A Greedy Algorithm for the General Multidimensional Knapsack Problem[1]

Yalçın Akçay • Haijun Li • Susan H. Xu

College of Administrative Sciences and Economics,
Koç University,
Rumeli Feneri Yolu, Sariyer, Istanbul, 80910, Turkey

Department of Mathematics,
Washington State University,
Pullman, Washington 99164, USA

Department of Supply Chain and Information Systems,
The Smeal College of Business Administration,
The Pennsylvania State University,
University Park, Pennsylvania 16802, USA

yakcay@ku.edu.tr • lih@math.wsu.edu • shx@psu.edu

Abstract: The multidimensional knapsack problem (MDKP) is a knapsack problem with multiple resource constraints. Both the general and the 0-1 versions of this problem have a wide array of practical applications. The MDKP is known to be strongly NP-hard. In this paper, we propose a new greedy-like heuristic method, which is primarily intended for the general MDKP, but proves itself effective also for the 0-1 MDKP. Our heuristic differs from the existing greedy-like heuristics in two aspects. First, existing heuristics rely on each item's aggregate consumption of resources to make item selection decisions, whereas our heuristic uses the *effective capacity*, defined as the maximum number of copies of an item that can be accepted if the entire knapsack were to be used for that item alone, as the criterion to make item selection decisions. Second, other methods increment the value of each decision variable only by one unit, whereas our heuristic adds decision variables to the solution in batches and consequently improves computational efficiency significantly for large-scale problems. We conduct intensive numerical studies on randomly generated test problems with a wide range of parameter settings and the benchmark problems from the related literature. We demonstrate that the new heuristic significantly improves computational efficiency of the existing methods and generates robust and near-optimal solutions. The new heuristic proves especially efficient for high dimensional knapsack problems with small-to-moderate numbers of decision variables, usually considered as "hard" MDKP and no computationally efficient heuristic is available to treat such problems.

(Keywords: *Integer programming, multidimensional knapsack problems; heuristics*)

1. Introduction

The purpose of this paper is to propose an efficient greedy algorithm for the *general multidimensional knapsack problem* (MDKP). The general MDKP and its variants have been widely used to model many practical problems such as project selection (Shih 1979, Kleywegt & Papastavrou 2001), capital budgeting (Lu, Chiu & Cox 1999), stock cutting (Caprara, Kellerer, Pferschy & Pisinger 2000), combinatorial auctions (de Vries & Vokra 2001), and inventory allocation in assemble-to-order systems (Akcay & Xu 2004), among others.

[1]Supported in part by the NSF grant DMI 9812994.

The general MDKP can be stated as follows. Consider a knapsack with m-dimensions, with the capacity of the ith dimension being b_i, $i = 1, \ldots, m$. There are n different items and let u_j be the number of copies of item j, $j = 1, 2, \ldots, n$. The jth item requires a_{ij} units of the ith dimension of the knapsack. The reward of including a single copy of item j in the knapsack is c_j. The objective is to maximize the total reward of the selected items. Then, the problem can be formulated as an integer program as follows:

$$\begin{array}{lll} \max Z = & \sum_{j=1}^{n} c_j x_j & \\ \text{subject to} & \sum_{j=1}^{n} a_{ij} x_j \leq b_i, & i = 1, 2, \ldots, m, \\ & x_j \leq u_j, & j = 1, 2, \ldots, n, \\ & x_j \geq 0, \text{ and integer}, & j = 1, 2, \ldots, n, \end{array} \tag{1}$$

When the decision variables are constrained to be binary variables ($u_j = 1$ and $x_j \in \{0, 1\}$, $j = 1, \ldots, n$), the above problem is commonly referred to as the 0-1 MDKP. Both the 0-1 MDKP and general MDKP are *strongly* NP-hard (Garey & Johnson 1979, Lin 1998). Consequently, significant research efforts have been directed towards finding efficient heuristic methods for MDKP.

The heuristic solution methods for the 0-1 MDKP have generated a great deal of interest in the literature. Senju & Toyoda (1968) proposed a *dual gradient* method that starts with a possibly infeasible initial solution ($x_j = 1$ for all j) and achieves feasibility by following an effective gradient path and dropping the non-rewarding variables one by one. Toyoda (1975) developed a *primal gradient* method that improves the initial feasible solution ($x_j = 0$ for all j) by incrementing the value of the decision variable with the steepest effective gradient. Loulou & Michaelides (1979) developed a greedy-like algorithm based on Toyoda's primal gradient method. Magazine & Oguz (1984) proposed a heuristic method combining Senju and Toyoda's dual gradient algorithm and Everett (1963)'s generalized Lagrange multipliers approach. Volgenant & Zoon (1990) improved Magazine and Oguz's heuristic by computing more than one multiplier value in each step and readjusting these values at the end of the algorithm. In addition, Pirkul (1987) presented an efficient algorithm that first constructs a standard 0-1 knapsack problem using the dual variables (known as the *surrogate multipliers*) obtained from the linear programming relaxation of the 0-1 MDKP, and then solves this simpler problem using a greedy approach. Other than these greedy and bound based methods, the literature also offers more complicated tabu search and genetic algorithm based heuristics. Chu & Beasley (1998) introduced a genetic algorithm and showed that it generates the best solutions for the benchmark problems in the literature. Their genetic algorithm, however, demands significantly more computational effort. We also refer the reader to an extensive

survey paper by Lin (1998) and the references therein on the results for the 0-1 MDKP and other non-standard knapsack problems.

