ such that either $u_i/s_i > v_i/w_i$ if $s_i > 0$, or $u_i > 0$ if $s_i = 0$. Alternatively, define a *penalty* item type $i \in P$ as an item type such that either $u_i/s_i \leq v_i/w_i$ if $s_i > 0$, or $u_i = 0$ if $s_i = 0$. Note that R and P partition the set of item types, $\{1, 2, \ldots, n\}$.

The FPTAS uses the BSKP Greedy heuristic, labeled $H_B^{1/2}$, which obtains solutions that are within a factor of $1/2$ of the optimal solution in $O(n)$ time. An integer feasible solution for BSKP can be easily constructed from the linear programming relaxation of BSKP, yielding z^g [14]. Let $z^b = \max_i\{u_i + b_i v_i\}$. Then, $H_B^{1/2}$ yields the objective function value

$$z^h = \max\{z^g, z^b\}.$$

The absolute error, given by $E = z - z^h$, where z is the optimal objective function value, can be determined by the critical item type in the linear programming relaxation. McLay [14] shows that $H_B^{1/2}$ provides an integer feasible solution whose objective function value is greater than half of the optimal objective function value and that this bound is tight.

The BSKP approximation algorithm, $F_B(\varepsilon)$, obtains solutions whose values are within a factor of ε of the optimal solution value. There is a preprocessing phase and two main phases, the dynamic programming and Greedy phases, in the execution of $F_B(\varepsilon)$ for $\varepsilon < 1/2$.

In the preprocessing phase, $H_B^{1/2}$ is called to find a lower bound on the optimal value, z^h. Let the threshold value be defined as $\theta = \varepsilon z^h/2$, and let the scale factor be defined as $\delta = \varepsilon^2 z^h/4$. The set of item types can be partitioned into four subsets, Dynamic Programming item types (D), Greedy item types (G), Transition Penalty item types (T_P)

and Transition Reward item types (T_R). An item type $i \in P$ is a Dynamic Programming item if $v_i \geq \theta$, and it is a Greedy item type if $v_i b_i + u_i \leq \theta$. Otherwise, item type i is a Transition Penalty item type (i.e., $v_i b_i + u_i > \theta$ and $v_i < \theta$). An item type $i \in R$ is a Dynamic Programming item if $v_i \geq \theta$, and it is a Greedy item type if $v_i + u_i \leq \theta$. Otherwise, item type i is a Transition Reward item type (i.e., $v_i + u_i > \theta$ and $v_i < \theta$).

The Dynamic Programming, Transition Penalty and Transition Reward item types are used to form an instance of the *augmented problem* with $n_A = |D| + |T_R| + |T_P| \leq n$ item types as follows. An item type in the augmented problem is created for each item type $i \in D$ with weight w_i, set-up weight s_i, value v_i, set-up value u_i, and bound b_i. The items are added during the dynamic programming phase with scaled values $\lfloor (v_i + u_i)/\delta \rfloor$ for the first copy and $\lfloor v_i/\delta \rfloor$ for the remaining copies. An item type in the augmented problem is created for each item type $i \in T_R$ with weight w_i, set-up weight s_i, value v_i, set-up value u_i, and bound 1. The item is added during the dynamic programming phase with scaled value $\lfloor (v_i + u_i)/\delta \rfloor$. For every item type $i \in T_P$, first find the smallest integer q_i such that $q_i v_i > \theta$. An item type in the augmented problem is created for each item type $i \in T_P$ with weight $q_i w_i$, set-up weight s_i, value $q_i v_i$, set-up value u_i, and bound $\lfloor b_i/q_i \rfloor$. The items are added during the dynamic programming phase with scaled values $\lfloor (q_i v_i + u_i)/\delta \rfloor$ for the first item type and $\lfloor q_i v_i/\delta \rfloor$ for the remaining item types, adding q_i copies each time.

The dynamic programming algorithm DP-L1 solves the augmented problem using the scaled values instead of the values and set-up values. Note that DP-L1 considers at most

$$\left\lfloor \frac{z}{\delta} \right\rfloor \leq \frac{2z^h}{\frac{1}{4}\varepsilon^2 z^h} = \frac{8}{\varepsilon^2}$$

total states.

The Greedy phase uses $H_B^{1/2}$ to insert the Greedy, Transition Penalty, and Transition Reward item types into the dynamic programming solutions. Copies of Greedy item types can be added to any dynamic programming solution. However, a Transition Penalty or a Transition Reward item type can only be added to dynamic programming solution with at least one copy of that item type in the knapsack, with the set-up weight added to the knapsack and the set-up value included in the objective function value. Therefore, the set-up weight and the set-up value for any item type $i \in T_P$ or $i \in T_R$ are set to zero in the Greedy phase.

