Author’s Accepted Manuscript

Solving the multidimensional knapsack problems with generalized upper bound constraints by the adaptive memory projection method

Vincent C. Li, Yun-Chia Liang, Haun-Fu Chang

PII: S0305-0548(11)00304-2
Reference: CAOR 2947

To appear in: Computers & Operations Research

Received date: 14 February 2011
Revised date: 30 August 2011
Accepted date: 17 October 2011

Cite this article as: Vincent C. Li, Yun-Chia Liang and Haun-Fu Chang, Solving the multidimensional knapsack problems with generalized upper bound constraints by the adaptive memory projection method, Computers & Operations Research, doi:10.1016/j.cor.2011.10.016

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting galley proof before it is published in its final citable form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.
Solving the Multidimensional Knapsack Problems with Generalized Upper Bound Constraints by the Adaptive Memory Projection Method

Vincent C. Li¹, Yun-Chia Liang²*, Haun-Fu Chang²

Abstract

An adaptive memory projection (referred as AMP) method is developed for multidimensional knapsack problems (referred as the MKP) with generalized upper bound constraints. All the variables are divided into several generalized upper bound (referred as GUB) sets and at most one variable can be chosen from each of the GUB sets. The MKP with GUBs (referred as the GUBMKP) can be applied to many real-world problems, such as capital budgeting, resource allocation, cargo loading, and project selection. Due to the complexity of the GUBMKP, good metaheuristics are sought to tackle this problem.

The AMP method keeps track of components of good solutions during the search and creates provisional solution by combining components of better solutions. The projection method, which can free the selected variables while fixing the others, is very useful for metaheuristics, especially when tackling large-scale combinatorial optimization. In this paper, the AMP method is implemented by iteratively using critical event tabu search as a search routine, and CPLEX in the referent optimization stage. Variables that are strongly determined, consistent, or attractive, are identified in the search process. Selected variables from

---

¹Partially Supported by National Science Council, Taiwan: 96-2221-E-259-004.
²Corresponding author. Fax: 886-3-4638907, TEL: 886-3-4638800#2521
Email addresses: vincentc11@mail.nctu.edu.tw (Vincent C. Li), ycliang@saturn.yzu.edu.tw (Yun-Chia Liang), bythesay588@hotmail.com (Haun-Fu Chang)
¹Department of Business Administration, National Chiayi University, Chiayi, Taiwan
²Department of Industrial Engineering and Management, Yuan Ze University, Taoyuan, Taiwan
this pool are fed into CPLEX as a small subproblem. In addition to the diversification effect within critical event tabu search, the pseudo-cut inequalities and an adjusted frequency penalty scalar are also applied to increase opportunities of exploring new regions.

This study conducts a comprehensive sensitivity analysis on the parameters and strategies used in the proposed AMP method. The computational results show several variants of the AMP method outperforms the tight oscillation method in the literature of GUBMKP. On average, consistent variables tend to perform best as a pure strategy. A pure strategy equipped with local search can lead into even better results. Last but not least, testing different types of variables in the referent optimization stage before selecting just one of the pure strategies is found to be very helpful.

Keywords: heuristics, multidimensional knapsack, generalized upper bound, critical event tabu search, adaptive memory projection

1. Introduction

This paper considers the multidimensional knapsack problem (commonly known as the MKP) with generalized upper bound (referred to GUB hereafter) constraints as shown in the following formulation: A set of $n$ items, $N \equiv \{1, \cdots, n\}$, should be packed to maximize the overall profit where item $j$ can contribute $c_j$ (Eq. 1), such that the capacity limitation $b_i, i \in M \equiv \{1, \cdots, m\}$, of each of the $m$ knapsack constraints will be met as selecting item $j$ will use $a_{ij}$ units of resource $i$ (Eq. 2). Items performing the same function are grouped into one category although the contribution of each of them are different generally. All the items are partitioned into $g$ mutually exclusive subsets and at most one item can be chosen from each of these subsets (Eqs. 3, 5, and 6). If an item $j$ is chosen, then the associated variable $x_j$ equals 1; otherwise it equals 0 (Eq. 4). All the parameters of the objective function and the constraints are assumed to be nonnegative (Eq. 7).
Formulation 1 (GUBMKP).

\[
\begin{align*}
\text{Maximize} & \quad \sum_{j \in N} c_j x_j & \quad (1) \\
\text{Subject to} & \quad \sum_{j \in N} a_{ij} x_j \leq b_i, \quad \forall i \in M \equiv \{1, \cdots, m\}, & \quad (2) \\
& \quad \sum_{j \in N_h} x_j \leq 1, \quad \forall h \in G \equiv \{1, \cdots, g\}, & \quad (3) \\
& \quad x_j \in \{0, 1\}, \quad \forall j \in N \equiv \{1, \cdots, n\}, \quad (4) \\
\text{where} & \quad \bigcup_{h=1}^{g} N_h = N, & \quad (5) \\
& \quad N_p \cap N_q = \emptyset \quad \forall p, q \in G, p \neq q, & \quad (6) \\
& \quad a_{ij}, c_j, b_i \geq 0 \quad \forall i \in M, j \in N. & \quad (7)
\end{align*}
\]

