An Approximate Dynamic Programming Approach to Multi-dimensional Knapsack Problems

Dimitris Bertsimas* Ramazan Demir†

August, 2000, revised November 2001, January, 2002

Abstract

We present an Approximate Dynamic Programming (ADP) approach for the multi-dimensional knapsack problem (MKP). We approximate the value function (a) using parametric and nonparametric methods and (b) using a base-heuristic. We propose a new heuristic that rounds adaptively the solution of the linear programming relaxation. Our computational study suggests: (a) the new heuristic produces high quality solutions fast and robustly; (b) state of the art commercial packages like CPLEX require significantly larger computational time to achieve the same quality of solutions; (c) the ADP approach using the new heuristic competes successfully with alternative heuristic methods such as genetic algorithms; (d) the ADP approach based on parametric and nonparametric approximations, while producing reasonable solutions, is not competitive. Overall, this research illustrates that the base-heuristic approach is a promising computational approach for MKPs worthy of further investigation.

*Boeing Professor of Operations Research, Sloan School of Management and Operations Research Center, Massachusetts Institute of Technology, E53-363, Cambridge, MA 02139, dbertsim@mit.edu. The research of this author was partially supported by NSF grant DMI-9610486, and the MIT-Singapore alliance.

†Operations Research Center, Massachusetts Institute of Technology Cambridge, MA 02139, rdemir@alum.mit.edu.

1 Introduction

We consider the multi-dimensional knapsack problem (MKP) defined as follows. A set $N = \{1, \ldots, n\}$ of items should be packed in a set $M = \{1, \ldots, m\}$ of knapsacks with given capacities, $b_{0,i}, i \in M$. Associated with every item $j \in N$ there is a value c_j and a weight a_{ij}, which is the amount of resource used by the item j in the i-th knapsack. The goal is to find a subset of the items that yields the maximum value subject to the capacity constraints of the knapsacks.

The MKP arises in many real-world applications such as combinatorial auctions (see de Vries and Vohra [13], Rothkopf et. al. [44]), computer systems design (see Ferreira et. al. [17]), project selection (see Peterson [41]), cutting stock and cargo-loading (see Gilmore and Gomory [23]), capital budgeting (see Weingartner [54]), and asset-backed securitization (see Mansini and Speranza [36]). The set cover and multi-cover (see Hochbaum [27]) problems can be reformulated as the MKP through variable complementing (see Section 2 for an illustration). In addition to these applications, many complex problems can be transformed to MKPs by some relaxation methodologies. For instance, the binary knapsack problem, a special case of the MKP, is used as a subproblem when solving the Generalized Assignment Problem, as well as the Vehicle Routing Problem (see Laporte [30]). Due to its practical and theoretical importance, both exact and approximate solution approaches have been applied to tackle the multi-dimensional knapsack problem (see Section 2 for references).

Optimization problems can be reformulated and solved by dynamic programming (DP). This approach is usually impractical for large-scale problems due to large number of value function calculation and state storage requirements, known as “ the curse of dimensionality.” Motivated by the need to address the curse of dimensionality several researchers have proposed approximations in dynamic programming. The collection of such methods is broadly known as Approximate Dynamic Programming (ADP). Many ingenious approaches such as Lagrange multiplier, successive approximation, function approximation (e.g., neural networks, radial basis representation, polynomial representation) methods have been proposed to break the curse of dimensionality while contributing diverse approximate dynamic programming methodologies to the literature (see for example Bellman [2], Morin [39], and Cooper and Cooper [12]). Functional approximations utilizing neural networks ideas were used in Kleywegt et. al. [29] for inventory routing problems, and in Van Roy et. al. [45] for inventory management problems. Powell and Shapiro [43] introduced a dynamic programming reformulation for dynamic resource allocation for fleet management problems and used ADP methodologies to generate high quality solutions. Bertsekas and Tsitsiklis [4] develop a framework of approximate dynamic programming primarily in the context of stochastic optimization and include several case studies in various fields. Sarkar et. al. [47] used greedy or approximate algorithms for machine scheduling problems in a look-ahead search mechanism in order to alleviate the performance of the stand-alone greedy or approximate methodology while Bertsimas et. al. [8] applied a similar idea for location problems. Other applications utilizing ADP methods are two-stage maintenance and repair problem in Bertsekas et. al. [5], stochastic scheduling problems in Bertsekas and Castanon [3], sequencing and stochastic vehicle routing in Secomandi [48], railroad scheduling in Christodouleas [10], deterministic supply chain problems in Wike [56] and revenue management in Bertsimas and Popescu [7].

Our objective in this paper is to examine whether ADP methods represent a realistic alternative for large scale MKPs. The computational evidence so far regarding the success of ADP methods suggests that such methods have been more successful in stochastic optimization problems. Although there has been positive evidence for deterministic problems as well (see the literature review above), we want to examine ADP methods with respect to speed, quality of solutions and robustness in the context of a reasonably rich class of integer programs like multidimensional knapsack problems.

1. In Section 2, we provide a dynamic programming reformulation of the MKP and introduce the relevant ADP concepts.

2. In Section 3, we describe an ADP algorithm under which we approximate the optimal value function by a base-heuristic. The computational evidence suggests that the ADP base-heuristic approach improves the performance of a base-heuristic and often the improvement is significant. We also design an adaptive-fixing heuristic for the MKP. Although this is not the main focus of the paper, the heuristic appears to be computationally attractive and generates high quality solutions compared to some heuristics from the literature.

3. In Section 4, we develop both parametric and nonparametric approximations of the value function for the MKP.

4. In Section 5, we report extensive computational results and comparisons. The evidence suggests that the base-heuristic approach with the adaptive fixing heuristic is the most promising methodology within the ADP family for the MKP and competes successfully with existing state of the art heuristic methodologies as well as with a state of the art commercial packages like CPLEX [14].

2 The MKP and ADP

An integer programming formulation of the MKP is as follows:

$$\begin{aligned} \text{maximize} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \le b_{0,i}, \quad i = 1, \ldots, m, \\ & x_j \in \{0,1\}, \qquad j = 1, \ldots, n. \end{aligned} \tag{1}$$

with $a_{ij}, b_{0,i}, c_j \ge 0$ for $i = 1, \ldots, m$ and $j = 1, \ldots, n$. We denote by $\mathbf{x}'$ the transpose of a vector $\mathbf{x}$. Let $\mathbf{A}_j = (a_{1j}, \ldots, a_{mj})'$, $\mathbf{b}_0 = (b_{0,1}, \ldots, b_{0,m})'$ and $\mathbf{c} = (c_1, \ldots, c_n)'$. Then the problem in Eq. (1) can be written as follows:

$$MKP(n, \mathbf{b}_0) \qquad \begin{aligned} \text{maximize} \quad & \mathbf{c}'\mathbf{x} \\ \text{subject to} \quad & \sum_{j=1}^{n} \mathbf{A}_j x_j \le \mathbf{b}_0 \\ & x_j \in \{0,1\}, \qquad j = 1, \ldots, n. \end{aligned} \tag{2}$$

The case for $m = 1$ is the binary knapsack problem (BKP) which has been extensively studied (see Martello and Toth [37]). For the MKP no pseudo-polynomial algorithm can exist unless P=NP, since the MKP is NP-hard in the strong sense (see Martello and Toth [37] and Magazine and Chern [34]). For the binary knapsack problems both polynomial time (Sahni [46]) and fully polynomial time approximation schemes (Ibarra and Kim [28]) have been developed. The set packing problem is a special case of the MKP with $a_{ij} \in \{0,1\}$ and $b_{0,i} = 1$ (see Chapter 3 of Hochbaum [27]). Moreover, the set cover problem (see Chapter 3 of Hochbaum [27]),

$$\text{minimize}\{\mathbf{c}'\mathbf{x} \mid \mathbf{A}\mathbf{x} \ge \mathbf{e}, x_j \in \{0,1\}\},$$

where $\mathbf{e}$ is a column vector of ones, $\mathbf{A} = [\mathbf{A}_1, \ldots, \mathbf{A}_n]$ is an $m \times n$ 0-1 matrix, and $\mathbf{x} = (x_1, \ldots, x_n)'$, can be reformulated as a MKP by substituting x_j by $y_j = 1 - x_j$ for $j = 1, \ldots, n$. The set cover problem is equivalent to the following MKP problem

$$\mathbf{c}'\mathbf{e} + \text{maximize}\{\mathbf{c}'\mathbf{y} \mid \mathbf{A}\mathbf{y} \le \mathbf{b}, y_j \in \{0,1\}\},$$

where $\mathbf{b} = \sum_j \mathbf{A}_j - \mathbf{e}$ and $\mathbf{y} = (y_1, \ldots, y_n)'$.

Branch-and-bound and dynamic programming techniques have been applied to solve the MKP to optimality but with limited success for large instances (see Thesen [52], Shih [50], Weingartner and Ness [55]). Since feasible solutions are valuable for large-scale practical problems, there has been a sizeable literature on heuristic methods for MKPs . Such methods can be classified as follows:

1. Greedy methods (Senju and Toyoda [49], Toyoda [53], Loulou and Michaelides [33], Lee and Guignard [31], Kochenberger et. al. [21], Magazine and Oguz [35]).

2. Aggregation methods (Freville and Plateau [18, 19], Glover [24], Gavish and Pirkul [22]).

3. Tabu search methods (Hanafi and Freville [26], Glover and Kochenberger [25], Aboudi and Jornsten [1], and Løkketangen and Glover [32]).

4. Genetic algorithms (Chu and Beasley [11]).

Computational experience reported in the literature suggests that genetic algorithms seem to produce high quality solutions for large scale problems. For this reason, in some of our computational results we compare the ADP approach with genetic algorithms.