Unlike the extensive researches that have been conducted for the 0-1 MDKP, the solution approaches for the general MDKP are scarce. Generalizations of the approximate solution methods intended for the 0-1 MDKP fail to yield efficient solutions for the general MDKP, unless the number of resource constraints or the upper bounds of the decision variables is small (Lin 1998). To the best of our knowledge, only two heuristics have been reported in the literature that are primarily developed for the general MDKP. Kochenberger, McCarl & Wyman (1974) generalized Toyoda's primal gradient algorithm, developed for the 0-1 MDKP, to handle the general MDKP. Pirkul & Narasimhan (1986) extended the approximate algorithm of Pirkul for the 0-1 MDKP to solve the general MDKP.

A common feature of these heuristics in the literature is the concept of the *effective gradient*. The effective gradient of an item is defined as the ratio (referred to as the "bang-for-buck ratio") of the reward of that item to its *aggregate* resource consumption among all resource constraints, usually computed as $\sum_i \bar{b}_i/a_{ij}$ for item j, where $\bar{b}_i$ is the remaining capacity of constraint i. The effective gradient method then increments the value of the item with the largest effective gradient in the solution. Loosely speaking, the effective gradient method and its variants use a "max-sum" criterion to make item selection decisions, where the "max" is over items j and the "sum" is over constraints i. In effect, these methods reduce a multidimensional problem to a single dimensional problem. We believe that the use of the *aggregate* resource consumption of an item in the effective gradient measure has several drawbacks. First, it does not distinguish different degrees of resource slackness and hence may select an item that results in bottleneck conditions for the constraints. Second, the effective gradient measure does not guarantee the feasibility of the selected item and thus additional computations are necessary to exam the solution feasibility in each iteration. Moreover, both heuristics for the general MDKP are generalizations based on their respective 0-1 MDKP counterparts and, as such, they inherently increment the value of each decision variable only by one unit in each iteration. This inevitably leads to computational inefficiency when the decision variables can assume large values.

In this paper, we introduce a new greedy-like heuristic method, named *primal effective capacity heuristic* (PECH), for the general MDKP based on a different approach. PECH is developed on the notion of *effective capacity*, defined by $\min_i\{\bar{b}_i/a_{ij}\}$, which is intuitively understood as the maximum number of copies of item j that can be accepted if the entire remaining capacity of the knapsack were to be used for that item alone. PECH starts with the feasible solution ($x_j = 0$ for all j) and

in each iteration selects the item that generates the highest total reward with its effective capacity, and commits a proportion, say α, of its effective capacity to that item. As such, PECH *always* generates a feasible solution in each iteration of the algorithm. In essence, this method uses a "max-min" criterion to make variable selection decisions and reflects the multidimensional nature of the general MDKP. In addition, PECH depletes the effective capacity at a geometric rate α and in the meantime avoids creating *bottleneck* conditions for other items. The ability of PECH to generate a feasible solution and to include multiple copies of the selected item in the solution in each iteration results in superior computational efficiency over other heuristics. It is worth noting that although our heuristic is primarily developed for the general MDKP, it can be tailored to treat the 0-1 MDKP.

In our computational study, we run experiments to test the performances of the two versions of our MDKP heuristics against that of other heuristic methods. In the general MDKP experiment, we compare the performance of various heuristics using a set of randomly generated problems with a wide range of parameter configurations. Here, we find that PECH dominates all other heuristics in computational time and provides the best solution among the studied heuristics in 61.2% of the cases. We also find that PECH is remarkably effective for the problem instances with large numbers of constraints (typically greater than 100) and small-to-medium numbers of decision variables (typically less than 200), usually considered as "hard" multidimensional knapsack problems, and excels when the tightness of the knapsack's constraints improve. This is not surprising, as the "max-min" criterion of PECH makes it particularly powerful to handle high-dimensional knapsack problems. As such, it complements the effective gradient methods and its variants, as the "max-sum" criterion used in those methods is generally effective for low-dimensional knapsack problems. In the 0-1 MDKP experiments, we compare the performance of various heuristics using a set of randomly generated problems and also the benchmark problems from the literature. For the randomly generated problem set, PECH again dominates all other heuristics in computational time and provides the best solution among the studied heuristics in 47.5% of the cases. For the benchmark problems, we compared PECH with four other algorithms, including the genetic algorithm by Chu & Beasley (1998). We find that PECH ranks the first in terms of computational efficiency and the third in terms of solution quality. The two better heuristics for the benchmark problems are the genetic algorithm and Pirkul & Narasimhan (1986)'s LP-based method. However, we note the following. First, these two algorithms are optimization-based heuristics and requires very significant computational efforts than that for the greedy-like heuristics, and the latter is more suitable to treat the reoccurring, large-scale problems that demand fast solutions. Second, as mentioned earlier, PECH is the most effective for the MDKP with a large number of constraints and a small-to-moderate number of

decision variables. The benchmark problems, however, have relatively small numbers of constraints (up to 30). Third, PECH is extremely simple and designed primarily for the general MDKP, but still outperforms other greedy-like heuristics designed for the 0-1 MDKP.