The formulation of the MKP consists of Eqs. 1, 2, 4, and 7. The GUBMKP model can be obtained by adding the GUB constraints (Eq. 3) with the properties stated in Eqs. 5 and 6. Following the first application on capital budgeting [1], the MKP has been used to model various application such as project selection, cutting stock, cargo loading, and combinatorial auctions [2, 3, 4, 5]. The GUBMKP can be transformed to the multiple-choice multidimensional knapsack problem (referred to the MMKP) by adding a slack variable in each of the GUB constraints (Eq. 3) and making it an equality constraint. In fact, the MMKP is a combination of the MKP and the multiple-choice knapsack problem (MCKP). The MCKP has many applications such as menu planning and capital budgeting where there is only one resource constraint and candidates to be chosen fall into disjoint subsets (e.g., [6, 7]). With this transformation, an active slack variable (i.e., it equals 1) indicates that the corresponding GUB constraint is not tight. Therefore, the GUBMKP can be seen as a relaxation of the MMKP. The MMKP has been used to model configuration problems that involve options [8] (e.g., modeling real-time multimedia domain [9] and building a utility model for a multisession adaptive multimedia system [10, 11]), whereas the GUBMKP formulation has been used to model a striking asset allocation problem [12] and a surveillance system for port and waterway security [13].
The MKP is NP-Hard [14], so is its close variant GUBMKP. Polynomial algorithms to solve it to optimality can not be found unless P = NP [15]. Metaheuristics have been shown effective in approximating optimal solutions for many difficult large-scale optimization problems. This paper utilizes the idea of adaptive memory projection proposed by Glover [16], where collections of important variables are identified by the underlying search heuristic, and the corresponding subproblem is solved by CPLEX, iteratively.

The remainder of this paper is organized as follows. Section 2 reviews the literature most relevant to the GUBMKP. Section 3 discusses the AMP methodology and the details of implementation. Computational results are shown in Section 4. Conclusions and future directions are discussed in Section 5.

2. Literature Review

The knapsack problem (KP) is one of the most well-known combinatorial problems. There are many variants of the KP. Readers can refer to Kellerer et al. [17] and Martello and Toth [14] for a full-scale presentation of most of the methods and techniques related to the KP. In our review, we focus on the MKP, MMKP, and GUBMKP.

The MKP has been investigated extensively by various approaches including but not limited to branch-and-bound, dynamic programming, greedy heuristics and metaheuristics. Fréville [18] provides a recent review of the MKP. The theoretical properties of the MKP, in particular, those relevant to surrogate and composite duality, can be found in Fréville and Hanafi [19].

For the MMKP, similar to what has been done to the MKP [20, 21], Moser et al. [9] adopt the Lagrange multiplier approach to solve this class of problem. Toyoda [22] proposes a greedy algorithm that aggregates resource consumption of selecting an item for the MKP. In Toyoda’s method, an item with relatively low profit and high aggregate resource consumption is penalized for selection. This concept is embraced by Khan et al. [23] for selecting items to pick, and then a swap method is applied iteratively to improve the MMKP solution. Hifi et
al. [24] propose a guided local search in which the solution trajectory is altered in a way that penalizes bad attributes of previously visited solutions. As an extension to the work above, Hifi et al. [25] use a reactive local search method that explicitly checks the repetition of solutions. Degrading and deblocking strategies are employed in order to escape the local optima and diversify the search. Parra-Hernandez and Dimopoulos [8] reduce the MMKP to the MKP, and utilize a method related to Pirkul’s [26] surrogate constraint approach to solve the reduced problem. Akbar et al. [27] transform the multidimensional resource consumption to single dimension, and then construct the convex hulls to reduce the search space. Hiremath and Hill [28] explore the problem structure of the available MMKP test problems, and present a more diverse set of the MMKP test instances. Cherfi and Hifi [29] design a column generation based method that consists of a rounding solution stage and a restricted exact solution procedure. Metaheuristics have also been applied to solve the MMKP. For example, Hiremath [30] proposes a tabu search method that utilizes a sequential fan candidate list for the MMKP. His method collects a master list of elite solutions obtained from a first level tabu search, then a fan of solution stream is generated for each solution in the master list. This process is also called beam search [31]. In addition to heuristic or metaheuristic approaches, exact algorithms have been designed for the MMKP. For examples, Sbihi [32] suggests a branch-and-bound algorithm with the best first search strategy that can be run in parallel; Ghasemi and Razzazi [33] develop an approximate core of the MMKP, and utilize the core to solve the problem exactly.

The GUBMKP can be converted to the MMKP to use the heuristics designed specifically for the MMKP. Nevertheless, in order to gain more insight to the GUBMKP, we propose metaheuristics for the problem class explicitly. Metaheuristics have been shown very effective in dealing with hard combinatorial problems. A variety of metaheuristics have been developed for the MKP, such as simulated annealing (e.g., [34]), genetic algorithms (e.g., [35]), and tabu search (e.g., [36]).

Glover and Kochenberger [37] design a critical event tabu search (referred
to CETS) for the MKP. CETS navigates around the feasibility boundary by iteratively using a constructive phase (add items) and a destructive phase (remove items), with variable depths of oscillation (the number of constructive/destructive moves after crossing the feasibility boundary). Solutions obtained immediately before or after crossing the feasibility boundary are referred to critical solutions and the moves that change the feasibility are called critical events. Attributes of critical solutions are recorded in the tabu memory to avoid cycling, and duplication of critical solutions, whereas the tabu list is to be checked near the turn-around points, and the constructive and destructive phases are switched after reaching these points. Nevertheless, tabu rules may be too restricted for selecting search moves, instead of forbidding the use of tabu-list variables, a penalty approach can be used that allows the use of such a variable but gives a penalty in its choice-rule evaluation. Local search can be used to strengthen the strategic oscillation, and they can be launched at critical solutions. Hanafi and Fréville [36] also design a similar tabu search approach which utilizes strategic oscillation with greedy algorithms for the MKP and obtain even better results.

Li and Curry [38] apply CETS to the GUBMKP by tackling the GUB constraints separately. They discuss various choice rules and show the merits of using surrogate constraints over a specific form of Lagrangian relaxation that relaxes all the knapsack constraints. Stronger surrogate constraints based on Glover [39] are obtained.

In order to exploit good critical solutions and good local optimal solutions, Li [40] applies a tight oscillation method to the GUBMKP as an improvement procedure. Similar to the CETS, this method also navigates around the feasibility boundary but the span value is 0. That is, the method switches its direction immediately after crossing the feasibility boundary from either the feasible side or the infeasible side. This strengthening subroutine is terminated as soon as it finds a possible duplicated solution. In order to detect duplication, the search history is captured by utilizing hashing functions (See Carlton and Barnes [41]). The choice rule used for this method incorporates both the change on the objective function value and the infeasibility change. This method enhances the
computational results for the GUBMKP instances significantly compared to Li and Curry [38].