Dynamic Programming Formulation

In order to reformulate the $MKP(n, \mathbf{b}_0)$ in Eq. (2) as a dynamic program we consider the subproblem $MKP(k, \mathbf{b})$ which includes the *first* k variables with the right-hand side $\mathbf{b}$. Let $\mathbf{x}^{OPT}(k, \mathbf{b}) = (x_1^*, \ldots, x_k^*)$ and $F(k, \mathbf{b})$ be an optimal solution and the optimal value for the subproblem $MKP(k, \mathbf{b})$, respectively. We set $F(k, \mathbf{b}) = -\infty$ if the subproblem $MKP(k, \mathbf{b})$ is infeasible. In the optimal solution to $MKP(k, \mathbf{b})$ the x_k is either set to zero or one. If we set x_k to zero, the objective value equals $F(k-1, \mathbf{b})$, the optimal value of the subproblem $MKP(k-1, \mathbf{b})$. If we set x_k to one, the objective value equals $F(k-1, \mathbf{b} - \mathbf{A}_k) + c_k$, the optimal value of the subproblem $MKP(k-1, \mathbf{b}-\mathbf{A}_k)$ plus c_k. Taking the maximum of the preceding objective values gives the optimal value $F(k, \mathbf{b})$ to the subproblem $MKP(k, \mathbf{b})$. Thus, the dynamic programming recursion can be stated as

$$F(k, \mathbf{b}) = \max \{F(k-1, \mathbf{b}), F(k-1, \mathbf{b} - \mathbf{A}_k) + c_k\} \tag{3}$$

for $k = 2, \ldots, n$, with an initial condition $F(1, \mathbf{b})$. Starting with $x^{OPT}(1, \mathbf{b})$ and using

$$x_k^* = \text{argmax}_{x \in \{0,1\}} \{F(k-1, \mathbf{b} - \mathbf{A}_k x) + c_k x\} \tag{4}$$

we calculate $\mathbf{x}^{OPT}(k, \mathbf{b}) = (\mathbf{x}^{OPT}(k-1, \mathbf{b} - \mathbf{A}x_k^*), x_k^*)$ for $k = 2, \ldots, n$. It is easy to see that exact dynamic programming requires $O(n(b^*)^m)$ calculations for $MKP(n, \mathbf{b}_0)$ where $b^* = \max \{b_{0,1}, \ldots, b_{0,m}\}$. Space requirements and value function computations become impractical for even moderate m. Thus, exact dynamic programming is not a practical methodology particularly for large-scale optimization problems. The basic idea behind approximate dynamic programming is to approximate the optimal value function $F(k, \mathbf{b})$ and to construct a suboptimal solution using Equations (3) and (4). In the next two sections, we approximate the optimal value function using (a) a base-heuristic and (b) parametric and nonparametric methods.

3 The ADP-Base Heuristic Approach

This section introduces the ADP-Base Heuristic (ADP-BH) approach for the MKP. The basic idea of ADP-BH is estimating the optimal value function $F(k, \mathbf{b})$ by the solution value of a suboptimal methodology to the corresponding subproblem $MKP(k, \mathbf{b})$ and constructing a solution through Eq. (4). This suboptimal methodology is called *base-heuristic* approach. We also propose a new heuristic that constructs solutions by adaptively fixing variables through solving linear programming relaxations iteratively, which we name *adaptive fixing heuristic*. Let $BH(k, \mathbf{b})$ be a base-heuristic for the subproblem $MKP(k, \mathbf{b})$. Let $\mathbf{x}^{BH}(k, \mathbf{b})$ be the corresponding heuristic solution and $H(k, \mathbf{b})$ be the heuristic value, i.e., an estimate of the optimal value $F(k, \mathbf{b})$.

The ADP-BH algorithm starts by applying $BH(n, \mathbf{b}_0)$ to the problem $MKP(n, \mathbf{b}_0)$ and getting $\mathbf{x}^{BH}(n, \mathbf{b}_0)$. If the solution $\mathbf{x}^{BH}(n, \mathbf{b}_0)$ is optimal the algorithm terminates with an optimal solution. If the problem is infeasible the algorithm terminates without a solution. Otherwise, the algorithm sets the variables *best-solution* $\mathbf{x}^{BEST} = \mathbf{x}^{BH}(n, \mathbf{b}_0)$ and *best-solution-value* $z^{BEST} = H(n, \mathbf{b}_0)$ (Step 2 in Figure 1). The algorithm proceeds by applying *reduced cost fixing* as described below to fix some of the variables to the corresponding values (0 or 1) in an *optimal* solution to the problem $MKP(n, \mathbf{b}_0)$ (Step 3 in Figure 1). Reduced cost fixing might effectively reduce the number of variables in advance. Let F denote the set of indices of the fixable variables by the reduced cost criterion. We denote by $\mathbf{x}^F$ the corresponding fixed values, that is, $x_j^F = 0$ or 1, for all $j \in F$.

The algorithm iteratively sets the variables to 0 or 1 as described in Step 4 of Figure 1. At each iteration, we update the *best-solution*, $\mathbf{x}^{BEST}$ and *best-solution-value*, z^{BEST}. We check if *early termination* is possible (true) or not (false). We also allow fixing a set of variables, which we name *lag-variable-fixing*. At the final step, we set the first variable to the optimal solution of the reduced problem (Step 5 of Figure 1) and the algorithm returns $\mathbf{x}^{BEST}$ and z^{BEST}. We provide the details of ADP-Base Heuristic algorithm in Figure 1.

Reduced-cost fixing:

At optimality of the linear programming relaxation (LP), $LP(n, \mathbf{b}_0)$, of the problem MKP $(n, \mathbf{b}_0)$, the objective function can be written as

$$z^{LP(n,\mathbf{b}_0)} = z_0 + \sum_{j \in L} \overline{c}_j x_j + \sum_{j \in U} \overline{c}_j x_j,$$

where $\mathbf{x}^{LP(n,\mathbf{b}_0)}$ and $z^{LP(n,\mathbf{b}_0)}$ are the LP solution and value, respectively, L denotes the set of non-basic variables at their lower bound, U denotes the set of non-basic variables at their upper bounds and $\overline{c}_j$ denotes the reduced cost of the variable $j \in \{1, \ldots, n\}$. Reduced cost fixing sets the variable x_k at its optimal LP solution value if $|\overline{c}_k| > z^{LP(n,\mathbf{b}_0)} - z^C$ where z^C is a lower bound to the problem $MKP(n, \mathbf{b}_0)$(Nemhauser and Wolsey [40], p. 389). In our algorithm, we use $H(n, \mathbf{b}_0)$ as z^C. Thus, $F = \{j : |\overline{c}_j| > z^{LP(n,\mathbf{b}_0)} - H(n, \mathbf{b}_0)\}$ where $x_j^F = x_j^{LP(n,\mathbf{b}_0)}$ for all $j \in F$.

1: Initialization:

$k = n, \ \mathbf{b} = \mathbf{b}_0, \ F = \emptyset$

$early - termination = false$

2: Apply $BH(k, \mathbf{b})$ and get $\mathbf{x}^{BH}(k, \mathbf{b}), \ H(k, \mathbf{b})$

Problem infeasible $\rightarrow$ exit, the problem is infeasible

$\mathbf{x}^{BH}(k, \mathbf{b})$ optimal $\rightarrow$ exit, an optimal solution available

$\mathbf{x}^{BEST} \leftarrow x^{BH}(k, \mathbf{b}), \ z^{BEST} \leftarrow H(k, \mathbf{b})$

3: Apply *reduced cost fixing* and get $F, \mathbf{x}^F$

4: While $(k > 2)$ and $(early - termination = false)$ do

if $(k \in F)$ then

$x_k^{ADP} \leftarrow x_k^F$

else

Calculate x_k^{ADP} by calling *variable assignment*

Update *best-solution* and *best-solution-value* and get $\mathbf{x}^{BEST}, z^{BEST}$

Check *early termination* criterion and get the status (true or false)

Fix variables by *lag variable fixing* and get $F, \mathbf{x}^F$

$\mathbf{b} \leftarrow \mathbf{b} - \mathbf{A}_k x_k^{ADP}$

$k \leftarrow k - 1$

5: Set x_1^{ADP} to the optimal solution of $MKP(1, \mathbf{b})$

6: Output: $\mathbf{x}^{BEST}$ and z^{BEST}

Figure 1: The ADP Base Heuristic Algorithm for the MKP.

Variable assignment:

In order to calculate x_k^{ADP} for a variable $k \notin F$, we employ Eq. (4) but with estimated values, $\widetilde{F}(k, \mathbf{b})$, instead of optimal ones, $F(k, \mathbf{b})$. This is the basic idea in ADP-BH. In our computations we noted that the following estimation generated better solutions in comparison to a straightforward estimation by a base heuristic value $H(k, \mathbf{b})$. We denote by $U(k, \mathbf{b})$ an upperbound to the problem $MKP(k, \mathbf{b})$. In our computations, we set the upperbound $U(k, \mathbf{b})$ to $\lfloor z^{LP(k,\mathbf{b})} \rfloor$, that is, the truncated LP solution value of the associated subproblem. Let ϵ_x denote the percentage deviation of $H(k-1, \mathbf{b} - \mathbf{A}_k x)$ with respect to the upperbound $U(k-1, \mathbf{b} - \mathbf{A}_k x)$ for $x = 0, 1$ and also let ϵ^* be the minimum of ϵ_0 and ϵ_1. We estimate the optimal values, $F(k-1, \mathbf{b} - \mathbf{A}_k x)$, by $\widetilde{F}(k-1, \mathbf{b} - \mathbf{A}_k x) = (1 - \epsilon^*) U(k-1, \mathbf{b} - \mathbf{A}_k x)$ for $x = 0, 1$. By the definition of ϵ^*, the optimal values are approximated such that the percentage deviations of the estimate values are the same with respect to the associated upperbounds. Finally, we set x_k^{ADP}

by Eq. (4) replacing the optimal values by the estimated ones.