The rest of the paper is organized as follows. Section 2 proposes the primal effective capacity heuristic for solving the general MDKP and the 0-1 MDKP, and briefly discusses computational complexity issues. Sections 3 and 4 report our computational results for the general and 0-1 MDKP, respectively, and compare them with these aforementioned heuristic methods. Finally, Section 5 provides final remarks and concludes the paper.

2. The Primal Effective Capacity Heuristic (PECH) for the General MDKP

Let $\lfloor a \rfloor$ be the largest integer not exceeding a. Let α, and $0 < \alpha \leq 1$, be a control parameter that determines at what rate the slack of the knapsack is committed to the selected item. The following algorithm, PECH$_\alpha$, describes an approximate solution procedure for the general MDKP formulated in (1).

Algorithm 1 (The Algorithm of PECH$_\alpha$ for solving the general MDKP)

BEGIN

STEP 1 **Initialize** decision variables $x_j = 0,\ \forall j$;
Initialize set $E = \{j | x_j = 0,\ \forall j\}$;
Initialize capacities of resources $\bar{b}_i = b_i,\ \forall i$;
Initialize upper bounds of decision variables $\bar{u}_j = u_j,\ \forall j$;

STEP 2 **Compute** effective capacity for item j: $\bar{y}_j = \min_i \left\{ \left\lfloor \frac{\bar{b}_i}{a_{ij}} \right\rfloor : a_{ij} > 0 \right\}, \forall j \in E$.
If $\bar{y}_j = 0, \forall j \in E$, then **go to** END, otherwise **go to** STEP 3.

STEP 3 **Compute** $c_j \times \bar{y}_j, \forall j \in E$ and **select** $j^* = \arg\max_{j \in E} \{c_j \times \bar{y}_j\}$.

STEP 4 **Compute** the increment of item j^*: $y_{j^*} = \min \left\{ \bar{u}_{j^*}, \max\{1, \lfloor \alpha \bar{y}_{j^*} \rfloor\} \right\}$.

STEP 5 **Update** the values of decision variables: $x_{j^*} \leftarrow x_{j^*} + y_{j^*}$;
Update remaining capacities of constraints: $\bar{b}_i \leftarrow \bar{b}_i - a_{ij^*} \times y_{j^*},\ \forall i$;
Update slacks of decision variables: $\bar{u}_{j^*} \leftarrow \bar{u}_{j^*} - y_{j^*}$;
Update set E: If $\bar{u}_{j^*} = 0$ or $\alpha = 1$, **set** $E \leftarrow E - \{j^*\}$;
If $E = \emptyset$, **go to** END; otherwise **go to** STEP 2.

END

A brief explanation of PECH$_\alpha$ is in order: STEP 1 sets the initial feasible solution to zero and also initializes resource constraints and upper bounds of decision variables to their respective initial values. STEP 2 computes, for each item $j \in E$, its *effective capacity* $\bar{y}_j$, which is the maximum number of copies of item j that can be accepted with the available capacity of the knapsack. STEP 3 computes $c_j \times \bar{y}_j$, the maximum reward of item j were the entire remaining capacity of the knapsack

dedicated to the item, $\forall j$, and selects the item j^* that has the largest attainable reward. STEP 4 accepts either $\max\{1, \lfloor \alpha \bar{y}_{j^*} \rfloor\}$ units of item j^*, which is $\alpha \times 100\%$ of the effective capacity of item j^*, or its current upper bound $\bar{u}_{j^*}$, whichever is smaller. Finally, STEP 5 updates the information and returns to STEP 2 for the next iteration. Note that the algorithm satisfies the integrality conditions for the decision variables and always generates a feasible solution.

The greediness of PECH_α is determined by the selection of the *greedy coefficient* α. As α increases, the rate at which the available resources are consumed by the selected item in each iteration increases. In the extreme case, $\alpha = 1$, the selected item can use its maximal effective capacity and thus maximum feasible number of copies will be included to the solution in a single iteration. For small α, the algorithm becomes conservative in its resource commitment to the selected item. The algorithm might even accept only a single copy of an item in an iteration as the resource capacities become tight. The use of the greedy coefficient reduces the possibility of creating bottleneck conditions for certain constraints, which may result in reduced rewards in future iterations. Moreover, the heuristic is likely to select different items in consecutive iterations since the maximum rewards may change over time. It is evident that PECH_α is a computationally more efficient algorithm over other general MDKP algorithms, since it depletes the effective capacity geometrically at rate α and does not need to examine the feasibility of the solution after each step.

Algorithm 1 can easily be modified to solve the 0-1 version of the MDKP, where all variables are binary. The greedy coefficient is no longer needed, as decision variables are bound to be either 0 or 1, i.e., $\alpha = 1$. Note that the increment of the selected item is always one unit and once an item is included in the solution it is immediately removed from E, the set of unselected items. As a result, PECH is terminated in at most n iterations.

We now turn our attention to the computational complexity of the PECH_α algorithm. First consider PECH_α, with $\alpha = 1$. The effect of the initialization step, STEP 1, is negligible on complexity. STEPS 2 requires $O(mn)$ basic operations and STEP 3 $O(n)$ basic operations. For $\alpha = 1$, the maximum feasible number of copies of the selected item will be included to the solution in a single iteration. Thus, the algorithm will be repeated once for each of the n items in the worst case. This implies that PECH_1 for the general MDKP and the 0-1 MDKP is a *polynomial-time* algorithm with the computational complexity $O(mn^2)$.