Li [42] applies a perturbation-guided oscillation strategy for intensification on solving the GUBMKP. Within the CETS framework, this method perturbs the right-hand-side of the knapsack constraints when searching for trial solutions from critical events. The feasibility is checked and restored upon finishing the perturbation, and a local search is done in order to improve the feasible solution.

3. Adaptive Memory Projection Method for the GUBMKP

Large-scale combinatorial problems can be very difficult to solve. Many heuristics use “problem reduction” techniques to tackle this issue. For example, Pirkul [26] utilizes surrogate constraint relaxation to capture the MKP. On the other hand, the idea of iteratively holding some variables fixed at some certain values while varying others, have appeared in various optimization techniques. For instance, Hill et al. [43] propose a heuristic based on Lagrangian relaxation which iteratively deals with MKP subproblems with less variables. This idea is also known as “large neighborhood search” in the constraint programming community [44]. Moreover, this concept is broadly adopted in projection mapping, an important technique in mixed integer programming and nonlinear programming [16]. Glover [16] outlines a generic method of adaptive memory projection (AMP) for pure and mixed integer programming. The AMP method originates from three concepts of tabu search including (1) strongly determined and consistent variables, (2) intensification/diversification tradeoffs, and (3) persistent attractiveness. A detailed description of these concepts can be found in Glover [16].

The AMP method used in this paper iteratively uses critical event tabu search [37, 36, 38] to identify sets of “important variables” to form a small subproblem and then uses CPLEX to solve the subproblem to optimality. The values of these variables in the corresponding optimal solution will be kept in combination with the set of fixed variables. Based on a heuristic fixation of
variables, Wilbaut et al. [45] propose an iterative scheme to solve the MKP. In their study, variables are dynamically fixed temporarily or permanently through iterative LP relaxations to reduce the size of the problem that is to be solved exactly. Moreover, Hanafi and Wilbaut [46] use a series of linear programming relaxations to solve a series of small sub-problems for the MKP.

Adapted from the outline of the AMP method proposed in Glover [16], our implementation for the GUBMKP can be described as follows.

**Step 1** (Initialization): Start from the solution where all variables are set to 0.

**Step 2** (Search Phase): The critical event tabu search is applied to identify collections of important variables including strongly determined variables, consistent variables, persistent attractive variables, and conditionally attractive variables.

**Step 3** (Referent-Optimization Phase): Select a subset of the variables recorded in Step 2. Notice some variables can appear in more than one subset. Set the selected variables free while fix the remaining variables at the values they obtained in the best solution identified in Step 2. CPLEX is then used to solve this “referent-optimization” subproblem.

**Step 4** (Stopping Criteria): If the run time limit or the number of iteration limit is reached, then stop; otherwise re-launch the search (Step 5).

**Step 5** (Relaunch): Starting from the solution obtained in Step 3, perform a new heuristic search with the restrictions imposed by the long-term tabu memory of critical solutions.

Several remarks are as follows.

1. In initialization, all variables are set to 0.
2. The oscillation of the CETS is controlled by the span limit parameter as specified in Li and Curry [38].
3. In the search phase, the choice rule is based on bang-for-buck (benefit/cost, referred to B/C) ratios between the objective function value and the surrogate constraint that is adaptive to the current solution. The generation
of surrogate constraint is based on “type 2 normalization rule” proposed
by Glover [47]. The steps are described as follows.

(a) For each of the constraints, update the right-hand-side (RHS) value
according to the current solution (assignment). The variables that
have been assigned values are removed from the constraint.
(b) Normalize each constraint by dividing through the updated RHS
value.
(c) For each constraint, multiply it by the sum of left-hand-side (LHS)
coefficients when add is considered; divide it by the sum of LHS
coefficients when drop is considered.
(d) Sum up all the constraints to form the surrogate constraint.

These ratios are penalized by the short-term and long-term critical-solution-
tabu-memory (as shown in the next remark) at the first 2 to 4 moves upon
switching between the constructive and destructive phases of CETS. The
duration of checking the tabu status is varied systematically by repeatedly
increasing from 2, 3, and then to 4, and then decreasing back to 3 and
then 2.

4. According to Li and Curry [38], the B/C ratio can be adjusted by two
terms: the recency penalty and the frequency penalty. The recency penalty
of a candidate variable \( x_j \) equals \( r_{\text{max}} \cdot TabuR_j \) where \( r_{\text{max}} \)
stands for the maximum B/C ratio of the eligible variables, and \( TabuR_j \)
represents the recency count of variable \( x_j \). The frequency penalty is evaluated as Eq. 8.

\[
\frac{r_{\text{max}} \cdot TabuF_j}{\text{the index of the current iteration} \cdot \text{frequency penalty scalar}}, \tag{8}
\]

where \( TabuF_j \) represents the frequency count of variable \( x_j \). Notice the
tabu list keeps a collection of the most recent critical solutions. The
recency and frequency vectors are obtained by summing up the solution
vectors in the current tabu list, and all the critical solutions from the past,
respectively. For diversification purpose, the frequency penalty scalar \( (p_f) \)
is dynamically adjusted by the duplication rate of critical solutions \( (r_d) \).
Specifically,
5. To introduce diversification within Step 3, two of the pseudo-cuts proposed by Glover [16] are introduced. The idea of using pseudo-cuts is to diversify the optimal solution of the sub-problem in the referent optimization phase. By observing the differences between the initial solution and the best solution, variables are categorized into three subsets $S_A$, $S_B$, and $S_C$, as described in the following:

- $S_A$: variables $(x_1, x_2, \cdots, x_p)$ that increase from 0 to 1.
- $S_B$: variables $(y_1, y_2, \cdots, y_q)$ that decrease from 1 to 0.
- $S_C$: variables $(z_1, z_2, \cdots, z_r)$ that do not change.