Update best known solution and value:

Once we set x_k^{ADP}, if necessary, we update the *best-solution*, $\mathbf{x}^{BEST}$, and *best-solution-value*, z^{BEST}, as follows. Let $\mathbf{b} = \mathbf{b}_0 - \sum_{j=k}^{n} \mathbf{A}_j x_j^{ADP}$. Due to construction of x_j^{ADP} for j in $[k, n]$, $\mathbf{x}^C = (\mathbf{x}^{BH}(k-1, \mathbf{b}), x_k^{ADP}, \ldots, x_n^{ADP})$ is a feasible solution to the problem $MKP(n, \mathbf{b}_0)$ where objective value of the solution $\mathbf{x}^C$ equals $z^C = H(k-1, \mathbf{b}) + \sum_{j=k}^{n} c_j x_j^{ADP}$. We update $\mathbf{x}^{BEST}$ to $\mathbf{x}^C$ and z^{BEST} to z^C, respectively, if z^C is larger than z^{BEST}.

Early termination:

The ADP base-heuristic algorithm can be terminated *early* while setting x_k^{ADP} if we determine that we find an optimal solution to the problem $MKP(k, \mathbf{b})$ where $\mathbf{b} = \mathbf{b}_0 - \sum_{j=k+1}^{n} \mathbf{A}_j x_j^{ADP}$. We apply the following tests to determine whether we have an optimal solution or not to the problem $MKP(k, \mathbf{b})$: (a) We check if $\mathbf{x}^{BH}(k-1, \mathbf{b})$ and $\mathbf{x}^{BH}(k-1, \mathbf{b} - \mathbf{A}_k)$ are optimal solutions to the subproblems $MKP(k-1, \mathbf{b})$ and $MKP(k-1, \mathbf{b} - \mathbf{A}_k)$, respectively. We conclude that an optimal solution exists if $H(k-1, \mathbf{b}) = U(k-1, \mathbf{b})$ and $H(k-1, \mathbf{b} - \mathbf{A}_k) = U(k-1, \mathbf{b} - \mathbf{A}_k)$; (b) We conclude that $\mathbf{x}^{BH}(k, \mathbf{b}) = (\mathbf{x}^{BH}(k-1, \mathbf{b}), 0)$ is an optimal solution if $H(k-1, \mathbf{b})$ is greater than $U(k-1, \mathbf{b} - \mathbf{A}_k) + c_k$ and $H(k-1, \mathbf{b}) = U(k-1, \mathbf{b})$; (c) Similarly, we conclude that $\mathbf{x}^{BH}(k, \mathbf{b}) = (\mathbf{x}^{BH}(k-1, \mathbf{b} - \mathbf{A}_k), 1)$ is an optimal solution if $H(k-1, \mathbf{b} - \mathbf{A}_k) + c_k$ is greater than $U(k-1, \mathbf{b})$ and $H(k-1, \mathbf{b} - \mathbf{A}_k) = U(k-1, \mathbf{b} - \mathbf{A}_k)$.

Lag variable fixing:

In order to allow the ADP base-heuristic to generate solutions in significantly shorter times (possibly of lower quality) we allow fixing a set of variables x_j for j in $[k-l, k-1]$ and $j \notin F$. We name the method *lag-variable-fixing* and those variables as *lag-fixable variables*. We denote by $F^{LAG,k} = \{j : j \in [k-l, k-1] \text{ and } j \notin F\}$. Once the k-th variable is assigned to 0 or 1, the variables x_j^F are set to $x_j^{BH}(k-1, \mathbf{b})$ or $x_j^{BH}(k-1, \mathbf{b} - \mathbf{A}_k)$, respectively, for all $j \in F^{LAG,k}$. We update the set F by including those indices in $F^{LAG,k}$. x_j^{ADP} are assigned to x_j^F for all $j \in F^{LAG,k}$ in Step 4 of Figure 1. `Lag-size` parameter l specifies the number of variables to be fixed lagging from k-th variable while setting x_k^{ADP} through the relationship, $l = \lfloor \frac{k}{\text{lag-time}} \rfloor$, where `lag-time` is a user-specified parameter.

3.1 Base-Heuristic Selection

The selection of the base-heuristic is important in the proposed ADP base-heuristic framework. Since we employ the base-heuristic many times within the ADP Base Heuristic framework, we focus on heuristic methodologies with small computation time requirements. Based on the available computational experience reported in the literature we consider the following heuristic methodologies as the base-heuristic for the MKP: Senju and Toyoda's *Dual Gradient Algorithm* [49], which starts from an infeasible solution and constructs a feasible solution by setting variables to zero with the lowest gradient. The gradient of variable j is calculated as the ratio of c_j and penalty of the j-th variable, which is $\mathbf{A}_j$ times the excess capacity usage by those variables at one. Toyoda's *Primal Gradient Algorithm* [53], which constructs a feasible solution from the all-zero solution by setting variables to one with the largest gradient. Loulou and Michaelides' *Greedy-like Heuristics* [33], a variation of primal gradient with complicated gradient computation and Kochenberger et. al.'s *Incremental Heuristic* [21], a variant of primal gradient designed for generic integer programming problems. We also propose a new heuristic that constructs solutions by adaptively fixing variables through solving linear programming relaxations iteratively, which we name *adaptive fixing heuristic*. We describe the details of this base-heuristic next.

3.1.1 A New Base-heuristic: Adaptive Fixing

We denote by $LP(k, \mathbf{b})$ the linear programming relaxation (LP) of the problem $MKP(k, \mathbf{b})$. We use X_0 and X_1 to denote the set of variables that are set to 0 and 1, respectively. We also define by $LP^C = [LP(k, \mathbf{b})|X_0, X_1]$ the $LP(k, \mathbf{b})$ with the constraints of $x_j = 0,\ j \in X_0$ and $x_j = 1,\ j \in X_1$. The optimal solution of the linear program LP^C is denoted by $\mathbf{x}^{LP^C}$.

The adaptive fixing (AF) heuristic, at first, assumes both X_0 and X_1 are empty sets, meaning that none of the variables $x_1, \ldots, x_k$ are assigned to a value. We solve $LP^C = [LP(k, \mathbf{b})|X_0, X_1]$ and get the optimal solution $\mathbf{x}^{LP^C}$. We fix variables x_j to 0 if $0 \leq x_j^{LP^C} < \gamma$, where γ is a user-specified parameter, and x_j to 1 if $x_j^{LP^C} = 1$. X_0 and X_1 are updated accordingly (Step 3 in Figure 2). We iteratively fix variables (Step 4 in Figure 2): Once we solve $LP^C = [LP(k, \mathbf{b})|X_0, X_1]$ and get $\mathbf{x}^{LP^C}$, we find the fractional variable with the lowest value, if any exists, and set it to 0. We also include those variables to X_0 and X_1, if $x_j^{LP^C} = 0$ and $x_j^{LP^C} = 1$, respectively. We repeat this process until no fractional variable exists when solving the adaptively changed linear program $LP^C = [LP(k, \mathbf{b})|X_0, X_1]$, meaning that all the variables are either assigned to 0 or 1. We define by $\mathbf{x}^{AF}$ and z^{AF} the heuristic solution and value, respectively, returned by the adaptive fixing heuristic (Step 5 in Figure 2). We set $x_j^{AF} = 0$ and $x_j^{AF} = 1$ for all j in X_0 and X_1, respectively. Thus, the solution value is $z^{AF} = \displaystyle\sum_{j=1}^{k} c_j x_j^{AF}$.

Our computational study suggests adaptive fixing with low γ (e.g., $\gamma = 0.05, 0.10$) values generates usually better solutions. On the other hand, it uses more computation time. Thus, in order to balance the trade-off between solution time and quality we set $\gamma = 0.25$ in our computational study.

We consider the special case $\gamma = 1$ of the AF heuristic, which we name *truncation heuristic*. We denote by $\mathbf{x}^T$ and z^T the truncation heuristic solution and value, respectively. We first

1: Initialization:

$\gamma = 0.25,\ X_1 = \emptyset,\ X_0 = \emptyset$

2: Solve $LP^C = [LP(k, \mathbf{b})|X_0, X_1]$ and get $\mathbf{x}^{LP^C}$

3: Update X_0 and X_1

$X_0 \leftarrow X_0 \cup \{j|\ 0 \leq x_j^{LP^C} < \gamma\}$

$X_1 \leftarrow X_1 \cup \{j|\ x_j^{LP^C} = 1\}$

4: While ($\mathbf{x}^{LP^C}$ includes fractional values) do

Solve $LP^C = [LP(k, \mathbf{b})|X_0, X_1]$ and get $\mathbf{x}^{LP^C}$

$X_0 \leftarrow X_0 \cup \{j|\ x_j^{LP^C} = 0\}$

$X_1 \leftarrow X_1 \cup \{j|\ x_j^{LP^C} = 1\}$

$j^* = \underset{0 < x_j^{LP^C} < 1}{\operatorname{argmin}} \{x_j^{LP^C}\}$

$X_0 \leftarrow X_0 \cup \{j^*\}$

5: Compute the AF solution and value: $\mathbf{x}^{AF}, z^{AF}$

Set $x_j^H = 0$, for all $j \in X_0$ and $x_j^H = 1$, for all $j \in X_1$

Calculate $z^{AF} = \sum_{j=1}^{k} c_j x_j^{AF}$

Figure 2: Description of the *adaptive fixing heuristic* for the $MKP(k, \mathbf{b})$.

solve the linear program $LP(n, \mathbf{b}_0)$ and obtain the optimal solution $\mathbf{x}^{LP}$. Then, we set $x_j^T = 0$ if $0 \leq x_j^{LP} < 1$ and $x_j^T = 1$ if $x_j^{LP} = 1$ for j in $[1, n]$. The corresponding heuristic value is $z^T = \sum_{j=1}^{k} c_j x_j^T$.