We next consider PECH_α, with $0 < \alpha < 1$, for the general MDKP. Similar to the case where $\alpha = 1$, each iteration requires $O(mn)$ basic operations. Let k_j be the number of iterations in which item j is selected by PECH_α, $j = 1, 2, \ldots, n$, in STEP 4. Clearly, k_j depends on the values of the problem parameters and the greedy coefficient α. Although its exact value is difficult to determine,

we recognize that k_j is bounded by the minimum of u_j and the number of times that PECH_α would have selected item j if the entire knapsack capacity were to be used for the item alone. Therefore, PECH_α is a *pseudo polynomial-time* algorithm.

We should mention that, the heuristics of Toyoda (1975), Loulou & Michaelides (1979) and Magazine & Oguz (1984) have a complexity of $O(mn^2)$, which is the same as ours. However, the heuristic of Pirkul (1987) has the potential of an exponential complexity of $O(m2^n)$, if a simplex algorithm is used to solve the LP relaxation of the problem.

3. Computational Results for the General MDKP

In this section, we present our computational results of three general MDKP algorithms, including PECH_α proposed in this paper, the primal effective gradient method outlined by Kochenberger et al. (1974) (with legend KOCH), and the dual surrogate method proposed by Pirkul & Narasimhan (1986) (with legend PIR_G). The performances of these heuristics are tested using a set of randomly generated problems with a wide range of parameter configurations. We coded the MDKP heuristics in C programming language and used the CPLEX solver to obtain optimal solutions to the random problems. We ran our tests on a 600 MHz PentiumIII personal desktop computer with 256MB of RAM. We report the following performance statistics:

1. *Mean*: This is the mean value of the percentage error, where the percentage error of each heuristic is measured as

$$\text{percentage error} = 100\% \times \frac{Z^* - Z_{\text{heuristic}}}{Z^*},$$

   where Z^* and $Z_{\text{heuristic}}$ are the optimal and heuristic solutions of the problem, respectively.

2. *CPU*: This is the percentage of the mean CPU time each heuristic takes to obtain the solution compared with the average CPU time to obtain the optimal solution.

3. *Best*: This is the percentage of problem instances that a particular heuristic provides the best solution in terms of the percentage gap with the optimal, among all competing heuristics.

We select the number of resource constraints, the number of decision variables, and the slackness of the knapsack constraints as the design parameters in our numerical tests. By varying the values of these parameters, we are able to generate a wide range of random problems. More specifically, we generate these design parameters and other problem data as follows:

- The number of resource constraints m and the number of decision variables n: We choose four levels for m, $m = 10, 50, 100$ and 200, and five levels for n, $n = 10, 50, 100, 200$ and 400;

- The greedy coefficient α: We set $\alpha = 1, 0.5$ and 0.1. This allows us to observe how the solution quality and computational time of PECH_α are affected by the greediness of the algorithm;

- We randomly generate the values of a_{ij}, c_j and u_j from discrete uniform distributions, $a_{ij} \sim Unif[0, 9]$, $c_j \sim Unif[1, 100]$ and $u_j \sim Unif[1, 200]$, for all j.

- The *slackness ratio* S_i: To generate the values of b_i with various levels of resource capacities, we adopt the *slackness ratio* S_i introduced by Zanakis (1977):

$$S_i = b_i / \sum_{j=1}^{n} a_{ij} u_j \quad \text{for } i = 1, 2, \ldots, m \tag{2}$$

  Note that resource capacities increase as the slackness ratio S_i increases. In our computations, we first generate a_{ij} and u_j as described above, and substitute them in equation (2) with S_i to determine b_i. We use iid uniform random distributions to obtain the value of S_i for each resource constraint. We study four different cases: $S_i \sim Unif(0.4, 0.6), S_i \sim Unif(0.6, 0.8), S_i \sim Unif(0.8, 0.1.0)$ and $S_i \sim Unif(0.4, 1.0)$;

This setup for the experiments gives us $4 \times 5 \times 4 = 80$ test cases. For each test case, we generate 100 instances of a_{ij}, c_j, u_j and S_i from their respective distributions. The summary of our computational results for all randomly generated problems are presented in Table 1.

Table 1: General MDKP - *Performance statistics for all problem instances*

	$\text{PECH}_{1.0}$	$\text{PECH}_{0.5}$	$\text{PECH}_{0.1}$	KOCH	PIR_G
Mean	2.67	1.13	1.24	1.52	4.64
CPU	1.23	1.92	3.29	3.89	4.52
Best	0.00	52.50	8.70	11.30	27.50

In terms of the average of the mean errors, the rank order of the performances of these heuristics, from the best to the worst, is $\text{PECH}_{0.5}$, $\text{PECH}_{0.1}$, PECH_1, KOCH, and PIR_G, with the average of the mean errors of the best performer, $\text{PECH}_{0.5}$, being only 1.13%, less than a quarter of that under PIR_G. In terms of the average of CPUs, the rank order of the performances of these heuristics, from the fastest to the slowest, is PECH_1, $\text{PECH}_{0.5}$, $\text{PECH}_{0.1}$,KOCH, and PIR_G, with the average of CPUs of the fastest heuristic, PECH_1, being only 1.23%. In terms of the best solution over 8,000 problem instances, the rank order of the performances of these heuristics, from the best to

the worst, is $\text{PECH}_{0.5}$, PIR_G, KOCH, $\text{PECH}_{0.1}$, and PECH_1. It is interesting to observe that the solution quality of PECH_α does not necessarily improve as α decreases; rather, it suggests that PECH_α with a moderate value of α provides an effective tradeoff between solution quality and computational time. It is also worth noting that although PIR_G fails to match the accuracy and speed of $\text{PECH}_{0.5}$ on average, its solution quality dominates those of all other heuristics for low-dimensional knapsack problems.