The Greedy phase executes $H_B^{1/2}$ using the Greedy, Transition Penalty, and Transition Reward item types as in $H_B^{1/2}$. The item types considered in the Greedy phase are inserted into each defined state considered in the dynamic programming phase. A Transition Reward or a Transition Penalty item type are not considered if they are not present in the dynamic programming solution. Therefore, the Greedy phase requires $O(n/\varepsilon^2)$ time to execute.

The total time and space bounds for $F_B(\varepsilon)$ are $O(n/\varepsilon^2)$ and $O(n + 1/\varepsilon^3)$, respectively. To see this, note that the dynamic programming phase considers at most $8/\varepsilon^2$ states, and there are no more than $4/\varepsilon$ items in the dynamic programming solution. These bounds may be improved by using techniques given by Lawler [10] and Kellerer et al. [9]. Theorem 1 shows that the solution obtained by $F_B(\varepsilon)$ is within a factor of ε of the optimal solution.

Theorem 1. *$F_B(\varepsilon)$ is a FPTAS for BSKP.*

Proof. Let z^f be the value of the solution obtained by $F_B(\varepsilon)$, and let z be the optimal solution value. Error between the heuristic and optimal solutions accumulates in the dynamic programming and the Greedy phases. Note that for any item i in the augmented problem with value $\hat{v}_i$, set-up value $\hat{u}_i$, scaled value for the first copy added to the knapsack $\bar{u}_i$, and scaled value for the remaining copies $\bar{v}_i$, $\delta\bar{u}_i \leq \hat{v}_i + \hat{u}_i < \delta(\bar{u}_i + 1)$ if the first copy of item type i is added, and $\delta\bar{v}_i \leq \hat{v}_i < \delta(\bar{v}_i + 1)$ otherwise. Since the augmented problem is defined such that no more than z/θ items can fit in any of the dynamic programming solutions, then the error during the dynamic programming phase is limited to $\delta z/\theta$.

The absolute error made by $H_B^{1/2}$ in the Greedy phase is limited by θ. To see this, note that the absolute error is limited by

$$\max\{\max_{i \in G}\{v_i b_i + u_i\}, \max_{i \in T_P}\{v_i\}, \max_{i \in T_R}\{v_i\}\} \leq \max\{\theta, \theta, \theta\} = \theta.$$

Therefore, the relative error is bounded above by ε since

$$\frac{z - z^f}{z} \leq \frac{\delta}{\theta} + \frac{\theta}{z} = \frac{\varepsilon^2 z^h/4}{\varepsilon z^h/2} + \frac{\varepsilon z^h/2}{z} \leq \varepsilon. \quad \square$$

5. Application to the bounded knapsack problem

There is a dearth of specialized algorithms and heuristics for BKP. The dynamic programming algorithms in Section 3 and the FPTAS $F_B^{1/2}$ can efficiently solve or approximate BKP.

The dynamic programming algorithms can be applied to solve BKP without modification. Therefore, DP-W1 solves BKP in O(nc) time and space, DP-DC solves BKP in O(nc) time and O($n+c$) space, and DP-L1 solves BKP in O($n \min\{c, U\}$) time and space. Pferschy [18] presents a dynamic programming algorithm that solves BKP in O(nc) time and O($n + c$) space. Therefore, DP-DC solves BKP and BSKP, its generalization, with the same time and space requirements. It is unclear how the dynamic programming algorithm given by Pferschy [18] could be modified to solve BSKP.

FPTASs for KP typically are used to obtain approximate solutions to BKP by converting BKP to a KP instance [4,10,11,13]. Doing so creates an instance of KP with

$$\hat{n} = \sum_{i=1}^{n} \lceil \log_2(b_i + 1) \rceil$$

items. The FPTAS in [8] executes in O($\hat{n} \min\{\hat{n}, \log(1/\varepsilon)\} + (1/\varepsilon^2) \log(1/\varepsilon) \min\{\hat{n}, 1/\varepsilon \log(1/\varepsilon)\}$) time and O($\hat{n} + 1/\varepsilon^2$) space, the best time and space bounds of a FPTAS for BKP.

The FPTAS $F_B(\varepsilon)$ can be modified to obtain ε-approximate solutions to BKP in O(n/ε^2) time and O($n + 1/\varepsilon^3$) space, using the Greedy heuristic for BKP (instead of $H_B^{1/2}$) and DP-L1. The worst-case time bound during the Greedy phase is O($n \log(1/\varepsilon) + 1/\varepsilon^2$) if the item types are sorted in nondecreasing order of their value-to-weight ratios. Therefore, $F_B(\varepsilon)$ has a better worst-case time complexity than applying any of the FPTASs designed for KP to BKP, although Kellerer and Pferschy [8] and Pferschy [18] provide a better space bound. Note that direct comparison of these time complexities depends on the relationship between n and $1/\varepsilon$ for particular instances of BKP.

