Al-Shihabi, Sameh

Article

A novel core-based optimization framework for binary integer programs- the multidemand multidimesional Knapsack problem as a test problem

Operations Research Perspectives

Provided in Cooperation with:
Elsevier

Suggested Citation: Al-Shihabi, Sameh (2021) : A novel core-based optimization framework for binary integer programs- the multidemand multidimesional Knapsack problem as a test problem, Operations Research Perspectives, ISSN 2214-7160, Elsevier, Amsterdam, Vol. 8, pp. 1-13, https://doi.org/10.1016/j.orp.2021.100182

This Version is available at:
https://hdl.handle.net/10419/246442

Operations Research Perspectives 8 (2021) 100182

Contents lists available at ScienceDirect

Operations Research Perspectives

journal homepage: www.elsevier.com/locate/orp

A Novel Core-Based Optimization Framework for Binary Integer Programs-the Multidemand Multidimesional Knapsack Problem as a Test Problem

Sameh Al-Shihabi [*,a,b]

[a] Industrial Engineering and Engineering Management Department, University of Sharjah, Sharjah, PO Box 27272, United Arab Emirates
[b] Industrial Engineering Department, University of Jordan, Amman, 11942, Jordan

ARTICLE INFO

Keywords:
Combinatorial optimization
Core problem
Multidemand multidimensional knapsack problem
Branch & Bound
Dual prices

ABSTRACT

The effectiveness and efficiency of optimization algorithms might deteriorate when solving large-scale binary integer programs (BIPs). Consequently, researchers have tried to fix the values of certain variables called adjunct variables, and only optimize a small problem version formed from the remaining variables called core variables, by relying on information obtained from the BIP's LP-relaxation solution. The resulting reduced problem is called a core problem (CP), and finding an optimal solution to a CP does not mean finding an optimal solution to the BIP unless the adjunct variables are fixed to their optimal values. Thus, in this work, we borrow several concepts from local search (LS) heuristics to move from a CP to a neighbouring CP to find a CP whose optimal solution is also optimal for the BIP. Thus, we call our framework CORE-LP-LS. We also propose a new mechanism to choose core variables based on reduced costs. To demonstrate and test the CORE-LP-LS framework, we solve a set of 126 multidemand multidimensional knapsack problem (MDMKP) instances. We solve the resulting CPs using two algorithms, namely, commercial branch and bound solver and the state-of-the-art meta-heuristic algorithm to solve MDMKP. As a by-product to our experiments, the CORE-LP-LS framework variants found 28 new best-known solutions and better average solutions for most of the solved instances.

1. Introduction

The purpose of the core concept to solve a binary integer program (BIP), as stated in [41], is æto reduce the original problem by only considering a core of items for which it is hard to decide if they belong to the optimal solution or not, whereas the variables for all items outside the core are fixed to certain values, i.e., 0 and 1 for BIP.ɡ Thus, based on the core concept, variables can belong to one of two sets: the core variables' set N^{core} and the adjunct variables' set N^{adj}, while the BIP variables' set is $N = N^{core} \cup N^{adj}$. The N^{adj} variables are fixed to either 0 or 1, while we create a core problem (CP) whose decision variables are those in N^{core} only and solve it instead of solving the BIP itself, e.g., [6]. Consequently, the BIP solution, x_{BIP}, can be represented as $x_{BIP} = (x_{adj}, x_{core})$ where x_{adj} and x_{core} represent the adjunct and core variable parts of the the solution, respectively. The CP is a reduction to the BIP; however, finding the CP's optimal solution, x^{opt}_{core}, does not mean finding the BIP's optimal solution, x^{opt}_{BIP}, unless we have $x^{opt}_{BIP} = (x^{opt}_{adj}, x^{opt}_{core})$, i.e., variables in N^{adj} are fixed to their optimal values in x^{opt}_{BIP}.