The sizes of $S_A$, $S_B$, and $S_C$ are $p$, $q$, and $r$, respectively. The first pseudo cut is defined as Eq. 9.

$$p_f = \begin{cases} 
100,000 & \text{if } r_d \leq 0.001, \\
10,000 & \text{if } 0.001 < r_d \leq 0.002, \\
1,000 & \text{if } 0.002 < r_d \leq 0.003, \\
100 & \text{if } 0.003 < r_d. 
\end{cases}$$

For a given value of $k$, Eq. 9 is the same as the local branching inequality proposed by Fischetti and Lodi [48]. This pseudo cut ensures that at least $k$ free variables in the referent-optimization phase obtain a value different from the best solution in the heuristic pass. In fact, Eq. 9 can be simplified to Eq. 10 as

$$(x_1 + x_2 + \cdots + x_p) - [(1 - y_1) + (1 - y_2) + \cdots + (1 - y_q)] \leq p + q - k. \quad (9)$$

where $k$ is a constant. As $k$ becomes larger, the resulting solution in the referent optimization phase will be subject to more deviation from the best solution in the heuristic pass. A possible value of $k$, based on Glover [16], can be $(p + q)/d$ where $d$ equals 2. The second pseudo cut,
shown in Eq. 11, forces \((r/d)\) free variables in \(S_C\) to change their values in the referent optimization phase.

\[
(1 - z_1) + (1 - z_2) + \cdots + (1 - z_r) \geq r/d. \tag{11}
\]

6. In Step 5 when re-launching the CETS after the referent optimization, all the free variables are set to the values obtained from Step 3 while all other variables are set to the values retrieved from the best solution prior to Step 3. To diversify the choice of free variables, all the free variables selected in the preceding referent-optimization phase is prohibited to change in the first 2, 3, or 4 moves. Incidentally, in case the aforementioned method is too restrictive, a penalty will be used instead.

7. Both short-term and long-term tabu memory are employed in Step 2. While the search is relaunched in Step 5, the short-term tabu memory is reset and the long-term memory lasts over relaunches.

3.1. Types of free variables

With this framework, different kinds of variables are collected as candidates of free variables to feed into the referent optimization. Glover [49] highlights the ideas of strongly determined variables and consistent variables. Glover [50] also points out persistent attractiveness of variables has the connotation of being “persistently unselected”. In our study, the collection and classification of free variables are described as follows.

**Consistent variables:** Maintain a list of \(\alpha\) solutions with best objective function values. After the search phase, all these \(\alpha\) solutions are summed over into a vector and variables with high frequency in the vector are chosen as consistent variables. The variable frequency is updated whenever a qualified solution replaces the worst solution with the lowest objective function value in the subset. The number of consistent variables is related to the size of the free variable set. The collection of consistent variables is denoted as \(V_C\).
Strongly determined variables: Variables with high frequency of being added or dropped at critical events influence the search trajectory in a more significant way. They are categorized as strongly determined variables and denoted by $V_{SD_1}$. If a variable has ever been added or dropped in the search process, it can also be considered a strongly determined variable. The collection of these variables is denoted as $V_{SD_2}$. In the implementation for this category of variables, the frequency information is updated whenever a variable is added or dropped.

Attractive variables: For each critical-event move, the heuristic maintains a list of variables with high evaluation (i.e., top $\gamma_{CA}$ choices) but not chosen. This set of variables are considered conditionally attractive candidates (denoted as $V_{CA}$). The heuristic maintains a list of variables with high evaluation (i.e., top $\gamma_{PA}$ choices) but not chosen eventually for each move (including critical events). Variables with high frequency in this subset are considered persistently attractive candidates (denoted as $V_{PA}$). When choosing free variables, variables in $V_{PA}$ have a lower priority of being chosen as free variables in comparison to variables in $V_{CA}$.

3.2. Ways to use free variables

There are many ways to exploit the subsets of variables described in Section 3.1 for the referent-optimization phase, and the following collection of experiments has been investigated.

1. Use one type of free variables only.

2. Use more than one type of free variables simultaneously, and the selected free variables form a single referent-optimization subproblem. This scheme is referred to the “merge strategy”.

3. Use more than one type of free variables separately, and each of the selected free variables form a single referent-optimization subproblem, respectively. Choose the subproblem with the best objective function value to move onto the next search phase. This scheme is referred as “competition strategy” in this paper. The idea is to exploit the efficiency of solving
small problems using the branch-and-bound approach. More computational effort is involved in this phase in order to get a better solution from the referent optimization.

In this study, a local search, described in the next subsection, is used after obtaining critical solutions as well as the results from the referent optimization. This helps to intensify the search. Moreover, the effect of identifying important variables from the local search is investigated as well. Important variables identified from the local search can be considered together with any of the five types of variables collected during the critical event tabu search. Different kinds of implementation are involved in this regard: for \( V_C \), the resulting solution from the local search competes with the elite solutions identified in CETS, whereas for other types of variables, a predetermined portion is collected from the local search part.

### 3.3. Local search

This paper proposes two swap-based local search methods (referred to \( LS_1 \) and \( LS_2 \)). Local search is launched from the feasible critical solutions as well as the resulting solution from the referent-optimization phase. The choice rules for prioritizing variables of these two swap procedures are described as follows.

**Add:** In local search \( LS_2 \), sort the variables according to their indexes. In local search \( LS_1 \), sort variables according to non-increasing bang-for-buck ratios between the objective function value and the surrogate constraint. The surrogate constraint here is generated by using the shadow prices of the original LP relaxation as the multipliers for each of the knapsack constraints. Therefore, this surrogate constraint does not change according to the solution.

**Drop:** Using the same surrogate constraint as in the Add part, variables are sorted according to non-decreasing bang-for-buck ratios in both \( LS_1 \) and \( LS_2 \).
4. Computational Results

The proposed method was coded in Visual C++ 2008 using a PC with Intel Core 2 Duo CPU E8500 3.16 GHz, 2.00 GB RAM and Windows XP operating system. Subproblems in the referent-optimization phase were solved using CPLEX 11.2.1.