6. Conclusions

The results presented in this paper provide a detailed analysis of the Bounded Set-up Knapsack Problem, a generalization of the Bounded Knapsack Problem, and the Integer Knapsack Problem with Set-up Weights. Specialized algorithms are presented for BSKP, included three dynamic programming algorithms and a FPTAS. An implication of this research is that the dynamic programming algorithms and the FPTAS $F_B(\varepsilon)$ can be applied to BKP. Although the specialized dynamic programming algorithms presented solve BSKP in pseudo-polynomial time, they are not practical for solving large problem instances. Work is in progress to design more efficient methods to solve BSKP that exploit the characteristics of this problems, including the bounds, set-up weights and set-up values.

Acknowledgments

The authors would like to thank the Associate Editor and two anonymous referees for their insightful comments and feedback, which has resulted in a significantly improved manuscript. This research has been supported in part by the National Science Foundation (DMI-0114499) and the Air Force Office of Scientific Research (FA9550-04-1-0110).

References

[1] E.A. Boyd, Polyhedral results for the precedence-constrained knapsack problem, Discrete Applied Mathematics 41 (1993) 185–201.
[2] E.E. Chajakis, M. Guignard, Exact algorithms for the setup knapsack problem, INFOR 32 (3) (1994) 124–142.
[3] D.S. Hirschberg, Algorithms for the longest common subsequence problem, Journal of the ACM 24 (4) (1977) 664–675.
[4] O.H. Ibarra, C.E. Kim, Fast approximation algorithms for the knapsack and sum of subset problems, Journal of the ACM 22 (4) (1975) 463–468.
[5] O.H. Ibarra, C.E. Kim, Approximation algorithms for certain scheduling problems, Mathematics of Operations Research 3 (3) (1978) 197–204.
[6] G.P. Ingargiola, J.F. Korsh, A general algorithm for the one-dimensional knapsack problem, Operations Research 25 (1977) 752–759.
[7] D.S. Johnson, K.A. Niemi, On knapsacks, partitions, and a new dynamic programming technique for trees, Mathematics of Operations Research 8 (1) (1983) 1–14.
[8] H. Kellerer, U. Pferschy, Improved dynamic programming in connection with an FPTAS for the knapsack problem, Journal of Combinatorial Optimization 8 (2004) 5–11.
[9] H. Kellerer, U. Pferschy, D. Pisinger, Knapsack Problems, Springer-Verlag, Berlin, 2004.

[10] E.L. Lawler, Fast approximation algorithms for knapsack problems, Mathematics of Operations Research 4 (4) (1979) 339–356.
[11] M.J. Magazine, O. Oguz, A fully polynomial approximation algorithm for the 0-1 knapsack problem, European Journal of Operational Research 8 (1981) 270–273.
[12] S. Martello, P. Toth, Chapter branch and bound algorithms for the solution of general unidimensional knapsack problem, in: Advances in Operations Research, North Holland, Amsterdam, 1977, pp. 295–301.
[13] S. Martello, P. Toth, Knapsack Problems: Algorithms and Computer Implementations, Wiley & Sons, New York, NY, 1990.
[14] L.A. McLay, Designing effective aviation security systems: Theory and practice, Ph.D. Thesis, University of Illinois, Urbana, IL, 2006.
[15] L.A. McLay, S.H. Jacobson, Integer knapsack problems with set-up weights, Computational Optimization and Applications (2007) (in press).
[16] L.A. McLay, S.H. Jacobson, J.E. Kobza, A multilevel passenger prescreening problem for aviation security, Naval Research Logistics 53 (3) (2006) 183–197.
[17] K. Park, S. Park, Lifting cover inequalities for the prcedence-constrained knapsack problem, Discrete Applied Mathematics 72 (1997) 219–241.
[18] U. Pferschy, Dynamic programming revisited: Improving knapsack algorithms, Computing 63 (4) (1999) 419–430.
[19] D. Pisinger, A minimal algorithm for the bounded knapsack problem, INFORMS Journal on Computing 12 (1) (2000) 75–82.
[20] H. Süral, L.N. van Wassenhove, C.N. Potts, The bounded knapsack problem with setups, Technical Report, INSEAD, Centre for the Management of Environmental Resources Report 97/71/TM, Cedex, France, 1997.