Fixing the N^{adj} variables to x^{opt}_{adj} is a pre-requisite condition to finding x^{opt}_{BIP} when using the core concept, so is it possible to find a CP that satisfy this condition? In other words, is it possible to find a CP whose optimal solution is also optimal for the BIP? In this paper, we propose a framework that tries to find such a CP which we call an optimal CP, as shown in Definition 1. The only guarantee that we find an optimal CP is to know x^{opt}_{BIP}. Consequently, we search for the best CP that results in the best objective function value, hoping that this CP is an optimal CP. In other words, we solve the BIP by searching for optimal CP.

Definition 1. An optimal CP is a CP whose adjunct variables are fixed to their values in the optimal solution of an BIP.

The proposed framework borrows several concepts from local search (LS) algorithms [23]. For example, we suggest a method to define neighbouring CPs similar to neighbouring solutions in LS. Moreover, moving from a CP to another one depends on finding a descent direction.

Like several previous researchers (e.g., [41], and [25]), we use the BIP's LP-relaxation solution, $LP(BIP)$, to identify the CP and to fix the N^{adj} variables. Similar to [41], we use a fixed core size; however, we try to identify related variables by considering the reactions of the

* Corresponding author.
E-mail address: salshihabi@sharjah.ac.ae.

https://doi.org/10.1016/j.orp.2021.100182
Received 2 December 2020; Received in revised form 11 February 2021; Accepted 15 March 2021
Available online 24 March 2021

variables' reduced costs. We increase the size of a CP by moving variables from N^{adj} to N^{core} if a variable in N^{adj} is related to another variable in N^{core}. We call this step probing, and it is found to improve the performance of the proposed framework.

We call the proposed framework CORE-LP-LS because we rely on LP-relaxation solutions to identify CPs, use an LS heuristic to move from a CP to a better, neighbouring CP, and the framework is based on the core concept. We select the multidemenand multidimensional knapsack problem (MDMKP) to test our algorithm; however, we can apply it to any BIP. Using a set of MDMKP instances, we try to find if we can find optimal BIP solutions by searching for optimal CPs, and how significant is the probing step.

MDMKP, also known as the multidimensional knapsack/covering problem (KCP) [27], is about selecting some items from set $N = \{1, 2, ..., n\}$ to maximize the sum of profits associated with the selected items where each item $j \in N$ has profit c_j. Thus, we use binary variable x_j that has a value of 1 if item $j \in N$ is selected, 0 otherwise. Equation 1 shows the objective function of MDMKP, while we define the decision variables in Equation 4.

MDMKP constraint set $R = \{1, 2, ..., m, m+1, m+2, ..., m+q\}$ can be divided into two sets. The knapsack constraint set, $R_{knap} = \{1, 2, ..., m\}$, is one of these two constraint sets where each constraint has capacity $b_i,\ \forall i \in R_{knap}$, and each item $j \in N$ consumes a_{ij} units of this capacity. The second constraint set is the weighted covering constraint set, $R_{cover} = \{m+1, m+2, ..., m+q\}$, where we need to cover demand $b_i,\ \forall i \in R_{cover}$, and each item $j \in N$ covers a_{ij} units of this demand. The knapsack and weighted covering constraint sets are shown in Equations 2 and 3, respectively. The full mathematical model of the MDMKP is shown in Equations 1–4.

$$Maximize\ z = \sum_{j=1}^{n} c_j x_j \tag{1}$$

s.t.

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i, \forall i \in R_{knap} \tag{2}$$

$$\sum_{j=1}^{n} a_{ij} x_j \geq b_i, \forall i \in R_{cover} \tag{3}$$

$$x_j \in \{0, 1\}, \forall j \in \{1, 2, ..., n\} \tag{4}$$

MDMKP assumes that the profit of item $j \in N$ can take any value, $c_j \in \mathbb{R},\ \forall j \in N$; however, capacity and demand limits can only take positive number $b_i \in \mathbb{R}^+,\ \forall i \in R$, while capacity consumption and demand contribution can be any non-negative value, $a_{ij} \geq 0,\ \forall i \in R$ *and* $\forall j \in N$.