4 ADP-Parametric and ADP-Nonparametric

In this section, we describe our parametric and nonparametric approach to approximating optimal value functions. The basic motivation is to estimate $F(k, \mathbf{b})$ using a set of $F(k, \mathbf{b}^i)$ values (or possibly estimates of them) where $\mathbf{b}^i = (b_1^i, \ldots, b_m^i)'$, $i = 1, \ldots, s$ are sampled from the state space $\Omega = \{\mathbf{b} \in \Re^m | \mathbf{0} \leq \mathbf{b} \leq \mathbf{b}_0\}$ for a given capacity $\mathbf{b}_0 = (b_{0,1}, \ldots, b_{0,m})'$ of the MKP problem. Let $S = \{\mathbf{b}^i \in \Omega | i = 1, \ldots, s\}$ be a sample of the state space Ω, which we chose in our computational study as follows: $\mathbf{b}^i = (i\lfloor b_{0,1}/s \rfloor, \ldots, i\lfloor b_{0,m}/s \rfloor)$ for $i = 1, \ldots, s-1$ and $\mathbf{b}^s = \mathbf{b}_0$. We name this sample S the *uniform sample*. We also enlarge the uniform sample by taking additional sample points $\mathbf{b} = (b_1, \ldots, b_m)'$ such that $\mathbf{b}^k \leq \mathbf{b} \leq \mathbf{b}^{k+1}$, where $b_i \sim U(b_i^k, b_i^{k+1})$ (i.e., uniformly distributed between b_i^k and b_i^{k+1}) and $\mathbf{b}^k, \mathbf{b}^{k+1} \in S$, which we name the *stratified sample*. In our computations, we found that the stratified sample does not necessarily generate better solutions than the uniform sample. An alternative to selecting the uniform sample is to choose the set S randomly over the state space Ω. We have found, however, that this choice is inferior to selecting the set S as outlined above. Thus, we fix *uniform sampling* as our sampling scheme. We define by *stage* k the number of first k variables of the MKP problem. We denote by $\Omega_k = \{(k, \mathbf{b}) | \mathbf{b} \in \Omega\}$ and $S_k = \{(k, \mathbf{b}) | \mathbf{b} \in S\}$, the state space and the sample for a given stage k in $[1, n]$. The estimate of $F(k, \mathbf{b}) : \Omega_k \rightarrow \Re$ is denoted by $\widetilde{F}(k, \mathbf{b})$.

4.1 ADP-Parametric Algorithm

We motivate the parametric approximation method using research results from the probabilistic analysis of the MKP (see Frieze and Clarke [20], Meanti et. al. [38], Dyer and Frieze [15], Szkatuła [51], Piersma [42]). Let $\mathbf{A}_j = (a_{1,j}, \ldots, a_{m,j})'$ be nonnegative i.i.d. random vectors and suppose that c_j are i.i.d. random variables, with $j \in \{1, \ldots, n\}$, that are independent of the $\mathbf{A}_j$. Let $b_i = n\tau_i$ for some fixed m-vector $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_m)'$. The solution value of the MKP, defined by the random variables $\mathbf{A}_j$ and c_j and by the constants $b_i \geq 0$ is denoted by Z_n^{MKP}. Let us denote by Z_n^{LP} the solution value of the LP relaxation of the MKP. Piersma [42] and Meanti et. al. [38] show that with probability one Z_n^{LP}/n and Z_n^{MKP}/n converge, as $n \rightarrow \infty$ and m remains fixed, to the same constant ρ that depends on the parameters b_i and on the expected values $E[a_{i,1}]$ and $E[c_1]$,

$$\rho = \min_{\boldsymbol{\lambda} \geq 0} E f_{\boldsymbol{\lambda}}(c_1, \mathbf{A}_1), \qquad f_{\boldsymbol{\lambda}}(t_0, \mathbf{t}) = \sum_{i=1}^{m} \lambda_i \tau_i + \max(0, t_0 - \sum_{i=1}^{m} \lambda_i t_i).$$

Thus, the previous results suggest that Z_n^{MKP} scales linearly with n when the vector $\mathbf{b} = n\boldsymbol{\tau}$.

The above fact motivates us to approximate $F(k, \mathbf{b})$ for a given $(k, \mathbf{b}) \in \Omega_k$ by a linear functional model, $\boldsymbol{\alpha}_k' \mathbf{b} + \beta_k$, with parameters $\boldsymbol{\alpha}_k = (\alpha_{k,1}, \ldots, \alpha_{k,m})'$ and β_k. We denote by $\tilde{\boldsymbol{\alpha}}_k$ and $\tilde{\beta}_k$ the *tuned* model parameters and we now describe how we calculate them. Let us assume that we know $\tilde{\boldsymbol{\alpha}}_{k-1}$ and $\tilde{\beta}_{k-1}$, that is, we can estimate $F(k-1, \mathbf{b})$ and $F(k-1, \mathbf{b} - \mathbf{A}_k)$ by $\widetilde{F}(k-1, \mathbf{b}) = \tilde{\boldsymbol{\alpha}}_{k-1}' \mathbf{b} + \tilde{\beta}_k$ and $\widetilde{F}(k-1, \mathbf{b} - \mathbf{A}_k) = \tilde{\boldsymbol{\alpha}}_{k-1}' (\mathbf{b} - \mathbf{A}_k) + \tilde{\beta}_k$, respectively. For a given $(k, \mathbf{b}) \in S_k$, we calculate $\widetilde{F}(k, \mathbf{b})$ by applying Eq. (3) and using $\widetilde{F}(k-1, \mathbf{b})$ and $\widetilde{F}(k-1, \mathbf{b} - \mathbf{A}_k)$ values. Then, we find $\tilde{\boldsymbol{\alpha}}_k$ and $\tilde{\beta}_k$ by minimizing the total absolute deviation of $\widetilde{F}(k, \mathbf{b})$ from the linear model values $\boldsymbol{\alpha}_k' \mathbf{b} + \beta_k$ for all $(k, \mathbf{b})$ in S_k as follows:

$$LAD(k) \qquad \underset{\boldsymbol{\alpha}_k, \beta_k}{\text{minimize}} \sum_{(k, \mathbf{b}) \in S_k} |\boldsymbol{\alpha}_k' \mathbf{b} + \beta_k - F(k, \mathbf{b})|. \tag{5}$$

In our computations, we reformulate $LAD(k)$ as a linear program (see Bertsimas and Tsitsiklis [9]) and use CPLEX to find $\tilde{\boldsymbol{\alpha}}_k$ and $\tilde{\beta}_k$ values. We name tuning the model parameters the *learning-phase* of ADP-P algorithm. To start the learning phase, we first calculate $F(1,\mathbf{b})$ for all $(k,\mathbf{b}) \in S_k$ and we find $\tilde{\boldsymbol{\alpha}}_1$ and $\tilde{\beta}_1$ using Eq. (5). We iteratively tune model parameters as described above and store them in a *look-up* table. We now describe how we construct ADP-P solution, $\mathbf{x}^{ADP} = (x_1^{ADP}, \ldots, x_n^{ADP})$, which we name ADP-P *solution construction* phase. We initialize $\mathbf{b} = \mathbf{b}_0$ and $k = n$ (Step 1 in Figure 3). To calculate x_k^{ADP} (Step 2 in Figure 3): We calculate $\widetilde{F}(k-1,\mathbf{b}) = \tilde{\boldsymbol{\alpha}}'_{k-1}\mathbf{b} + \tilde{\beta}_{k-1}$ and $\widetilde{F}(k-1,\mathbf{b}-\mathbf{A}_k) = \tilde{\boldsymbol{\alpha}}'_{k-1}\mathbf{b} + \tilde{\beta}_{k-1}$ by retrieving $\tilde{\boldsymbol{\alpha}}_{k-1}$ and $\tilde{\beta}_{k-1}$ from the look-up table. We set x_k^{ADP} according to Eq. (4), in which we use estimated values, and we update $\mathbf{b}$ by $\mathbf{b} - \mathbf{A}_k x_k^{ADP}$. We iteratively find x_k^{ADP} for $k = n$ to 1, which is also described in Figure 3.

1: $\mathbf{b} \leftarrow \mathbf{b}_0$

2: For $k = n, \ldots, 2$ do

  Find $\widetilde{F}(k-1,\mathbf{b}-\mathbf{A}_k x)$ for $x \in \{0,1\}$

    Retrieve $\tilde{\boldsymbol{\alpha}}_{k-1}, \tilde{\beta}_{k-1}$ from the look-up table

    $\widetilde{F}(k-1,\mathbf{b}-\mathbf{A}_k x) \leftarrow (\tilde{\boldsymbol{\alpha}}_{k-1})'(\mathbf{b}-\mathbf{A}_k x) + \tilde{\beta}_{k-1}$

  $x_k^{ADP} \leftarrow \operatorname{argmax}_{x\in\{0,1\}}\{\widetilde{F}(k-1,\mathbf{b}-\mathbf{A}_k x) + c_k x\}$

  $\mathbf{b} \leftarrow \mathbf{b} - \mathbf{A}_k x_k^{adp}$

3 : Set x_1^{ADP} to the optimal solution of the problem $MKP(1,\mathbf{b})$

  Output: $\mathbf{x}^{ADP} = (x_1^{ADP}, \ldots, x_n^{ADP})$

Figure 3: ADP-P Solution Construction Phase.

4.2 ADP-Nonparametric Algorithm

Suppose we know $F(k,\bar{\mathbf{b}})$ for all $(k,\bar{\mathbf{b}}) \in S_k$. Under the nonparametric approximation, we estimate the optimal value of a problem $MKP(k,\mathbf{b})$ for which $(k,\mathbf{b}) \notin S_k$ as follows:

$$\widetilde{F}(k,\mathbf{b}) = \frac{\sum\limits_{(k,\bar{\mathbf{b}})\in S_k} w(\mathbf{b},\bar{\mathbf{b}})F(k,\bar{\mathbf{b}})}{\sum\limits_{(k,\bar{\mathbf{b}})\in S_k} w(\mathbf{b},\bar{\mathbf{b}})}, \tag{6}$$

where $w(\mathbf{b},\bar{\mathbf{b}})$ are local weights assigned to sample points $(k,\bar{\mathbf{b}})$ in S_k. Given $\mathbf{b},\bar{\mathbf{b}}$, we calculate

$$w(\mathbf{b},\bar{\mathbf{b}}) = \sum_{i=1}^{m} K\left(\tfrac{|b_i-\bar{b}_i|}{h_i}\right),$$

where $\mathbf{h} = (h_1, \ldots, h_m)$ and $K(\cdot) : \Re \to \Re$ are user-specified. We utilize $K(t) = (1-|t|)I(|t| \leq 1)$ (see Fan [16]), where $I(\cdot)$ is an indicator function, i.e., $I(|t| \leq 1) = 1$ (0) if $|t| \leq 1$ (otherwise). We use $\mathbf{h} = \boldsymbol{\delta}_q + \boldsymbol{\delta}_r$ through setting $\boldsymbol{\delta}_q = (\lfloor b_{0,1}/s \rfloor, \ldots, \lfloor b_{0,m}/s \rfloor)$ and $\boldsymbol{\delta}_r = \mathbf{b}_0 - s\boldsymbol{\delta}_q$, where s denotes the sample size of S_k.