4.1. Parameter settings

A computational run can be terminated whenever the solution gap, the time limit, or the number of relaunches has been reached. Based on our preliminary tests, the parameters used in the computational experiments are fixed as follows:

- To identify consistent variables, $60$ (i.e., $\alpha$ as mentioned in Section 3.1) best solutions are collected.
- Based on preliminary results, setting pseudo-cut threshold parameter $k$ in Eq. 10 to $(p + q)/4$ (i.e., $d = 4$) is found effective in obtaining better solutions and reducing the duplication of critical solutions. The value of $d$ in Eq. 11 is also set to 4. Detailed results can be found in Appendix A.
- $\gamma_{CA}$ and $\gamma_{PA}$ are set to 3 and 5, respectively, for conditionally attractive and persistent attractive variables (based on results of preliminary experiments), respectively.
- Except for the local search part, the surrogate constraint is updated every move.
- Tabu list size is 4 for the critical solution memory.

4.2. Test instances

Thirty instances used in Li [40] are tested in this study. There are three problem classes: $S$, $M$, and $L$, distinguished by the number of knapsack constraints and the number of GUB constraints. There are 10, 20, and 30 knapsack constraints and 20, 40, and 60 GUB constraints for problem classes $S$, $M$, and

14
The number of variables in problem classes $S$, $M$, and $L$ is 3,994, 8,321, and 15,365, respectively. Each problem class includes 10 variant instances with different percentages of the original variables. The percentages range from 10% to 100% with an increment of 10%. The convention of naming the test instances is as follows. Let $Dx$ be the name of a variant instance of problem $D$ by collecting approximately $x\%$ of the number of variables in problem class $D$, where $D$ can be $S$, $M$, or $L$, and $x$ can be one element from $\{10, 20, \ldots, 100\}$.

In this study, the base setting of the AMP experiments is to use CETS as the heuristic, consistent variable as the free variable type, and $LS_1$ as the local search. A variety of heuristic variants are designed and tested as shown in Table 1.

To evaluate the performance of heuristic variants, the best found values are compared with the results obtained from CPLEX MIP Solver. As it is very time-consuming to obtain the optimal solution for some of our test problems, the CPLEX best bound denoted as CBB hereafter within one-hour run time limit is used to approximate the optimal solution.

$$\text{gap(\%)} = \frac{\text{CBB} - \text{best found value of a heuristic variant}}{\text{CBB}} \times 100(\%)$$  \hspace{1cm} (12)

The gap is calculated using Eq. 12. By default, CPLEX uses a branch-and-cut algorithm to solve a series of relaxed LP problems. Denote the optimal LP objective value of the $i^{th}$ node in the branch-and-bound tree as $Z_i$, the optimal LP objective value of the original MIP maximization problem as $Z_0$, and the optimal MIP objective solution to be $Z^\ast$. It is straightforward that the LP relaxation of the original MIP problem provides an upper bound to the MIP maximization problem; that is, $Z_0 \geq Z^\ast$. In a maximization problem, it is clear that $Z_i \leq Z_0$ for all nodes $i$ as they are more restricted than the $0^{th}$ node. A node can be fathomed if its upper bound is higher than the incumbent solution (the best feasible integer solution found so far), or it is infeasible. Denote $R$ as the collection of nodes that have not been fathomed in the branch-and-bound
Table 1: List of experiments

<table>
<thead>
<tr>
<th>Heuristic variants</th>
<th>Collection of free variables</th>
<th>Local search</th>
<th>Parallel free variables</th>
<th>$V_C$ within local search</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1</td>
<td>$V_{SD1}$</td>
<td>LS$_1$</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>H2</td>
<td>$V_{SD2}$</td>
<td>LS$_1$</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>H3</td>
<td>$V_C$</td>
<td>LS$_1$</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>H4</td>
<td>$V_{FA}$</td>
<td>LS$_1$</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>H5</td>
<td>$V_{CA}$</td>
<td>LS$_1$</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>H6</td>
<td>$V_C$</td>
<td>LS$_2$</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>H7</td>
<td>$V_C$</td>
<td>LS$_1$</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>H8</td>
<td>$V_C$</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>H9</td>
<td>$V_C$, $V_{SD1}$, $V_{FA}$</td>
<td>LS$_1$</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>H10</td>
<td>$V_C$, $V_{SD2}$, $V_{FA}$</td>
<td>LS$_1$</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>H11</td>
<td>$V_C$, $V_{CA}$, $V_{CA}$</td>
<td>LS$_1$</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>H12</td>
<td>$V_C$, $V_{SD2}$, $V_{CA}$</td>
<td>LS$_1$</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>H13</td>
<td>$V_C$, $V_{SD1}$, $V_{FA}$</td>
<td>LS$_1$</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>H14</td>
<td>$V_C$, $V_{SD2}$, $V_{FA}$</td>
<td>LS$_1$</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>H15</td>
<td>$V_C$, $V_{SD1}$, $V_{CA}$</td>
<td>LS$_1$</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>H16</td>
<td>$V_C$, $V_{SD2}$, $V_{CA}$</td>
<td>LS$_1$</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
tree. Consequently, $Z^* \leq CBB = \max_{i \in R}(Z_i) \leq Z_0$. Thus it is more accurate to use CBB than $Z_0$ to calculate the solution gap, as CBB can only be closer to the objective function value of the optimal solution. Notice if the optimal solution (with a tolerance gap of 0.01%) can be obtained by CPLEX within one hour, then the CBB just equals the optimal objective value. The average gap (%) of each class of problems with different settings is summarized in Table 2 using one-minute CPU time as the stopping criterion. The details of sensitivity analysis are discussed in subsections 4.3 to 4.6.