MDMKP has several applications such as facility location ([11] and [42], and [40]), and capital budgeting and portfolio selection [7]. MDMKP subsumes the famous MKP [18] that only considers the knapsack constraints, i.e. having $q = 0$ [33], while KP is a special case of MKP where we have a single knapsack constraint. Both KP and MKP are NP-hard problems ([32] and [18]), respectively, and since MKP is a special case of MDMKP; consequently, MDMKP is also NP-hard problem. Researchers have used exact [33], heuristic [10], and meta-heuristic algorithms to solve MKP instances. Examples of meta-heuristic algorithms used to solve MKP include genetic algorithm (GA), tabu search (TS), and ant colony optimization (ACO), as in [14], [21], and [2], respectively. For more information about MKP solution methodologies and variants, readers are referred to [29], [18], and [30].

Similar to MKP and other NP-hard problems, researchers have used heuristics and meta-heuristics to solve large MDMKP instances. Researchers have focused on Tabu search(TS)([12], [5], and [31]) and scatter search (SS)([28] and [19]) to solve MDMKP. The alternating control tree (ACT) of [27] is another algorithm solution framework used to solve MDMKP that integrate heuristic and exact techniques.

We use MDMKP to check the effectiveness and efficiency of the CORE-LP-LS framework. On the one hand, the effectiveness of the CORE-LP-LS framework, i.e., its ability to find optimal CPs leading to optimal BIP solutions, is checked by solving the CPs using an exact algorithm, namely the B&B algorithm of *CPLEX12.10*. Thus, we call this hybrid CORE-LS-LP-IP. We use an exact solver in this experiment because we want to block the effects of the algorithm used to solve the CPs. On the other hand, the efficiency of the CORE-LP-LS framework, its ability to find high-quality solutions in short times, is checked by replacing the exact solver with the state-of-the-art algorithm to solve MDMKP. For this purpose, we use the two-stage tabu search (TSTS) algorithm of [31] to solve the CPs. Thus, henceforth, we denote this algorithm by CORE-LP-LS-TSTS. In the efficiency experiment, we do not benefit from the parallel nature of the CORE-LP-LS framework, which is expected to expedite the execution of the algorithm. Using the two CORE-LS-LP hybrids, CORE-LS-LP-IP and CORE-LS-LP-TSTS, we were able to identify 28 new best-known values out of 126 solved instances. Moreover, the average solution values found by the hybrids are better than the TSTS algorithm. Thus, finding new competitive algorithms to the state-of-the-art algorithm to solve MDMKP is another contribution of this work, although it is not the main objective of this paper.

This paper is organized as follow. In section 2, we review previous work related to the core concept, whereas in sections 3 and 4, we explain the CORE-LP-LS framework and introduce the probing concept, respectively. Experiments are discussed and presented in sections 5– 7. Section 5 checks the effectiveness of the CORE-LP-LS framework by experimenting with the CORE-LP-LS-IP algorithm. And in section 6, we test the CORE-LP-LS-TSTS algorithm to check the framework efficiency. We check the importance of the probing step in section 7. Lastly, we present our conclusions and possible future extensions to this work in section 8.

2. The Core concept

Reducing a BIP to a set of core variables depends on defining efficiency measures, $\tau_j,\ \forall j \in N$. Starting with KP, [6] relied on the benefit/cost ratio, $\tau_j = \frac{c_j}{a_j}$. Because M constraints are considered in MKP compared to one constraint in KP, [36] replaced $\tau_j = \frac{c_j}{a_j}$ by $\tau_j = \frac{c_j}{\sum_{i=1}^{m} a_{ij} \times r_i}$, where r_i is the surrogate multiplier of constraint, $i \in M$. [41] used the dual values of the LP-relaxation solution for the surrogate multipliers, which makes the efficiency measure dependant of variables' reduced costs. [26] have generalized the core concept to any IP, even if the problem inputs are negative. In addition to an efficiency measure that depends on reduced costs, [26] used another measure that considers the effect of a variable on the objective function and resources' consumption. Similar to previous researcher(e.g., [41], [25], [33], and [1]) CORE-LP-LS uses reduced costs to identify the core variables.