Let us denote by $\mathbf{x}^{ADP}$ the ADP-N solution. The key element in calculating x_k^{ADP} is being able to estimate the optimal values. Under the nonparametric approximation, this corresponds knowing a set of optimal values for those states in the given sample space. In our computations, we use estimated values instead optimal ones. Initially, we calculate estimates $\widetilde{F}(k, \mathbf{b})$ for all states $(k, \mathbf{b}) \in S_k$ for $k = 1, \ldots, n$, which we name the ADP-N learning phase. Let us assume that $\widetilde{F}(k-1, \bar{\mathbf{b}})$ for all $(k-1, \bar{\mathbf{b}})$ in S_{k-1} are known. Given $(k, \mathbf{b}) \in S_k$, in order to calculate $\widetilde{F}(k, \mathbf{b})$ we first estimate $F(k-1, \mathbf{b})$ and $F(k-1, \mathbf{b} - \mathbf{A}_k)$ by $\widetilde{F}(k-1, \mathbf{b}) = \sum_{(k-1,\bar{\mathbf{b}}) \in S_{k-1}} \hat{w}(\mathbf{b}, \bar{\mathbf{b}}) \widetilde{F}(k-1, \bar{\mathbf{b}})$ and $\widetilde{F}(k-1, \mathbf{b} - \mathbf{A}_k) = \sum_{(k-1,\bar{\mathbf{b}}) \in S_{k-1}} \hat{w}(\mathbf{b}, \bar{\mathbf{b}}) \widetilde{F}(k-1, \bar{\mathbf{b}} - \mathbf{A}_k)$, respectively, and then apply Eq. (3), where $\hat{w}(\mathbf{b}, \bar{\mathbf{b}}) = \dfrac{w(\mathbf{b}, \bar{\mathbf{b}})}{\sum_{(k-1,\bar{\mathbf{b}}) \in S_{k-1}} w(\mathbf{b}, \bar{\mathbf{b}})}$. In the ADP-N learning phase, we first calculate $F(1, \mathbf{b})$ for all $(1, \mathbf{b}) \in S_1$ and then find $\widetilde{F}(k, \mathbf{b})$ for all $(k, \mathbf{b})$ in S_k for $k = 2, \ldots, n$ by using Eqs. (3) and (6). Finally, we store those estimates for the sample space in a *look-up* table.

We construct an ADP-N solution as follows. We first initialize $\mathbf{b} = \mathbf{b}_0$ and $k = n$ (Step 1 in Figure 4). To calculate x_k^{ADP} (Step 2 in Figure 4), we find $\widetilde{F}(k-1, \mathbf{b})$ and $\widetilde{F}(k-1, \mathbf{b} - \mathbf{A}_k)$ by using Eq. (6) and by retrieving sample values $\widetilde{F}(k-1, \bar{\mathbf{b}})$ for all $(k-1, \bar{\mathbf{b}}) \in S_{k-1}$ from the look-up table. Once we set x_k^{ADP} according to Eq. (4), in which we use estimated values, we update $\mathbf{b}$ by $\mathbf{b} - \mathbf{A}_k x_k^{ADP}$. We iteratively find x_k^{ADP} for $k = n$ to 1, which is also described in Figure 4.

Once ADP-P or ADP-N generate a feasible solution $\mathbf{x}^{ADP}$ with X_0 (set of variables assigned to zero in the solution) and X_1 (set of variables assigned to one in the solution), we calculate the slack vector $\mathbf{s} = \mathbf{b}_0 - \sum_{j \in X_1} \mathbf{A}_j$. We iteratively set a variable $x_k, k \in X_0$ to 1 if $\mathbf{A}_k \leq \mathbf{s}$ while updating $\mathbf{s}$ accordingly (i.e., $\mathbf{s} = \mathbf{s} - \mathbf{A}_k$ once x_k is set to 1). This final step can be considered as a *local* improvement to the solution obtained by ADP-P or ADP-N.

5 Computational Results

In this section, we provide computational results for all the ADP-based approaches we developed for the MKP. We set as performance criteria for comparing different methods: solution quality, computation time, and robustness, i.e., the degree of deviation of the computational resources needed to solve the problems as the instances of the same size change.

Our overall objective is to assess whether ADP is a promising methodology for MKPs. More specifically, we want to answer the following questions:

1: $\mathbf{b} \leftarrow \mathbf{b}_0$

2: For $k = n, \ldots, 2$ do

Find $\widetilde{F}(k-1, \mathbf{b} - \mathbf{A}_k x)$ for $x \in \{0, 1\}$

Retrieve $\widetilde{F}(k-1, \bar{\mathbf{b}}), (k-1, \bar{\mathbf{b}}) \in S_{k-1}$ from the look-up table

$$\widetilde{F}(k-1, \mathbf{b} - \mathbf{A}_k x) \leftarrow \frac{\sum_{(k-1,\bar{\mathbf{b}}) \in S_{k-1}} w(\mathbf{b} - \mathbf{A}_k x, \bar{\mathbf{b}}) \widetilde{F}(k-1, \bar{\mathbf{b}})}{\sum_{(k-1,\bar{\mathbf{b}}) \in S_{k-1}} w(\mathbf{b} - \mathbf{A}_k x, \bar{\mathbf{b}})}$$

$$w(\mathbf{b}, \bar{\mathbf{b}}) = \sum_{l=1}^{m} K\left(\frac{|b_l - \bar{b}_l|}{h_l}\right)$$

$x_k^{ADP} \leftarrow$ argmax $_{x \in \{0,1\}} \{\widetilde{F}(k-1, \mathbf{b} - \mathbf{A}_k x) + c_k x\}$

$\mathbf{b} \leftarrow \mathbf{b} - \mathbf{A}_k x_k^{ADP}$

3 : Set x_1^{ADP} to the optimal solution of the problem $MKP(1, \mathbf{b})$

Output: $\mathbf{x}^{ADP} = (x_1^{ADP}, \ldots, x_n^{ADP})$

Figure 4: ADP-N Solution Construction Phase.

1. Given that the choice of the base-heuristic is important for the base-heuristic approach, how does the adaptive fixing heuristic compare with alternative approaches in the literature?

2. How do heuristic methods based on parametric and nonparametric value approximations perform?

3. Is there an improvement of the base-heuristic performance when a base-heuristic framework is applied? How significant is such an improvement ?

4. How does the best of the ADP-based methods compare with the state of the art heuristics as well as commerical packages (CPLEX)?

5. Most importantly, is ADP for MKPs a competitive method worth further studying?

The instances of the MKP we used in our experiments were both randomly generated as well as from the literature. We use the notation $x \sim U(1, X)$ to denote an integer number that is uniformly generated in $[1, X]$. We construct *uncorrelated (UC)*, *weakly (WC)* and *strongly correlated (SC)* random MKP instances as follows. Let $N = \{1, \ldots, n\}$ and $M = \{1, \ldots, m\}$.

- Uncorrelated instances: $c_j \sim U(1, C)$ and $a_{ij} \sim U(1, A)$ for all $i \in M$ and $j \in N$, i.e., c_j, a_{ij} are uniformly distributed in $[1, C], [1, A]$ respectively.

- Weakly correlated instances: $a_{ij} \sim U(1, A)$ and $c_j = \max\{1, U(\sum_i a_{ij}/\ m - wc, \sum_i a_{ij}/m + wc)\}$ for all $i \in M$ and $j \in N$, where wc is a user-specified parameter (see Table 1).

- Strongly correlated instances: $a_{ij} \sim U(1, A)$ and $c_j = \sum_i a_{ij}/m + sc$ for all $i \in M$ and $j \in N$ where sc is a user-specified parameter (see Table 1).

Given the number of constraints m and the number of variables n we generate 10 test problems in each class (uncorrelated, weakly, and strongly correlated) with different parameters C, A, wc, sc as provided in Table 1. We set $b_{0,i} = 0.5 \sum_{j=1}^{n} a_{ij},\ i = 1, \ldots, m$. We make our instances available on the internet[1]. All computational studies are done on a Dell Precision 410 with Linux operating system. We use the following notation to present the computational study.

- Let v(X) be the objective value of the solution obtained by the methodology X, e.g., v(ADP-H) is the objective value of the solution obtained by the methodology ADP-H.

- Let PE(X) be the percentage deviation of v(X) from the linear programming (LP)relaxation objective value v(LP), i.e., PE(X) $= \frac{v(LP)-v(X)}{v(LP)} \times 100$.

- PI(X): Percentage Improvement of ADP-X over X, i.e., $\frac{PE(X)-PE(ADP-X)}{PE(X)} \times 100$

- T(X): Computation time of the methodology X in CPU seconds.

Uncorrelated		Weakly-correlated		Strongly-correlated	
A	C	A	wc	A	sc
100	100	100	10	100	10
500	100	500	50	500	50
500	1,000	1,000	100	1,000	100
1,000	500	1,000	500	1,000	500
1,000	1,000	3,000	300	2,000	500
1,000	2,000	4,000	500	3,000	100
5,000	1,000	5,000	500	5,000	500
5,000	500	10,000	1,000	5,000	1,000
10,000	1,000	10,000	2,000	10,000	2,000
10,000	5,000	15,000	3,000	10,000	5,000

Table 1: Parameters for the uncorrelated, weakly and strongly correlated type problems.