### 4.3. The effectiveness of local search

Critical solutions in CETS are important as they are critical points when the feasibility is reversed. In order to achieve intensification, local search can be launched at critical solutions. To show the effectiveness of using local search in the AMP heuristic, a comparison is made among two local search variants, $LS_1$ (H3), $LS_2$ (H6), and another method that does not use any local search (H8) at all.
Swapping variables is considered in the local search by dropping a variable first, followed by adding another variable. Variables are sorted by the ratio of the objective coefficient to the surrogate constraint coefficient (referred to B/C ratio). To form the surrogate constraint, the multipliers of each of the constraints are equal to the corresponding shadow price obtained at the LP relaxation of the underlying problem. When dropping is considered, variables are sorted in a nondecreasing sequence according to their B/C ratios. Meanwhile, two choice rules of adding variables are considered. H3 (LS₁), the first strategy, chooses the variable with the largest B/C ratio within each GUB set when adding, while H6 (LS₂), the second strategy, chooses the variable with the smallest index within each GUB set.

The results show H3 performs better than H6 in most of the cases. Local search methods H3 and H6 both perform better than the one without using any local search (H8) (Figure 1). Moreover, the average gaps of H3 are also better than the ones of H6 (Table 3).

Figure 1: Effectiveness comparison of local search methods
Table 3: One-min. ave. gap (%) of each class of problems with different local search settings

<table>
<thead>
<tr>
<th>Heuristic variants</th>
<th>S-instances</th>
<th>M-instances</th>
<th>L-instances</th>
<th>All instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>H3 (LS₁)</td>
<td>0.70</td>
<td>2.02</td>
<td>3.92</td>
<td>2.22</td>
</tr>
<tr>
<td>H6 (LS₂)</td>
<td>0.67</td>
<td>2.04</td>
<td>4.17</td>
<td>2.30</td>
</tr>
<tr>
<td>H8 (no local search)</td>
<td>1.00</td>
<td>1.82</td>
<td>4.57</td>
<td>2.46</td>
</tr>
</tbody>
</table>

Table 4: One-min. ave. gap (%) of each class of problems with different types of free variables

<table>
<thead>
<tr>
<th>Heuristic variants</th>
<th>S-instances</th>
<th>M-instances</th>
<th>L-instances</th>
<th>All instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1 (V₁)</td>
<td>1.74</td>
<td>3.92</td>
<td>4.70</td>
<td>3.45</td>
</tr>
<tr>
<td>H2 (V₂)</td>
<td>1.05</td>
<td>3.49</td>
<td>4.62</td>
<td>3.05</td>
</tr>
<tr>
<td>H3 (V₃)</td>
<td>0.70</td>
<td>2.02</td>
<td>3.92</td>
<td>2.22</td>
</tr>
<tr>
<td>H4 (V₄)</td>
<td>1.05</td>
<td>3.47</td>
<td>4.61</td>
<td>3.04</td>
</tr>
<tr>
<td>H5 (V₅)</td>
<td>1.20</td>
<td>3.91</td>
<td>4.61</td>
<td>3.24</td>
</tr>
</tbody>
</table>

4.4. Comparison among different free variables with local search

To test the effectiveness of different categories of variables defined in Section 3.1, comparison is made among H1 (V₁), H2 (V₂), H3 (V₃), H4 (V₄) and H5 (V₅). It is clear that H3 (V₃) performs best among the five types of variables (Figure 2, Table 4) when the average performance of H1 (V₁) is inferior to the other four types of variables in all three-class (S, M, and L) instances. Meanwhile, when comparing the performance of two strongly determined variables V₁ and V₂, the one considering all moves, i.e. H2 (V₂), outperforms H1 (V₁) that considers the critical moves only. In addition, in the comparison of two attractive variables V₄ and V₅, similar outcomes can be observed that considering all moves (H4: V₄) performs better than considering just critical moves (H5: V₅). The consistent variable performs best on average, whereas some other types of free variables also have very good performance. These two observations lead us to examine the effect of using the consistent variable in combination of other types of free variables as discussed in Section 4.5.
4.5. The effectiveness of hybridizing free variables

Four kinds of mixtures are selected and named as H9, H10, H11, and H12. For each kind of mixture, two different approaches are used—"competition strategy" and "merge strategy". When using the competition strategy, a referent-optimization subproblem is solved using $V_C$ and other two types of free variables independently. When using the merge strategy, $V_C$ and other two types of variables form a pool of candidates without competing with each other. Table 5 compares the results of different kinds of mixtures including H9, H10, H11, and H12 using the competition strategy versus the merge strategy. The results show the competition strategy dominates the merge strategy. Tables 4 and 5 also indicate that the competition strategy enhances the solo performance of strongly determined variables (H1: $V_{SD1}$ and H2: $V_{SD2}$) and attractive variables (H4: $V_{PA}$ and H5: $V_{CA}$). However, the performance of competition strategies H9, H10, and H12 are still a little bit worse than H3 ($V_C$ only). Moreover, the combination of $V_C$, $V_{SD1}$, and $V_{CA}$ (H11) surprisingly provides better results than H3.
Table 5: Comparing one-min. ave. gap (%) of each class of problems when hybridizing free variables in two different versions

<table>
<thead>
<tr>
<th>Heuristic variants</th>
<th>S- instances</th>
<th>M- instances</th>
<th>L- instances</th>
<th>All instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>H9 (competition)</td>
<td>0.63</td>
<td>2.08</td>
<td>4.12</td>
<td>2.28</td>
</tr>
<tr>
<td>H9 (merge)</td>
<td>1.26</td>
<td>3.33</td>
<td>4.90</td>
<td>3.17</td>
</tr>
<tr>
<td>H10 (competition)</td>
<td>0.68</td>
<td>2.11</td>
<td>4.19</td>
<td>2.33</td>
</tr>
<tr>
<td>H10 (merge)</td>
<td>1.22</td>
<td>2.57</td>
<td>4.73</td>
<td>2.84</td>
</tr>
<tr>
<td>H11 (competition)</td>
<td>0.55</td>
<td>2.09</td>
<td>3.93</td>
<td>2.19</td>
</tr>
<tr>
<td>H11 (merge)</td>
<td>1.11</td>
<td>2.92</td>
<td>4.77</td>
<td>2.93</td>
</tr>
<tr>
<td>H12 (competition)</td>
<td>0.73</td>
<td>2.11</td>
<td>4.34</td>
<td>2.39</td>
</tr>
<tr>
<td>H12 (merge)</td>
<td>1.05</td>
<td>2.48</td>
<td>4.58</td>
<td>2.71</td>
</tr>
</tbody>
</table>