Both [6] and and [38] used a fixed core size to solve KP, whereas [34] used a core size of that depends on the problem size. [41] used a percentage of the problem size to solve MKP.

Several researchers highlighted the importance of problem characteristics on algorithms design and performance e.g.,[37], [38] and [35]. [25] have shown that it is better to have a flexible core size that depends on the problem difficulty. Because hard problems have low efficiency measure dispersion compared to easy problems, [25] have normalized the efficiency measure between $[-1, 1]$ and used cut-off values to define variables in N^{core} based on the normalized efficiency measures. CORE-LP-LS algorithm uses a fixed percentage of the problem size; however, the probing step, which takes into account problem characteristics, allows the expansion of the core set as explained later.

Researches have suggested several local search algorithms that solve reduced versions of BIPs where the size and elements of the reduced problem keeps changing. Kernel search (KS) [3] is an example of such algorithms where promising variables form a kernel, while other variables are divided into several buckets. A bucket variable might be added

to the kernel if it improves the solution value when solving a reduced problem from both the kernel and the bucket variables [4]. The KS variant of [20] allows variable removal from the kernel.

The relax-and-fix (RF) [45] and the fix-and-optimize (FO) [24] are two related algorithms where RF generates feasible solutions, whereas FO improves solutions. RF and FO have been widely used to solve lot sizing problems (e.g., [8], [16], [43], and [13]). Both algorithms use a moving window to change the reduced problem elements, and after each solution, a set of variables is fixed. In both algorithms, variables chosen by the window are forced to be binary.

[17] used a local branching constraint (LB-constraint) to define neighbouring solutions. Given feasible solution x to a BIP, we partition N into sets N^0 and N^1 that include variables having a value of 0 and 1, respectively, in x. A k-opt LB-constraint is defined in Equation 5. Both [17] and [15] used such neighbourhood definition to explore nodes in *B&B* and branch-and-cut algorithms. [44] also used LB-constraints in an algorithm that solves a series of reduced problems. Solutions to the core problem, which only includes variables having fractional values in the LP-relaxation solution, provides a lower bound (LB) for the MKP, while the LP-relaxation provides an upper bound (UB). LB-constraints are iteratively added to the problem to redefine the core problem and re-evaluate the UB and LB, until UB and LB converge. Researchers have improved [44] algorithm by considering only dominant solutions in [22] and improving the quality of UB by defining and solving mixed integer program programming relaxation instead of LP-relaxation in [46]. Unlike the LB-constraints, the neighbourhood definition in CORE-LP-LS forces one variable to change its value compared to the incumbent solution. Moreover, in the LB-based algorithms, added constraints are not removed as algorithm progress, which does not happen in CORE-LP-LS. Lastly, the solution evolution and problem reduction in the KS, RF, and FO are completely different that CORE-LP-LS.

$$\sum_{j\in N^1}(1-x_j)+\sum_{j\in N^0}x_j \le k \tag{5}$$

The CORe ALgorithm (CORAL) [33] starts with a small core that increases in size without using LB-constraints. Similar to CORE-LP-LS, CORAL checks if changing a variable value, compared to its value in the incumbent solution, would improve the incumbent solution. However, CORE-LP-LS uses the same core size, unless the size is temporarily increase at an iteration due to probing. Moreover, CORAL does not

Table 1
Computational results of the CORE-LP-LS-IP algorithm with probing for instances having $n = 100$ and $n = 250$.