[1]http://web.mit.edu/dbertsim/mkp

5.1 Comparison of base-heuristics

In this section, we compare the proposed stand-alone heuristics, namely the truncation heuristic H1 (see Section 3.1) and adaptive fixing heuristic H2 (see Section 3.1 and Figure 2), with some heuristics from the literature, namely the primal gradient heuristic H3 (Toyoda [53]), dual gradient heuristic H4 (Senju and Toyoda [49]), greedy-like heuristic H5 (Loulou and Michaelides [33]), incremental heuristic H6 (Kochenberger et. al. [21]). We compare these heuristics on uncorrelated, weakly correlated and strongly correlated problem instances as described earlier.

			H1		H2		H3		H4		H5		H6	
m	n	type	T	PE	T	PE	T	PE	T	PE	T	PE	T	PE
10	100	UC	0.01	3.92	0.01	1.69	0.00	35.12	0.01	38.56	0.11	34.49	0.02	34.52
10	100	WC	0.01	8.08	0.02	2.36	0.01	18.52	0.00	20.98	0.22	19.70	0.01	16.86
10	100	SC	0.02	9.81	0.02	3.84	0.01	8.83	0.01	13.83	0.21	10.72	0.01	10.11
50	100	UC	0.02	9.59	0.07	2.13	0.03	31.77	0.02	37.79	0.76	33.67	0.07	30.96
50	100	WC	0.05	26.85	0.20	3.58	0.04	17.38	0.02	21.65	1.07	17.56	0.06	16.88
50	100	SC	0.13	50.45	0.60	4.26	0.04	10.84	0.03	14.47	1.07	11.23	0.06	10.31
100	500	UC	0.36	3.25	0.93	0.45	2.21	35.31	1.51	36.05	209.17	36.09	3.46	34.59
100	500	WC	0.97	10.29	3.25	0.85	2.42	16.97	1.28	17.25	215.00	17.67	3.41	16.88
100	500	SC	6.93	19.87	14.92	1.30	2.27	6.50	1.57	7.58	201.82	6.58	3.52	6.40
100	1000	UC	1.04	1.94	2.69	0.28	10.44	34.51	6.78	34.71	2,015.61	34.68	15.45	34.28
100	1000	WC	2.31	6.29	7.13	0.46	10.68	15.92	5.58	16.68	1,995.10	15.72	14.93	15.74
100	1000	SC	8.42	10.00	18.67	0.59	10.79	4.84	5.69	5.89	1,986.85	5.07	14.95	5.03

Table 2: Base-heuristics results.

The results in Table 2 suggest that H2 is the leading algorithm among the ones we examined in terms of both solution quality and computation time. It constructs high quality solutions within short computation times for all types of instances where percentage deviations are 1.14, 1.81 and 2.49 for uncorrelated, weakly-correlated and strongly-correlated problems, respectively.

5.2 Computational results for the ADP parametric and ADP non-parametric

In this section, our objective is to demonstrate the performance of the ADP-parametric (ADP-P) and ADP-nonparametric (ADP-N) algorithms (see Section 4) on the same uncorrelated, weakly correlated and strongly correlated problem instances. For a given m, n combination (e.g., $m = 10, n = 100$) we apply ADP-P or ADP-N and get the associated solution statistics such as solution quality and computation time for varying sample sizes s in $\{5, 10, 15, 20, 25\}$. We report the associated times T and percentage deviations PEs in Table 3. In our study, we observed that solution qualities of ADP-N and ADP-P did not necessarily improve on larger samples, where s was taking values of $\{50, 100, 150, 200, 250\}$.

			T(ADP-P)			PE(ADP-P)			T(ADP-N)			PE(ADP-N)		
m	n	type	min	avg	max	min	avg	max	min	avg	max	min	avg	max
10	100	UC	0.03	0.06	0.10	8.0118	16.8413	25.0646	0.02	0.05	0.07	3.7448	10.5054	21.9889
10	100	WC	0.03	0.06	0.09	9.7910	14.0499	22.6718	0.02	0.05	0.07	6.0482	11.8696	19.7634
10	100	SC	0.03	0.06	0.11	5.8559	8.6297	11.7666	0.02	0.05	0.008	5.0795	8.7346	13.5535
50	100	UC	0.10	0.19	0.26	10.1345	19.0510	30.6011	0.09	0.15	0.23	4.9710	14.4108	26.8070
50	100	WC	0.09	0.19	0.27	10.4399	14.0215	26.2094	0.09	0.16	0.24	9.0369	12.9657	23.3960
50	100	SC	0.10	0.19	0.30	7.5812	10.3166	12.0993	0.09	0.18	0.28	7.6882	10.4663	13.6747
100	500	UC	0.95	1.61	2.25	4.4152	11.4099	21.3784	0.96	1.62	2.33	2.5496	10.6635	21.8752
100	500	WC	0.96	1.83	2.88	5.1505	7.9797	10.7776	0.95	1.74	2.62	6.0106	8.4587	22.4693
100	500	SC	0.95	2.91	8.40	4.4200	5.3768	6.4244	0.96	2.95	8.47	4.1320	5.3237	6.6767
100	1000	UC	1.98	3.50	4.96	4.1738	7.0525	11.5795	1.98	3.42	4.85	3.4661	8.8820	22.9663
100	1000	WC	1.99	3.76	6.13	4.1417	6.8218	8.7893	2.00	3.83	6.03	3.6354	6.8907	17.0984
100	1000	SC	1.99	4.92	12.11	2.8619	3.8747	5.3549	2.00	4.80	10.96	2.9938	4.0020	5.5049

Table 3: ADP-P and ADP-N results.

In summary, using Table 3 and 2 statistics, both ADP-P and ADP-N are capable of generating solutions at least as good as the ones obtained by heuristics H1, H3, H4, H5 and H6. We note that ADP-P and ADP-N need extra tuning in order to determine the best sample size because the quality of solutions deviates substantially for varying sample sizes. As an interesting observation, average percentage deviations and computation times of ADP-P and ADP-N stayed stable across different types of instances which illustrates robustness across uncorrelated, weakly and strongly correlated problem instances. Overall, the adaptive fixing heuristic H2 continues to be the leading heuristic compared to other heuristics, namely H1, H3, H4, H5, H6, ADP-P and ADP-N in terms of both solution quality and computation time.

5.3 Computational results for the ADP base-heuristic

In this section, our basic aim is to illustrate the performance of the ADP base-heuristic algorithm. We consider heuristics H1, H2, H3, H4, H5 and H6 as our base-heuristics. Let ADP-X be the ADP base-heuristic algorithm with the base-heuristic X (e.g., ADP-H1 is the ADP base-heuristic algorithm with the base-heuristic H1). In this computational study, we apply *lag-variable-fixing* scheme in which we set `lag-time`=10 (see Section 3 for a description). We apply the ADP base-heuristic algorithm to the same uncorrelated, weakly correlated and strongly correlated problem instances for which we present the results in Table 4.

Our available computational study illustrates that ADP with adaptive fixing heuristic, ADP-H2, is the most promising methodology in terms both solution quality and computation time. ADP-H2 generated near-optimal solutions in modest computation times for large-scale instances

			ADP-H1			ADP-H2			ADP-H3			ADP-H4			ADP-H5			ADP-H6		
m	n	type	T	PE	PI(H1)	T	PE	PI(H2)	T	PE	PI(H3)	T	PE	PI(H4)	T	PE	PI(H5)	T	PE	PI(H6)
10	100	UC	0.47	1.63	58.26	0.74	0.89	34.00	0.51	20.95	40.23	0.27	21.14	44.63	5.88	21.29	36.82	0.94	20.54	39.82
10	100	WC	0.67	4.90	41.06	1.25	1.19	37.31	0.60	10.64	39.38	0.40	12.00	39.84	6.23	10.57	44.32	0.83	11.65	29.73
10	100	SC	0.80	5.75	39.65	2.25	1.48	49.21	0.61	5.28	39.92	0.38	6.98	47.39	6.15	5.93	40.27	0.78	5.72	39.89
50	100	UC	2.56	3.05	65.22	6.12	1.33	31.08	2.07	18.80	39.12	1.14	20.45	45.26	38.17	21.75	32.27	4.31	18.60	37.06
50	100	WC	7.54	12.88	51.00	27.38	2.29	34.64	2.63	12.36	28.09	1.81	14.07	35.10	39.28	12.29	26.04	3.84	12.02	26.82
50	100	SC	10.37	10.98	77.82	69.20	3.13	24.76	2.59	7.83	27.23	1.70	8.85	35.87	39.11	7.99	28.20	3.21	7.79	23.18
100	500	UC	23.88	0.70	77.53	58.23	0.32	24.71	20.14	32.73	7.04	13.94	33.00	8.21	1,151.20	33.74	6.38	32.26	32.14	6.83
100	500	WC	60.65	2.71	73.29	239.67	0.61	27.82	17.71	15.18	9.84	12.07	15.95	7.12	901.60	15.88	9.32	24.72	15.42	7.06
100	500	SC	110.59	3.21	83.81	814.32	0.95	24.86	23.60	5.41	16.47	18.21	6.63	12.08	1,267.80	5.69	13.05	30.10	5.32	6.31
100	1000	UC	58.70	0.40	79.39	163.11	0.17	32.32	81.62	33.14	3.89	63.21	33.27	4.03	10,540.60	33.26	4.06	122.56	32.98	3.78
100	1000	WC	118.00	1.48	77.02	513.88	0.34	26.07	70.86	14.85	6.62	46.56	15.59	7.09	8,805.60	14.94	3.94	102.11	14.64	6.02
100	1000	SC	169.31	1.67	83.80	1,268.40	0.47	20.09	83.65	4.27	11.40	50.12	5.35	8.94	8,311.20	4.49	11.36	106.39	4.30	13.93

Table 4: ADP base-heuristics results.

with thousands of variables. The average percentage deviations of ADP-H2 are 0.68, 1.11 and 1.51 for uncorrelated, weakly and strongly correlated instances, respectively. We also observe that solution qualities of ADP-H2 do not deviate from one another. As an example, for the problems of size $m = 100, n = 1,000$, the average percentage deviations of ADP-H2 are 0.17, 0.34 and 0.47 for uncorrelated, weakly and strongly correlated problem instances, respectively. An important observation is that ADP base-heuristic algorithm provides a framework that exploits a certain base-heuristic to generate higher quality of solutions than those obtained by the corresponding base-heuristic at an additional computation time. We measure the corresponding enhancement in the quality of solutions through percentage improvements as provided in Table 4. For instance, average percentage improvements of ADP-H2 over H2 are 30.38, 31.62 and 29.73 for uncorrelated, weakly-correlated and strongly-correlated problems, respectively.