We use Tables 2, 3 and 4 to observe the performance statistics of the heuristics with varying problem parameters, namely the number of resource constraints, the number of decision variables and the slackness ratios of resource constraints. We construct these tables by simply taking the averages of the performance data with respect to the fixed m, n or S_i, over the values of other parameters.

Table 2: General MDKP - *Performance statistics for different number of resource constraints*

	$\text{PECH}_{1.0}$		$\text{PECH}_{0.5}$		$\text{PECH}_{0.1}$		KOCH		PIR_G	
m	Mean	CPU	Mean	CPU	Mean	CPU	Mean	CPU	Mean	CPU
10	1.44	0.18	0.78	0.30	0.88	0.49	0.56	0.56	1.64	0.54
50	2.58	0.68	1.18	1.14	1.32	1.93	1.34	2.28	4.28	2.53
100	2.94	1.24	1.25	2.02	1.36	3.63	1.79	4.23	5.70	4.82
200	3.70	2.83	1.31	4.23	1.39	7.11	2.39	8.50	6.93	10.18

Table 3: General MDKP - *Performance statistics for different number of decision variables*

	$\text{PECH}_{1.0}$		$\text{PECH}_{0.5}$		$\text{PECH}_{0.1}$		KOCH		PIR_G	
n	Mean	CPU	Mean	CPU	Mean	CPU	Mean	CPU	Mean	CPU
10	8.20	0.06	1.67	0.09	1.74	0.17	3.65	0.16	17.25	0.05
50	1.77	0.16	1.08	0.25	1.21	0.42	1.40	0.44	3.05	0.28
100	1.34	0.40	1.04	0.64	1.19	1.04	1.11	1.25	1.66	1.07
200	1.10	1.39	0.97	1.97	1.08	3.42	0.84	4.04	0.86	4.26
400	0.92	4.15	0.87	6.65	0.97	11.41	0.60	13.56	0.38	16.93

Table 4: General MDKP - *Performance statistics for different slackness ratios*

	$\text{PECH}_{1.0}$		$\text{PECH}_{0.5}$		$\text{PECH}_{0.1}$		KOCH		PIR_G	
S_i	Mean	CPU	Mean	CPU	Mean	CPU	Mean	CPU	Mean	CPU
(0.4-0.6)	4.24	1.82	1.84	2.86	1.98	4.63	2.27	2.33	7.08	2.63
(0.6-0.8)	2.21	0.94	0.83	1.72	0.89	2.82	1.28	3.21	4.32	3.79
(0.8-1.0)	1.03	0.72	0.33	1.11	0.34	1.95	0.64	5.54	2.23	6.72
(0.4-1.0)	3.18	1.44	1.51	2.00	1.73	3.76	1.88	4.49	4.92	4.94

We note from Table 2 that the performance of all heuristics deteriorates as the number of resource constraints increases, both in terms of *Mean* and *CPU*. However, this trend is more striking for PIR_G. This is to be expected, as it is particularly designed to solve the general MDKP with a small number of constraints (Pirkul & Narasimhan 1986). Note also that KOCH provides the best solution for the problems with a small number of constraints.

Table 3 shows that the solution quality of the heuristics increases as the number of decision variables increases, but the CPU time also increases. Observe that, in terms of the mean error, $\text{PECH}_{0.5}$ outperforms all other heuristics for the instances with $n = 10,\ 50,\ 100$, but underperforms KOCH and PIR_G for the instances with $n = 200,\ 400$. A plausible explanation of this result is that the batch assignment feature of PECH can exhaust the knapsack's capacity prematurely when the number of decision variables are large. We also notice that the performance of KOCH is relatively robust, whereas that of PIR_G is very sensitive to the number of decision variables. In terms of the CPU time, $\text{PECH}_{0.5}$ dominates both KOCH and PIR_G, particularly for the problem with a large number of decision variables.

As seen in Table 4, when the expected value of the slackness of the constraints increases, with the variance fixed (compare the first three rows of Table 4), the solution quality of all heuristics improves. Notice that the performances of both PECH and KOCH are relatively robust against the tightness of slackness ratios, but PIR_G is not. It is particularly striking to observe that the CPU time of our heuristics decreases as the expected value of the slackness of the constraints increases, whereas those of KOCH and PIR_G worsen. We believe that this phenomenon can be explained by the unique capability of PECH to deplete its remaining capacity at a geometric rate α. When the knapsack has ample slacks, the batch assignment feature of PECH shows its power, as the other two heuristics can only add a single copy of the selected item to the solution in each iteration. Finally, when the variance of the slackness of the constraints increases, with the mean fixed (compare the second and the fourth rows of Table 4), both performance measures suffer in all heuristics.

In Table 5, we display the heuristics with the minimum mean error for various problem sizes in terms of the number of constraints and number of decision variables, averaged over a wide range of slackness ratios. As noted earlier, PECH dominates the other heuristics when the number of variables is relatively smaller compared to the number of constraints. Therefore, it is desirable to use our heuristic for general knapsack problems with a large number of dimensions.