Instance 100-	BKV	BFV-IP	Core Statistics	Instance 250-	BKV	BFV-IP	Core Statistics
5-2-0-0	28384	28384	40,40,40,1,36	5-2-0-0	78486	78486	40,40,40,0,1
5-2-0-1	26386	26386	40,40,40,0,1	5-2-0-1	75132	75132	40,40,40,0,1
5-2-0-2	23484	23484	40,40,40,0,1	5-2-0-2	71003	71003	40,52,44.5,1,15
5-2-0-3	27374	27374	40,40,40,0,1	5-2-0-3	80311	80311	40,40,40,0,1
5-2-0-4	30632	30632	40,40,40,0,1	5-2-0-4	70935	70935	40,47,44.0,1,10
5-2-0-5	44674	44674	40,52,46.2,1,23	5-2-0-5	130981	130981	40,48,44.3,2,38
5-2-1-0	10379	10369	40,46,42.5,1,38	5-2-1-0	26666	26666	40,40,40,0,1
5-2-1-1	11114	11114	40,40,40,0,1	5-2-1-1	26864	26864	40,48,45.4,1,17
5-2-1-2	10124	10124	40,40,40,0,1	5-2-1-2	27280	27280	40,55,47.2,1,61
5-2-1-3	10567	10567	40,40,40,0,1	5-2-1-3	26269	26269	40,40,40,0,1
5-2-1-4	10658	10658	40,40,40,0,1	5-2-1-4	27293	27293	40,40,40,0,1
5-2-1-5	17550	17550	40,42,40.1,1,13	5-2-1-5	44419	44410	40,53,43.2,1,132
5-5-0-0	21892	21892	40,40,40,0,1	5-5-0-0	68026	68026	40,50,50,0,1
5-5-0-1	26280	26280	40,40,40,0,1	5-5-0-1	60795	60795	40,56,50,1,35
5-5-0-2	20628	20628	40,40,40,0,1	5-5-0-2	62093	62093	40,49,44.5,2,85
5-5-0-3	21547	21547	40,40,40,0,1	5-5-0-3	66567	66567	40,40,40,0,1
5-5-0-4	25074	25074	40,40,40,0,1	5-5-0-4	61929	61929	40,40,40,0,1
5-5-0-5	40327	40327	40,40,40,0,1	5-5-0-5	127934	127934	40,40,40,0,1
5-5-1-0	10263	10263	40,40,40,0,1	5-5-1-1	26665	26665	40,40,40,0,1
5-5-1-1	10625	10625	40,40,40,0,1	5-5-1-2	26648	26648	40,53,48.3,1,26
5-5-1-2	10198	10198	40,40,40,0,1	5-5-1-2	26648	26648	40,53,48.3,1,26
5-5-1-3	10030	10030	40,40,40,0,1	5-5-1-3	25923	25923	40,54,44.8,1,6
5-5-1-4	9964	9964	40,46,43.3,1,6	5-5-1-4	26064	**26126**	40,52,46.3,1,23
5-5-1-5	15603	15603	40,45,42.4,1,13	5-5-1-5	41372	41372	40,45,42.0,3,82
10-5-0-0	21852	21852	40,50,45.3,1,10	10-5-0-0	56306	**56354**	40,56,47.4,2,35
10-5-0-1	20645	20645	40,43,41.6,1,6	10-5-0-1	59619	59619	40,57,51.3,3,73
10-5-0-2	19517	19517	40,40,40,0,1	10-5-0-2	54912	**54954**	40,42,41.4,1,8
10-5-0-3	20596	20596	40,54,46.4,4,27	10-5-0-3	52399	52399	40,40,40,0,1
10-5-0-4	19423	19264	40,59,52.8,3,45	10-5-0-4	58234	58234	40,40,40,0,1
10-5-0-5	35933	35933	40,43,40.6,3,24	10-5-0-5	99682	**99689**	40,41,40.2,4,37
10-5-1-0	10018	10018	40,54,48.4,2,16	10-5-1-0	26976	26976	40,56,48.6,2,28
10-5-1-1	9839	9839	40,40,40.0,0,1	10-5-1-1	26658	**26660**	40,52,45.6,1,5
10-5-1-2	10000	10000	40,40,40.0,0,1	10-5-1-2	25749	**25825**	40,57,48.2,2,23
10-5-1-3	10544	10544	40,59,52.7,1,21	10-5-1-3	27181	27172	40,57,49.7,4,71
10-5-1-4	10011	10011	40,40,40.0,0,1	10-5-1-4	26856	26816	40,57,48.7,3,45
10-5-1-5	16230	16230	40,44,41.3,1,18	10-5-1-5	46244	46244	40,49,44.6,6,84
10-10-0-0	22054	22054	40,61,54.8,1,24	10-10-0-0	52441	**52442**	40,55,49.2,3,18
10-10-0-1	20103	20103	40,40,40.0,0,1	10-10-0-1	53745	53745	40,62,58.5,3,35
10-10-0-2	19381	19381	40,49,44.3,4,28	10-10-0-2	46927	46927	40,40,40,0,1
10-10-0-3	17434	17434	40,40,40.0,0,1	10-10-0-3	54856	54856	40,55,50.5,2,31
10-10-0-4	18833	18833	40,44,41.3,2,8	10-10-0-4	49675	**49683**	40,47,42.6,1,5
10-10-0-5	33837	33837	40,48,44.6,2,8	10-10-0-5	92989	**92991**	40,50,43.2,5,141
10-10-1-0	8560	8560	40,45,42.1,1,14	10-10-1-0	26696	26696	40,42,41.4,1,9
10-10-1-1	8493	8493	40,47,42.9,4,25	10-10-1-1	25893	25893	40,54,45.9,2,17
10-10-1-2	9266	9266	40,40,40.0,0,1	10-10-1-2	26517	26517	45,60,55.4,6,34
10-10-1-3	9823	9823	40,40,40.0,0,1	10-10-1-3	26684	26684	40,65,59.1,4,30
10-10-1-4	8929	8929	40,40,40.0,0,1	10-10-1-4	26676	26676	40,62,56.4,5,71
10-10-1-5	14152	14152	40,45,42.2,1,40	250-10-10-1-5	42629	42629	40,59,56.2,3,18