5.4 Test problems from the literature

In this section, our aim is to compare the most promising ADP methodology, ADP-H2, with one of the best heuristics in the recent literature for MKPs. In recent research, Chu and Beasley [11] developed a genetic algorithm (GA) to solve MKPs. Their computational study shows that GA provides high quality solutions at modest computation time to some large MKPs. In addition, to our knowledge, they attempted to solve the largest instances in the literature. We compare H2 and ADP-H2 with GA on their largest instances[2]. Their problem generation scheme is as follows: $a_{ij} \sim U(1, 1000)$, $c_j = \sum_i a_{ij}/m + 500q_j$ where $q_j \sim U(0, 1)$ for all $i \in M$ and $j \in N$. For each $(m, n) \in \{(30, 250), (30, 500)\}$ combination, the right hand side coefficients (b_i's for all $i \in M$) were set using $b_i = \tau \sum_{j=1}^{n} a_{ij}$ where τ is a user-specified tightness ratio taking values $0.25, 0.50,$ and 0.75. Their computations are conducted on a Silicon Indigo workstation which is supposed to be two times slower than our machine Dell Precision 410. So, we divide the GA's computation times by two in our reporting.

We provide computational results for the adaptive-fixing heuristic H2, ADP-H2 (ADP-H2 under no lag-variable-fixing), ADP-H2* (ADP-H2 under lag-variable-fixing with `lag-time`=100) and ADP-H2** (ADP-H2 under lag-variable-fizing with `lag-time`=200) on Chu and Beasley's data set in Table 5. We denote by PI(H2)* and PI(H2)** the percentage improvements of ADP-H2* and ADP-H2** over H2, respectively. We observe H2 generated good solutions (with an average PE of 1.20) in very small computation times (with an average T of 0.3 CPU seconds). ADP base-heuristic algorithms (ADP-H2, ADP-H2* and ADP-H2**) significantly improve the performance of H2 to construct higher quality of solutions with an average PI of 28.97. Average PEs of ADP-H2, ADP-H2* and ADP-H2** are 0.76, 0.80 and 0.74, respectively. Interestingly, ADP-H2** generated, on average, higher quality of solutions than the ones by ADP-H2 at smaller computation times. Even though, GA (with an average PE of 0.56) provides *slightly* better solutions than ADP-H2s, its average computation time, 1,287.15, is an order of magnitude larger than the one of ADP-H2s, 87.06 (average of average times T(ADP-H2), T(ADP-H2*) and T(ADP-H2**)). Overall, ADP-H2 algorithms achieved high quality solutions to some of the test problems from the literature in short computation times.

[2] http://mscmga.ms.ic.ac.uk/jeb/orlib/mknapinfo.html

			GA		H2		ADP-H2		PI(H2)	ADP-H2*		PI(H2)*	ADP-H2**		PI(H2)**
m	n	τ	T	PE	T	PE	T	PE		T	PE		T	PE	
30	250	0.25	749.75	1.19	0.21	2.73	69.64	1.61	39.39	25.80	1.74	34.87	31.68	1.64	38.51
		0.50	990.03	0.53	0.19	1.31	63.62	0.69	44.59	23.44	0.82	35.05	29.71	0.72	42.22
		0.75	1,220.70	0.61	0.20	0.76	55.51	0.58	22.15	20.39	0.53	26.98	26.20	0.54	26.62
30	500	0.25	1,218.85	0.61	0.40	1.39	284.73	0.98	25.50	74.23	0.97	25.03	99.90	0.81	37.45
		0.50	1,599.45	0.26	0.41	0.65	241.47	0.43	28.41	63.30	0.46	24.19	87.97	0.45	25.77
		0.75	1,944.10	0.17	0.39	0.35	223.90	0.29	15.07	61.48	0.30	14.60	84.08	0.29	15.09

Table 5: Comparison of H2, ADP-H2, ADP-H2* and ADP-H2** with Chu and Beasley's GA on their test problems.

5.5 Comparison of the most promising ADP algorithm with CPLEX

In this section, we study the performance of CPLEX 6.0 [14] on the same randomly generated problems. We also present minimum, average and maximum computation times (Ts) and minimum, average and maximum percentage deviations (PEs) of the most promising ADP algorithm, ADP-H2 (ADP with adaptive-fixing heuristic), for each problem instance.

CPLEX allows a variety of options in its tree search such as node selection strategy, cover cuts generation, clique cuts generation and heuristic frequency. Different settings might change the performance of CPLEX significantly in terms of both solution time and quality. In our computations, we study the performance of CPLEX under both default setting (denoted by CPLEX-D), and new settings (denoted by CPLEX-N), which we describe below. In either case, we use the following setup in our study. We set CPLEX tolerance parameter `CPX_PARAM_EPGAP` to PE(ADP-H2)/100 to achieve the same quality solutions as those obtained by ADP-H2. We also set the tree-memory limit and time-limit to 250 MB and 6,000 CPU seconds, respectively. Once CPLEX terminates (possibly by reaching the specified time-limit), we retrieve the solution value and calculate its percentage deviation from the LP objective value. Let C denote the number of instances for which CPLEX found a solution within PE(ADP-H2) below the specified time-limit. For these instances, we use T and PE to denote average CPLEX computation time and percentage deviations from LP, respectively. We denote by C* the number of instances for which CPLEX could not find a solution within PE(ADP-H2) below the time-limit. T* and PE* are used to report average CPLEX time and percentage deviations.

Under CPLEX with new settings (CPLEX-N), we turned off both clique and cover cut generations in comparison to default setting in which cuts are automatically generated. We turned on the periodic heuristic and set the parameter `CPX_PARAM_HEURFREQ` to 25 to help CPLEX find more feasible solutions. We also set node selection strategy to depth first search allowing CPLEX to dive deeper in the tree where integer feasible solutions are more likely to be found. We provide our computational results both for CPLEX-D and CPLEX-N in Table 6 where we put the symbol - (dash) when CPLEX results do not apply. For the instances where both CPLEX-D and CPLEX-N find solutions within PE(ADP-H2) below the time-limit of 6,000 CPU seconds, we observe that both CPLEX-D (with average T of 565.84) and CPLEX-N (with average T of 592.19) use significantly larger computation time to reach the same quality solutions as the ones obtained by ADP-H2 (with an average T of 7.52). In addition, the quality of solutions obtained by CPLEX-D (with average PE* 3.9388) and CPLEX-N (with average PE* 2.3508) are lower than the ones by ADP-H2 (average PE 0.9337) when CPLEX terminates due to time-limit 6000 CPU seconds, where average time of ADP-H2 is 380.26 CPU seconds. Overall, CPLEX-N returned better solutions than the ones by CPLEX-D. For those instances that are solved both by CPLEX-D (44 out of 120) and CPLEX-N (45 out of 120) below the time-limit, average (T and PE) of CPLEX-D and CPLEX-N are (565.84 and 1.6515) and (1,312.68 and 1.0960), respectively. Furthermore, for those instances for which CPLEX-D and CPLEX-N reach time-limit, average PE* of CPLEX-D and CPLEX-N are 3.9388 and 2.3508, respectively. We also observe that ADP-H2 performed quite robustly in terms of solution quality and time, since T and PEs of ADP-H2 did not deviate significantly from one another for a certain problem size (m, n).

			T(ADP-H2)			PE(ADP-H2)			CPLEX-D								CPLEX-N							
m	n	type	min	avg.	max	min	avg.	max	C	T	# of nodes	PE	C*	T*	# of nodes*	PE*	#	T	# of nodes	PE	#*	T*	# of nodes*	PE*
10	100	UC	0.16	0.72	1.21	0.4804	0.8947	1.1321	10	12.20	13,722.40	0.5796	-	-	-	-	10	1.35	2,959.30	0.9363	-	-	-	-
10	100	WC	0.07	1.26	2.49	0.7064	1.1919	1.5768	10	1,394.80	1,312,296.80	1.7911	-	-	-	-	10	377.93	1,249,720.40	1.3865	-	-	-	-
10	100	SC	0.04	2.27	2.73	1.0694	1.4825	2.0281	10	1,320.00	1,123,245.80	1.7442	-	-	-	-	5	761.92	1,285,415.40	1.6789	5	time-limit	10,022,713.60	1.5523
50	100	UC	0.98	5.99	10.23	0.8528	1.3285	1.8455	10	100.40	53,161.90	1.6264	-	-	-	-	10	8.91	14,690.00	1.4514	-	-	-	-
50	100	WC	4.91	27.37	36.68	1.2837	2.2910	3.1162	4	1,810.75	527,833.25	2.5164	6	time-limit	1,277,742.00	3.7080	4	1,810.83	1,108,184.50	2.2101	6	time-limit	3,365,328.00	3.3352
50	100	SC	62.98	69.74	76.18	2.9389	3.1329	3.4859	0	-	-	-	10	time-limit	819,612.50	5.7340	0	-	-	-	10	time-limit	6,626,481.20	4.2177
100	500	UC	13.64	58.98	89.99	0.2101	0.3215	0.4880	0	-	-	-	10	time-limit	480,048.00	3.3196	3	3,903.07	580,353.00	0.3566	7	time-limit	845,285.00	0.6469
100	500	WC	56.09	238.79	335.12	0.3138	0.6091	0.7491	0	-	-	-	10	time-limit	141,415.30	3.9888	1	3,635.98	731,705.00	0.3316	9	time-limit	1,043,239.44	2.8808
100	500	SC	692.50	774.21	891.98	0.8633	0.9513	1.0342	0	-	-	-	10	time-limit	46,111.10	4.3003	0	-	-	-	10	time-limit	1,983,315.30	2.5962
100	1,000	UC	115.45	154.82	192.74	0.1424	0.1710	0.2166	0	-	-	-	10	time-limit	195,550.80	3.6750	0	-	-	-	10	time-limit	738,504.80	1.0233
100	1,000	WC	137.03	520.54	732.07	0.1431	0.3361	0.4143	0	-	-	-	10	time-limit	50,164.90	3.9381	0	-	-	-	10	time-limit	1,183,343.33	2.6974
100	1,000	SC	1,086.36	1,222.69	1,323.54	0.4357	0.4651	0.4953	0	-	-	-	10	time-limit	20.090.40	2.8485	1	1.41	51.00	0.4169	9	time-limit	2,000,927.33	2.2078

Table 6: Comparison of ADP-H2 with CPLEX-D and CPLEX-N.