Figure 3: Effectiveness comparison of hybridizing different types of free variables using the competition strategy
4.6. The effectiveness of using consistent variables within local search

Based on the findings of local search (Section 4.3) and competition strategies (Section 4.5), this section examines the effect of collecting consistent variables within local search. First, a comparison is made between the one with $V_C$ only in CETS (H3) and the one with $V_C$ both in CETS and local search (H7). The results show that H7 performs better than H3 on average as shown in Figure 4 and Table 6. Secondly, comparisons are made among the heuristic variants using the competition strategy and collecting consistent variables within local search. Therefore, the feature of collecting consistent variables within local search is also added to H9, H10, H11, and H12 to become H13, H14, H15, and H16, respectively. The results are summarized in Figure 5 and Table 7. By comparing Table 7 with Table 5, we can find encouraging improvement after incorporating $V_C$ in the local search. For examples, the average gap over all instances is reduced from 2.28% (H9) to 1.57% (H13), and from 2.39% (H12) to 1.64% (H16).
Table 6: One-min. ave. gap (%) of each class of problems with $V_C$ in local search

<table>
<thead>
<tr>
<th>Heuristic variants</th>
<th>S- instances</th>
<th>M- instances</th>
<th>L- instances</th>
<th>All instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>H3 (no $V_C$ within local search)</td>
<td>0.70</td>
<td>2.02</td>
<td>3.92</td>
<td>2.22</td>
</tr>
<tr>
<td>H7 ($V_C$ within local search)</td>
<td>0.26</td>
<td>1.91</td>
<td>4.15</td>
<td>2.11</td>
</tr>
</tbody>
</table>

Figure 5: Effectiveness of hybridizing free variables and using $V_C$ within local search

Table 7: One-min. ave. gap (%) of each class of problems when hybridizing free variables and using $V_C$ within local search

<table>
<thead>
<tr>
<th>Heuristic variants</th>
<th>S- instances</th>
<th>M- instances</th>
<th>L- instances</th>
<th>All instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>H13 ($V_C$ &amp; $V_{SD}$ &amp; $V_{PA}$)</td>
<td>0.18</td>
<td>1.44</td>
<td>3.10</td>
<td>1.57</td>
</tr>
<tr>
<td>H14 ($V_C$ &amp; $V_{SD}$ &amp; $V_{PA}$)</td>
<td>0.15</td>
<td>1.47</td>
<td>2.97</td>
<td>1.53</td>
</tr>
<tr>
<td>H15 ($V_C$ &amp; $V_{SD}$ &amp; $V_{CA}$)</td>
<td>0.11</td>
<td>1.86</td>
<td>2.95</td>
<td>1.64</td>
</tr>
<tr>
<td>H16 ($V_C$ &amp; $V_{SD}$ &amp; $V_{CA}$)</td>
<td>0.16</td>
<td>1.71</td>
<td>3.05</td>
<td>1.64</td>
</tr>
</tbody>
</table>
4.7. The effectiveness of AMP heuristics

To demonstrate the strength of our AMP heuristic comparing to the literature, Table 8 compares several variants of our heuristics with CETS with local search, a very simple method in Li and Curry [38], and tight oscillation, a method with more advanced local search in Li [40]. It is obvious that these AMP variants perform much better than the very simple approach-CETS with local search. Moreover, the results indicate H14 and H15 both perform better than tight oscillation under different time limits (1, 5, and 10 minutes). Although our simpler variants like H3 and H7 do not dominate the average results of tight oscillation, they do show competitive advantage over tight oscillation when running longer. All of the results above illustrate the effectiveness of the AMP heuristic.

5. Conclusions and Future Study

This paper proposes an adaptive memory projection method for multidimensional knapsack problems with generalized upper bound constraints. From our experiments, we have the following conclusions:

1. This study examines several types of free variables including two types of strongly determined variables, consistent variables, conditional attractive variables and persistent attractive variables. The results show consistent variables perform best among these five alternatives.

2. Using local search is helpful to find better solution. In particular, if consistent variables are also collected in local search, the overall results will generally be improved.

3. When hybridizing different types of free variables, the competition strategy outperforms the merge strategy. In addition, the competition strategy improves the solo-performance in general.

4. Variants of the AMP heuristics can produce very good computational results. Improved results for the GUBMKP are found using the AMP method.
Table 8: Ave. gaps of AMP heuristics versus tight oscillation

<table>
<thead>
<tr>
<th>Method</th>
<th>S- instances</th>
<th>M- instances</th>
<th>L- instances</th>
<th>All instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>H3 ($V_C$)</td>
<td>0.70</td>
<td>2.02</td>
<td>3.92</td>
<td>2.22</td>
</tr>
<tr>
<td>H7 ($V_C$ within local search)</td>
<td>0.26</td>
<td>1.91</td>
<td>4.15</td>
<td>2.11</td>
</tr>
<tr>
<td>H14 ($V_C$ &amp; $V_{SDJ}$ &amp; $V_{PA}$)</td>
<td>0.15</td>
<td>1.47</td>
<td>2.97</td>
<td>1.53</td>
</tr>
<tr>
<td>H15 ($V_C$ &amp; $V_{SO1}$ &amp; $V_{CL}$)</td>
<td>0.11</td>
<td>1.86</td>
<td>2.95</td>
<td>1.64</td>
</tr>
<tr>
<td>CETS with local search</td>
<td>2.66</td>
<td>8.84</td>
<td>12.92</td>
<td>8.14</td>
</tr>
<tr>
<td>tight oscillation</td>
<td>0.46</td>
<td>2.39</td>
<td>3.27</td>
<td>2.04</td>
</tr>
</tbody>
</table>