remove any core variable, whereas CORE-LP-LS allows that.

3. CORE-LP-LS Framework

We start this section by describing how to create a CP of a BIP, followed by defining neighbouring cores. We then show how we use a LS algorithm to move from one core to a better one in pursuit for the optimal core, while implicitly, we optimize the BIP.

3.1. Core problem creation

Algorithm 1 shows a pseudo-code of the steps needed to create a CP of a BIP of size r. The inputs to the algorithm are both BIP and r, while the output is set N^{core} and partial solution x_{adj}. The algorithm starts by solving LP(BIP), from which we get the variables' reduced costs $\Pi = \{\pi_1, \pi_2, ..., \pi_n\}$, where π_j is the reduced cost of variable $j \in N$. We then rank variables in an ascending order based on their absolute reduced costs, as shown in line 3. The ranked variables are saved to vector RV, from which we choose the first r variables to form set N^{core}. Consequently, we include the remaining $n - r$ variables into set N^{adj}, as shown in lines 4–5. Note that the minimum value of r should be equal to the number of variables whose reduced costs is 0. In our implementation to solve MDMKP instances, we compare r to the number of variables having 0 reduced cost and choose the larger number.

Since N^{adj} has variables whose reduced costs are not equal to 0, and these variables are bounded in LP(BIP). We fix variables in N^{adj} such that variables having negative reduced cost are fixed to 0; otherwise, they are fixed to 1. Based on these fixed values, we can further partition N^{adj} into sets N_0^{adj} and N_1^{adj} that include variables fixed to 0 and 1, respectively. Henceforth, we use $CP(BIP, r)$ to denote the resulting CP, which can be represented using Equations 6 –10. The solution of the BIP problem is also a function of BIP and r, $x(CP(BIP, r)) = (x_{adj}(CP(BIP, r)), x_{core}^{incumb}(CP(BIP, r)))$, where $x_{core}^{incumb}(CP(BIP, r))$ is the incumbent solution of the core problem.

$$Maximize\ z^{core} = \sum_{j \in N^{core}} c_j x_j \tag{6}$$

Table 2
Comparison between CORE-LP-LS-TSTS and TSTS in solving MDMKP instances having $n = 100$.

instance	CORE-LP-LS-TSTS					TSTS	
	Best	Avg	Changes Number	CPs Number	time (s)	Best	Avg
100-5-2-0-0	28384	28384.0	0	15	18.2	28384	28384.0
100-5-2-0-1	26386	26386.0	0	15	14.2	26386	26386.0