Table 5: General MDKP - *Best heuristic (in terms of Mean) for different number of constraints and variables*

m \ n	10	50	100	200	400
10	$PECH_{0.5}$	KOCH	KOCH	PIR_G	PIR_G
50	$PECH_{0.5}$	$PECH_{0.5}$	$PECH_{0.5}$	PIR_G	PIR_G
100	$PECH_{0.5}$	$PECH_{0.5}$	$PECH_{0.5}$	PIR_G	PIR_G
200	$PECH_{0.5}$	$PECH_{0.5}$	$PECH_{0.5}$	$PECH_{0.5}$	PIR_G

4. Computational Results for the 0-1 MDKP

This section presents our computational results for the test cases of the 0-1 MDKP. We use randomly generated problems and some benchmark problems from the related literature to test the performance of PECH against the approximate solution methods of Senju & Toyoda (1968), Toyoda (1975), Loulou & Michaelides (1979), Magazine & Oguz (1984), and Pirkul (1987), Volgenant & Zoon (1990) and genetic algorithm of Chu & Beasley (1998). The legends used in the rest of the tables for these heuristics are self-explanatory.

4.1 Randomly Generated Test Problems

We choose the system configuration similarly as in Section 3 for the general MDKP, except that we let $u_j \equiv 1$ for all j. Table 6 reports the summary of the performance statistics over 80 different test cases, with 100 random instances for each case. Overall, PECH ranks the best among all other heuristics in terms of the averages of *Mean* error, followed by SEN and PIR. The computational times of PECH, SEN, TOY and MAG appear competitive. Also note that PECH and PIR provide the best solutions for these cases.

Table 6: 0-1 MDKP - *Performance statistics for all problem instances*

	PECH	SEN	TOY	LOU	MAG	PIR
Mean	1.97	3.15	3.46	3.48	4.50	3.25
CPU	0.78	0.82	0.90	1.83	1.13	3.25
Best	47.50	0.00	0.00	0.00	0.00	52.5

Table 7 shows that PIR provides the minimal mean error when m is small, whereas PECH becomes dominant when m increases. On the other hand, Table 8 indicates that PECH provides the minimal mean error when n is small, but gradually yields its leading role to PIR (when $n \geq 100$), SEN (when $n \geq 200$) and MAG (when $n = 400$) as n increase. It is worth noting that even though the performance of PECH as compared with that of several others deteriorates for large n, it still

provides excellent approximate solutions. For example, when $n = 400$, the mean errors of the best heuristic, PIR, and PECH are 0.32% and 0.78%, respectively. Table 9 shows that PECH outperforms all other heuristics both in terms of solution quality and CPU against all levels of the slackness ratio. The robustness of PECH is particularly striking since it is primarily designed to solve the general MDKP.

Table 7: 0-1 MDKP - *Performance statistics for different number of resource constraints*

	PECH		SEN		TOY		LOU		MAG		PIR	
m	Mean	CPU	Mean	CPU	Mean	CPU	Mean	CPU	Mean	CPU	Mean	CPU
10	0.92	0.12	1.50	0.12	2.02	0.13	2.42	0.23	2.36	0.15	0.86	0.39
50	1.23	0.45	2.29	0.47	2.53	0.52	2.51	1.03	4.08	0.64	1.84	1.83
100	1.43	0.86	2.60	0.91	2.84	0.99	2.73	2.03	5.30	1.25	3.04	3.61
200	4.28	1.70	6.21	1.79	6.47	1.98	6.27	4.03	6.25	2.48	7.27	7.19

Table 8: 0-1 MDKP - *Performance statistics for different number of decision variables*

	PECH		SEN		TOY		LOU		MAG		PIR	
n	Mean	CPU	Mean	CPU	Mean	CPU	Mean	CPU	Mean	CPU	Mean	CPU
10	5.49	0.04	10.39	0.04	8.03	0.04	8.55	0.04	16.02	0.03	12.17	0.04
50	1.56	0.10	2.48	0.10	2.83	0.10	3.08	0.18	3.05	0.13	2.15	0.22
100	1.11	0.25	1.46	0.26	2.31	0.29	2.27	0.57	1.72	0.35	1.07	0.79
200	0.89	0.81	0.86	0.84	2.13	0.94	1.86	1.88	1.04	1.16	0.55	3.06
400	0.78	2.71	0.57	2.87	2.02	3.16	1.65	6.48	0.67	3.98	0.32	12.16

Table 9: 0-1 MDKP - *Performance statistics for different slackness ratios*

	PECH		SEN		TOY		LOU		MAG		PIR	
S_i	Mean	CPU	Mean	CPU	Mean	CPU	Mean	CPU	Mean	CPU	Mean	CPU
(0.4-0.6)	3.28	1.10	4.88	0.48	5.38	0.53	5.44	1.06	7.13	0.66	5.34	1.87
(0.6-0.8)	1.48	0.67	2.58	0.71	2.67	0.78	2.70	1.57	3.81	0.97	2.86	2.79
(0.8-1.0)	0.62	0.47	1.32	1.16	1.15	1.28	1.17	2.60	1.98	1.60	1.44	4.64
(0.4-1.0)	2.48	0.89	3.83	0.93	4.66	1.03	4.62	2.09	5.07	1.29	3.38	3.71

Table 10 presents the heuristics with the minimum mean error for various problem sizes in terms of the number of constraints and number of decision variables. Clearly, PECH outperforms the other heuristics when the number of constraints are relatively larger than the number of variables, which can be observed in the lower triangle region of the table.