6 Summary and Conclusions

Based on the computational evidence from Section 5, we offer the following conclusions:

1. Although not the main focus of the paper the adaptive fixing heuristic for the MKP was surprisingly strong. It is fast and accurate.

2. The ADP base-heuristic methodology is encouraging. It improves the solution quality, it is flexible as it works with an arbitrary base-heuristic. The computational evidence suggests that the heuristic ADP-H2 competes successfully with state of the art heuristic and commerical software.

3. Approximating the value function with either parametric and nonparametic methods is not competitive with the best heuristic methods for MKP including base-heuristic methods.

In summary, the computational evidence suggests that the ADP base-heuristic approach for the MKP seems an attractive alternative to existing methodologies as it produces near optimal solutions fast and robustly. In Bertsimas and Demir [6] we apply ADP based approaches on generic binary integer programming problems.

References

[1] R. Aboudi and K. Jörnsten. Tabu search for general zero-one integer programs using the pivot and complement heuristic. *ORSA Journal on Computing*, 6:82–93, 1984.

[2] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, N.J., 1957.

[3] D. P. Bertsekas and D. A. Castañon. Rollout algorithms for stochastic scheduling problems. Technical Report LIDS-P-2413, MIT, 1997.

[4] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena-Scientific, Belmont, 1996.

[5] D. P. Bertsekas, J. N. Tsitsiklis, and C. Wu. Rollout algorithms for combinatorial optimization. *Journal of Heuristics*, 3:245–262, 1997.

[6] D. Bertsimas and R. Demir. An approximate dynamic programming approach to binary integer programming. Technical report, 2001.

[7] D. J. Bertsimas and I. Popescu. Revenue management in a dynamic network environment. Technical report, MIT, ORC, 2000.

[8] D. J. Bertsimas, C. P. Teo, and R. Vohra. Greedy, randomized and approximate dynamic programming algorithms for facility location problems. Technical report, MIT, ORC, 1998.

[9] D. J. Bertsimas and J. Tsitsiklis. *Introduction to Linear Optimization*. Athena-Scientific, Belmont, 1997.

[10] J. D. Christodouleas. *Solution Methods for Multiprocessor Network Scheduling Problems with application to railroad operations*. PhD thesis, MIT, ORC, 1997.

[11] P. C. Chu and J. E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4:63–86, 1998.

[12] L. Cooper and M. W. Cooper. *Introduction to Dynamic Programming*. Pergamon Press, Elmsford, New York, 1981.

[13] S. DeVries and R. Vohra. Combinatorial auctions: A survey, 2000.

[14] ILOG Inc. CPLEX Division. *Using the CPLEX Callable Libraries, Version 6.0*. 889 Alder Avenue, Suite 200, Incline Village, NV 89541, 1998.

[15] M. E. Dyer and A. M. Frieze. Probabilistic analysis of the multidimensional knapsack problem. *Mathematics of Operations Research*, 14:162–176, 1989.

[16] J. Fan and I. Gijbels. *Local Polynomial Modeling and Its Applications*. Chapman & Hall, London, UK, 1996.

[17] C. Ferreira, M. Grotschel, S. Kiefl, C. Krispenz, A. Martin, and R. Weismantel. Some integer programs arising in the design of mainframe computers, 1993.

[18] A. Freville and G. Plateau. Heuristics and reduction methods for multiple constraints 0-1 linear programming problems. *European Journal of Operational Research*, 24:206–215, 1986.

[19] A. Freville and G. Plateau. An efficient preprocessing procedure for the multidimensional 0-1 knapsack problem. *Discrete Applied Mathematics*, 48:189–212, 1994.

[20] A. M. Frieze and M. R. B. Clarke. Approximation algorithms for the m-dimensional 0-1 knapsack problem: Worst case and probabilistic analysis. *European Journal of Operational Research*, 15:100–109, 1984.

[21] B. A. McCarl G. A. Kochenberger and F. P. Wymann. A heuristic for general integer programming. *Decision Sciences*, 5:36–44, 1974.

[22] B. Gavish and H. Pirkul. Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality. *Mathematical Programming*, 31:78–105, 1985.

[23] P. C. Gilmore and R. E. Gomory. The theory and computation of knapsack functions. *Operations Research*, 14:1045–1075, 1966.

[24] F. W. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8:156–166, 1977.

[25] F. W. Glover and G. A. Kochenberger. Critical event tabu search for multidimensional knapsack problems. In *Meta-Heuristics: Theory and Applications*, pages 407–427. Kluwer Academic Publishers, Boston, 1996.

[26] S. Hanafi and A. Freville. An efficient tabu search approach for the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, 106:659–675, 1998.

[27] D.S. Hochbaum. *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, 1996.

[28] O. H. Ibarra and C. E. Kim. Fast approximation algorithms for the knapsack and sum of subset problem. *Journal of ACM*, 22:463–468, 1975.

[29] A. J. Kleywegt, V. S. Nori, and M. W. P. Savelsbergh. A computational approach for the inventory routing problem. Technical report, School of Industrial and Systems Engineering, Georgia Institute of Technology, 1998.

[30] G. Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59:345–358, 1992.

[31] J. S. Lee and M. Guignard. An approximate algorithm for multidimensional zero-one knapsack problems- a parametric approach. *Management Science*, 34:402–410, 1988.

[32] A. Løkketangen and F. W. Glover. Solving zero-one mixed integer programming problems using tabu search. *European Journal of Operational Research*, 106:624–658, 1998.

[33] R. Loulou and E. Michaelides. New greedy-like heuristics for the multidimensional 0-1 knapsack problem. *Operations Research*, 27:1101–1114, 1979.

[34] M. J. Magazine and M. S. Chern. A fully polynomial approximation schemes for multidimensional knapsack problem. *Mathematics of Operations Research*, 9:244–247, 1984.

[35] M. J. Magazine and O. Oguz. A heuristic algorithm for the multidimensional zero-one knapsack problem. *European Journal of Operational Research*, 16:319–326, 1984.

[36] R. Mansini and M. G. Speranza. A multidimensional knapsack model for the asset-backed securitization. Technical report, 1997.

[37] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementation.* Wiley, Chichester, England, 1990.

[38] M. Meanti, A. H. G. Rinnooy Kan, L. Stougie, and C. Vercellis. A probabilistic analysis of the multiknapsack value function. *Mathematical Programming*, 46:237–247, 1990.

[39] T. L. Morin. *Dynamic Programming and Its Applications*, chapter in Computational Advances in Dynamic Programming edited by M. L. Puterman, pages 53–90. Academic Press, New York, 1978.

[40] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* Wiley, New York, 1988.

[41] C. C. Peterson. Computational experience with variants of the Balas algorithm applied to the selection of research and development projects. *Management Science*, 13:736–750, 1967.

[42] N. Piersma. *Combinatorial Optimization and Empirical Processes.* PhD thesis, The Tinbergen Institute, 1993.

[43] W. P. Powell and J. A. Shapiro. A dynamic programming approximation for ultra large-scale dynamic resource allocation problems. Technical Report SOR-96-06, Statistics and Operations Research, Princeton University, 1996.

[44] M. H. Rothkopf, A. Pekec, and R. M. Harstad. Computationally manageable combinatorial auctions. Technical Report 95-09, 19, 1995.

[45] B. Van Roy, D. Bertsekas, Y. Lee, and J. N. Tsitsiklis. A neuro-dynamic programming approach to retailer inventory management. *IEEE Transactions on Control*, 16:1–23, 1998.

[46] S. Sahni. Approximate algorithms for the 0-1 knapsack problem. *Journal of ACM*, 22:115–124, 1976.

[47] U. K. Sarkar, P. P. Chakrabarti, S. Ghose, and S. C. DeSarkar. Improving greedy algorithms by lookahead-search. *Journal of Algorithms*, 16:1–23, 1994.

[48] N. Secomandi. *Exact and heuristic dynamic programming algorithms for the vehicle routing problem with stochastic demand.* PhD thesis, Faculty of the College of Business Administration, University of Houston, 1998.

[49] S. Senju and Y. Toyoda. An approach to linear programming with 0-1 linear variables. *Management Science*, 15:196–207, 1968.

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

[51] K. Szkatuła. The growth of multi-constraint random knapsacks with various right-hand sides of the constraints. *European Journal of Operational Research*, 73:199–204, 1994.

[52] A. Thesen. A recursive branch and bound algorithm for multidimensional knapsack problem. *Naval Research Logistics*, 22:341–353, 1975.

[53] Y. Toyoda. A simplified algorithm for obtaining approximate solutions to zero-one programming problems. *Management Science*, 21:1417–1427, 1975.

[54] H. M. Weingartner. Capital budgeting of interrelated projects: survey and synthesis. *Operations Research*, 12:485–516, 1966.

[55] H. M. Weingartner and D. N. Ness. Methods for the solution of the multidimensional 0/1 knapsack problem. *Operation Research*, 15:83–103, 1967.

[56] E. Wike. Supply chain optimization: Formulations and algorithms. Master's thesis, MIT, ORC, 1998.