100-5-2-0-2	23484	23484.0	0	15	17.5	23484	23484.0
100-5-2-0-3	27374	27374.0	0	16	16.7	27374	27374.0
100-5-2-0-4	30362	30362.0	0	16	24.9	30632	30362.0
100-5-2-0-5	44674	44652.3	0	22	32.0	44674	44674.0
100-5-2-1-0	10364	10364.0	0	13	26.2	10364	10364.0
100-5-2-1-1	11114	11114.0	0	14	14.1	11114	11114.0
100-5-2-1-2	10124	10124.0	0	13	19.4	10124	10112.4
100-5-2-1-3	10567	10567.0	0	14	12.3	10567	10567.0
100-5-2-1-4	**10658**	10658.0	0	14	13.9	10547	10547.0
100-5-2-1-5	17545	17545.0	0	23	25.8	17545	17539.6
100-5-5-0-0	21892	21892.0	0	14	15.2	21892	21892.0
100-5-5-0-1	26280	26280.0	0	15	13.3	26280	26280.0
100-5-5-0-2	20628	20628.0	0	15	17.6	20628	20628.0
100-5-5-0-3	21547	21547.0	0	15	13.7	21547	21547.0
100-5-5-0-4	25074	25074.0	0	14	14.7	25074	25074.0
100-5-5-0-5	40327	40327.0	0	23	25.2	40327	40327.0
100-5-5-1-0	10263	10263.0	0	13	22.8	10263	10263.0
100-5-5-1-1	10625	10625.0	0	14	14.1	10625	10625.0
100-5-5-1-2	10198	10198.0	0	13	29.7	10198	10139.6
100-5-5-1-3	10030	10030.0	0	14	31.8	10030	10030.0
100-5-5-1-4	9964	9964.0	0	13	59.5	9964	9964.0
100-5-5-1-5	15603	15600.2	0	24	36.2	15603	15603.0
100-10-5-0-0	21852	21845.9	1	16	24.5	21852	21852.0
100-10-5-0-1	20645	20610.7	0.3	14.6	54.8	20645	20603.4
100-10-5-0-2	19157	19157.0	0	13	33.01	19517	19157.0
100-10-5-0-3	20556	20526.1	1.9	16.8	37.4	**20596**	20506.8
100-10-5-0-4	**19283**	19269.2	0.3	14.6	31	19278	19252.2
100-10-5-0-5	35816	35790.6	1.5	26.7	26.7	**35933**	35857.0
100-10-5-1-0	10018	9991.8	0.7	13.8	29.4	10018	10018.0
100-10-5-1-1	9839	9839.0	0	13	26.9	9839	9839.0
100-10-5-1-2	10000	10000.0	0	13	22.6	10000	10000.0
100-10-5-1-3	10510	10510.0	0	13	36.9	**10544**	10512.3
100-10-5-1-4	10011	10011.0	0	13	40.4	10011	9983.5
100-10-5-1-5	16230	16209.8	0.9	25.6	35.6	16230	16230.0
100-10-10-0-0	22054	22034.1	0.2	9.2	23	22054	22054.0
100-10-10-0-1	20103	20103.0	0	8	17.7	20103	20103.0
100-10-10-0-2	19132	19132.0	1.2	9.2	23.7	**19381**	19367.2
100-10-10-0-3	17434	17434.0	0	8	43.3	17434	17434.0
100-10-10-0-4	**18833**	18789.2	2.6	12	33.4	18792	18789.7
100-10-10-0-5	33837	33823.4	1.2	16.2	37.3	33837	33836.6
100-10-10-1-0	8542	8542.0	0	12	44	**8560**	8524.5
100-10-10-1-1	8439	8416.2	1.3	14.3	38.3	**8493**	8439.0
100-10-10-1-2	9266	9266.0	0	13	45.7	9266	9266.0
100-10-10-1-3	9823	9823.0	0	12	27.5	9823	9811.4
100-10-10-1-4	8929	8929.0	0	12	30.9	8929	8929.0
100-10-10-1-5	14151	14151.0	0	25	58.6	**14152**	14152.0