4.2 Benchmark Problems from the Literature

We solve two capital budgeting problems taken from Weingartner & Ness (1967), two others from Petersen (1967) and two resource planning problems from Senju & Toyoda (1968). These six problems have been used extensively as the benchmark problems in the existing literature to test the

Table 10: 0-1 MDKP - *Best heuristic (in terms of Mean) for different number of constraints and variables*

m \ n	10	50	100	200	400
10	PECH	PIR	PIR	PIR	PIR
50	PECH	PECH	PIR	PIR	PIR
100	PECH	PECH	PECH	PIR	PIR
200	PECH	PECH	PECH	PECH	PIR

effectiveness of proposed heuristics. We report the optimal solution and the heuristic solutions for each problem in Table 11. The results for SEN, TOY, LOU and MAG are taken from their respective references, and the results for PECH and PIR are based on our own computations.

Table 11: Optimal and heuristic solutions of six 0-1 MDKP from literature

Problem	m	n	Z^*	Z_{PECH}	Z_{SEN}	Z_{TOY}	Z_{LOU}	Z_{MAG}	Z_{PIR}
Weingartner & Ness (1967)									
1	2	28	141,278	140,618	138,888	139,278	135,673	139,418	140,477
2	2	105	1,095,445	1,094,262	1,093,181	1,088,365	1,083,086	1,092,971	1,094,757
Petersen (1967)									
1	5	39	10,618	10,480	9,548	10,320	9,290	10,354	10,547
2	5	50	16,537	16,463	16,237	15,897	14,553	16,261	16,436
Senju & Toyoda (1968)									
1	30	60	7,772	7,700	7,590	7,719	7,675	7,719	7,728
2	30	60	8,722	8,666	8,638	8,709	8,563	8,623	8,697

Table 11 indicates that the maximum deviation of the solution by PECH from the optimal solution is only 1.29%. In the remaining five problems, the deviation is much smaller then 1%. PECH provides the best solution in two of these problems and the second best solution in another two. The closest competitor of PECH is PIR, which requires to solve a linear program and hence needs a larger CPU. Therefore, we conclude that our heuristic manages these benchmark problems aptly.

Finally we test the performance of our heuristic against the algorithms of Magazine & Oguz (1984), Volgenant & Zoon (1990), Pirkul (1987) and the genetic algorithm of Chu & Beasley (1998) in 270 problem instances described in Chu & Beasley (1998). Note that genetic algorithms are *advanced* probabilistic search algorithms and require *significant* computational effort (for the problem instances solved here, Chu & Beasley (1998) report 1267.4 CPU seconds as an average solution time, whereas a greedy algorithm provides a solution almost instantaneously). We should also mention that these problems have relatively small knapsack dimensions and large number of variables. The average percentage optimality gap for these problems are provided in Table 12.

Clearly, the genetic algorithm of Chu & Beasley (1998) dominates all heuristic methods. However

Table 12: 0-1 MDKP - *Performance Comparison with a Genetic Algorithm*

Problem			Mean				
m	n	S_i	MAG	VZ	PIR	PECH	GA
5	100	0.25	13.69	10.30	1.59	7.34	0.99
		0.50	6.71	6.90	0.77	3.47	0.45
		0.75	5.11	5.68	0.48	2.02	0.32
	Average		8.50	7.63	0.95	4.28	0.59
5	250	0.25	6.64	5.85	0.53	7.12	0.23
		0.50	5.22	4.40	0.24	3.20	0.12
		0.75	3.56	3.59	0.16	1.77	0.08
	Average		5.14	4.61	0.31	4.03	0.14
5	500	0.25	4.93	4.11	0.22	6.41	0.09
		0.50	2.96	2.53	0.08	3.43	0.04
		0.75	2.31	2.41	0.06	1.70	0.03
	Average		3.40	3.02	0.12	3.85	0.05
10	100	0.25	15.88	15.55	3.43	8.20	1.56
		0.50	10.41	10.72	1.84	3.73	0.79
		0.75	6.07	5.67	1.06	1.82	0.48
	Average		10.79	10.65	2.11	4.58	0.94
10	250	0.25	11.73	10.53	1.07	5.86	0.51
		0.50	6.83	5.92	0.57	2.56	0.25
		0.75	4.42	3.77	0.33	1.52	0.15
	Average		7.66	6.74	0.66	3.31	0.30
10	500	0.25	8.81	7.90	0.52	5.06	0.24
		0.50	5.71	4.14	0.22	2.45	0.11
		0.75	3.62	2.93	0.14	1.23	0.07
	Average		6.05	4.99	0.29	2.92	0.14
30	100	0.25	17.39	17.21	9.02	6.83	2.91
		0.50	11.82	10.19	3.51	3.19	1.34
		0.75	6.58	5.92	2.03	1.91	0.83
	Average		11.93	11.11	4.85	3.98	1.69
30	250	0.25	13.54	12.41	3.70	4.83	1.19
		0.50	8.64	7.12	1.53	2.08	0.53
		0.75	4.49	3.91	0.84	1.16	0.31
	Average		8.89	7.81	2.02	2.69	0.68
30	500	0.25	9.84	9.62	1.89	3.69	0.61
		0.50	7.10	5.71	0.73	1.70	0.26
		0.75	3.72	3.51	0.48	0.88	0.17
	Average		6.89	6.28	1.03	2.09	0.35
Average			7.69	6.98	1.37	3.46	0.54