*Run time limit: 1 min

<table>
<thead>
<tr>
<th>Method</th>
<th>S- instances</th>
<th>M- instances</th>
<th>L- instances</th>
<th>All instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>H3 ($V_C$)</td>
<td>0.34</td>
<td>1.57</td>
<td>2.86</td>
<td>1.59</td>
</tr>
<tr>
<td>H7 ($V_C$ within local search)</td>
<td>0.11</td>
<td>1.40</td>
<td>1.96</td>
<td>1.16</td>
</tr>
<tr>
<td>H14 ($V_C$ &amp; $V_{SDJ}$ &amp; $V_{PA}$)</td>
<td>0.08</td>
<td>1.27</td>
<td>1.07</td>
<td>0.81</td>
</tr>
<tr>
<td>H15 ($V_C$ &amp; $V_{SO1}$ &amp; $V_{CL}$)</td>
<td>0.08</td>
<td>1.25</td>
<td>1.10</td>
<td>0.81</td>
</tr>
<tr>
<td>CETS with local search</td>
<td>2.04</td>
<td>7.75</td>
<td>11.78</td>
<td>7.19</td>
</tr>
<tr>
<td>tight oscillation</td>
<td>0.38</td>
<td>2.00</td>
<td>2.77</td>
<td>1.72</td>
</tr>
</tbody>
</table>

*Run time limit: 5 mins

<table>
<thead>
<tr>
<th>Method</th>
<th>S- instances</th>
<th>M- instances</th>
<th>L- instances</th>
<th>All instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>H3 ($V_C$)</td>
<td>0.30</td>
<td>1.50</td>
<td>2.58</td>
<td>1.46</td>
</tr>
<tr>
<td>H7 ($V_C$ within local search)</td>
<td>0.11</td>
<td>1.28</td>
<td>1.29</td>
<td>0.89</td>
</tr>
<tr>
<td>H14 ($V_C$ &amp; $V_{SDJ}$ &amp; $V_{PA}$)</td>
<td>0.06</td>
<td>1.23</td>
<td>0.88</td>
<td>0.73</td>
</tr>
<tr>
<td>H15 ($V_C$ &amp; $V_{SO1}$ &amp; $V_{CL}$)</td>
<td>0.07</td>
<td>1.20</td>
<td>0.83</td>
<td>0.70</td>
</tr>
<tr>
<td>CETS with local search</td>
<td>2.01</td>
<td>7.13</td>
<td>11.35</td>
<td>6.83</td>
</tr>
<tr>
<td>tight oscillation</td>
<td>0.38</td>
<td>1.94</td>
<td>2.59</td>
<td>1.64</td>
</tr>
</tbody>
</table>

*Run time limit: 10 mins
This study focuses on choosing free variables. It is intriguing if the attention can be switched to fixed variables. That is, in the referent-optimization phase, we may consider to fix these variables and then solve the problem with the remaining variables in order to achieve the combinatorial implosion effect.

Appendix A. Impact of Pseudo Cuts

Glover [16] proposes two types of pseudo-cuts to diversify the solutions in the referent phase. This study hybridizes these two types of pseudo-cut in the reference phase. The parameter for the pseudo-cut is a constant ($d$). The use of the pseudo-cut can lead search into a new region. If the method for diversification is not under consideration, the heuristic may find a lot of duplicated solutions. The duplication rate of critical solutions is calculated by Eq. A.1.

This study records two critical solutions for each iteration, so the maximal number of critical solutions is equal to number of iterations multiplying by 2.

\[
\text{Duplication rate of critical sols} = \frac{\text{Max. no. of critical sols} - \text{no. of unique critical sols}}{\text{Max. no. of critical sols}} \quad (A.1)
\]

The impact of $d$ value of the pseudo-cut method is analyzed. Three values of $d$, 2, 3 and 4 are implemented. Figures A.1 and A.2 illustrate the impact of different strategies of pseudo-cut on solution quality. After re-launch, the heuristic visits different regions by different strategies. According to the results above, the pseudo-cut with $d = 4$ and without using pseudo-cut perform better than other settings.

Figures A.3 and A.4 show the impact of different strategies of pseudo-cut on duplication rate. The use of pseudo-cut is able to lead the heuristic to different regions. For instance, the duplication rate is rapidly increased when without using pseudo-cut after the second re-launch. Therefore, an ideal setting should keep the acceptable duplication rate within an acceptable range.
By considering both the solution quality and duplication, the setting of pseudo-cut with $d = 4$ is adopted since it attains a better solution quality with a lower duplication rate.

**References**


Figure A.2: The standard deviation of gaps using different pseudo-cut strategies

Figure A.3: The average duplication of different pseudo-cut strategies
Figure A.4: The standard deviation of duplication of different pseudo-cut strategies


[48] M. Fischetti, A. Lodi, Local branching, Mathematical Programming 98 (1)

[49] F. Glover, Heuristics for integer programming using surrogate constraints,
Decision Sciences 8 (1) (1977) 156–166.

[50] F. Glover, Multi-start and strategic oscillation methods–principles to ex-
plot adaptive memory, in: M. Laguna, J. L. G. Velarde (Eds.), Computing
Tools for Modeling, Optimization and Simulation: Interfaces in Computer
Science and Operations Research, Kluwer Academic Publishers, Boston,
USA, 2000, pp. 1–24.
• Consistent variables perform best among all five types of free variables.
• If consistent variables are also collected in local search, the overall results will generally be improved.
• When hybridizing different types of free variables, the competition strategy outperforms the merge strategy.
• Variants of the AMP method can produce very good computational results for the GUBM KP.