we should also state that the genetic algorithm cannot be considered a direct competitor of the other heuristics since its running time is very high. Further, for problems with small knapsack dimensions and large number of variables, we had already pointed out that PIR would outperform other heuristics in Section 4.1. From the above table, we see that for a given number of variables, the performance of PIR deteriorates as the dimension of the knapsack increases. Hence, at some point we should expect that PECH would outperform PIR, as the performance of PECH improves as the knapsack dimensions increase. Finally PECH significantly dominates the heuristics of Magazine & Oguz (1984) and Volgenant & Zoon (1990). It is worth mentioning that the performance of PECH compared to these more complicated heuristics is astonishing because it is extremely simple and fast, and is mainly tailored for the general MDKP, not for the 0-1 MDKP.

5. Conclusion

In this paper, we propose a heuristic to approximately solve the multidimensional knapsack problem. The intuition behind our heuristic is the following: Instead of using a criterion based on each item's aggregate consumption of the resources, our heuristic uses the notion of effective capacity, defined as the maximum number of copies of an item that can be accepted if the entire knapsack were to be used for that item alone. Furthermore, our heuristic includes the selected items in the solution in batches, which consequently improves computational effort. Although our heuristic is primarily design for the general MDKP, it also solves the 0-1 version of the problem effectively. The numerical results on a large number of randomly generated problem instances for both the general MDKP and 0-1 MDKP demonstrate the superiority/competitiveness of our heuristics against other existing ones. Other computations based on the benchmark problems from the literature support the strength of our heuristic and its trustworthy for real-time implementations.

References

Akcay, Y. & Xu, S. H. (2004), 'Joint inventory replenishment and component allocation optimization in an assemble-to-order system', *Management Science* **50**, 99–116.

Caprara, A., Kellerer, H., Pferschy, U. & Pisinger, D. (2000), 'Approximation algorithms for knapsack problems with cardinality constraints', *European Journal of Operational Research* **123**, 333–345.

Chu, P. C. & Beasley, J. E. (1998), 'A genetic algorithm for the multidimensional knapsack problem', *Journal of Heuristics* **4**, 63–86.

de Vries, S. & Vokra, R. V. (2001), Combinatorial auctions: A survey, Discussion Paper 1296, The Center for Mathematical Studies in Economics and Management Science, Northwestern University.

Everett, H. (1963), 'Generalized langrange multiplier method for solving problems of optimum allocation of resources', *Operations Research* **2**, 399–417.

Garey, M. R. & Johnson, D. S. (1979), *Computers and Intractability : A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco.

Kleywegt, A. & Papastavrou, J. D. (2001), 'The dynamic and stochastic knapsack problem with random sized items', *Operations Research* **49**, 26–41.

Kochenberger, G. A., McCarl, B. A. & Wyman, F. P. (1974), 'A heuristic for general integer programming', *Decision Sciences* **5**, 36–44.

Lin, E. Y.-H. (1998), 'A bibliographical survey on some well-known non-standard knapsack problems', *INFOR* **36**(4), 274–317.

Loulou, R. & Michaelides, E. (1979), 'New greedy-like heuristics for the multidimensional 0-1 knapsack problem', *Operations Research* **27**, 1101–1114.

Lu, L. L., Chiu, S. Y. & Cox, L. A. (1999), 'Optimal project selection: Stochastic knapsack with finite time horizon', *Operations Research* **50**, 645–650.

Magazine, M. J. & Oguz, O. (1984), 'A heuristic algorithm for the multidimensional zero-one knapsack problem', *European Journal of Operational Research* **16**, 319–326.

Petersen, C. C. (1967), 'Computational experience with variants of the balas algorithm applied to the selection of R&D projects', *Management Science* **13**, 736–750.

Pirkul, H. (1987), 'A heuristic solution procedure for the multiconstraint zero-one knapsack problem', *Naval Research Logistics* **34**, 161–172.

Pirkul, H. & Narasimhan, S. (1986), 'Efficient algorithms for the multiconstraint general knapsack problem', *IIE Transactions* pp. 195–203.

Senju, S. & Toyoda, Y. (1968), 'An approach to linear programming with 0-1 variables', *Management Science* **15**(4), B196–B207.

Shih, W. (1979), 'A branch and bound method for the multiconstraint zero-one knapsack problem', *Journal of Operational Research Society* **30**, 369–378.

Toyoda, Y. (1975), 'A simplified algorithm for obtaining approximate solutions to zero-one programming problems', *Management Science* **21**(12), 1417–1427.

Volgenant, A. & Zoon, J. A. (1990), 'An improved heuristic for multidimensional 0-1 knapsack problems', *Journal of Operational Research Society* **41**, 963–970.

Weingartner, H. M. & Ness, D. N. (1967), 'Methods for the solution of the multi-dimensional 0/1 knapsack problem', *Operations Research* **15**, 83–103.

Zanakis, S. H. (1977), 'Heuristic 0-1 linear programming: Comparisons of three methods', *Management Science* **24**, 91–103.