Table 3
Comparison between CORE-LP-LS-TSTS and TSTS in solving MDMKP instances having $n = 250$.

instance	CORE-LP-LS-TSTS					TSTS	
	Best	Avg	Changes Number	CPs Number	time (s)	Best	Avg
250-5-2-0-0	78486	78486.0	0	23	52.8	78486	78400.0
250-5-2-0-1	75132	75132.0	0	23	61.2	75132	74956.2
250-5-2-0-2	70987	70987.0	0	23	79.8	70987	70888.9
250-5-2-0-3	80311	80311.0	0	23	90.6	80311	80219.8
250-5-2-0-4	70935	70919.2	0.1	23.6	91.2	70935	70836.4
250-5-2-0-5	**130910**	130910.0	0	32	79.9	130468	129551.5
250-5-2-1-0	**26666**	26666.0	0	21	58.2	26659	26573.8
250-5-2-1-1	26864	26816.5	0.3	23.2	76.9	26864	26815.2
250-5-2-1-2	27274	27274.0	0	21	108	**27280**	27237.6
250-5-2-1-3	**26269**	26253.8	0.2	22.9	60.8	26250	26184.1
250-5-2-1-4	**27293**	27273.2	1.3	27	136.5	27269	27225.4
250-5-2-1-5	**44410**	44397.3	0.3	40.8	232.2	44402	44341.8
250-5-5-0-0	68026	68026.0	2.2	28.8	97.8	68026	68024.3
250-5-5-0-1	**60795**	60770.7	0.8	29.8	92.1	60757	60714.9
250-5-5-0-2	62088	62083.1	0.3	23.3	56.3	**62093**	62073.0
250-5-5-0-3	66657	66657.0	0	22	85.6	66567	66525.5
250-5-5-0-4	61929	61929.0	0	22	45.5	61929	61913.7
250-5-5-0-5	**127934**	127934.0	0	33	101.2	127885	127785.2
250-5-5-1-0	26966	26966.0	0	20	47.5	26966	26917.0
250-5-5-1-1	26665	26665.0	0	19	50.2	26665	26557.5
250-5-5-1-2	26591	26597.8	0.6	23.9	98.2	**26620**	26562.4
250-5-5-1-3	25923	25892.8	0.2	20.7	61.1	25923	25779.5
250-5-5-1-4	**26058**	26038.2	0.1	20.7	84.5	26036	25976.6
250-5-5-1-5	**41372**	41336.1	2.1	45.1	94.9	41350	41237.2
250-10-5-0-0	**56327**	56282.6	2.2	28.3	108.6	56315	56029.8
250-10-5-0-1	59604	59583.8	1.7	30.7	160.9	**59605**	59568.8
250-10-5-0-2	**55034***	54936.5	1.7	32.8	105.4	54814	54727.0
250-10-5-0-3	**52416***	52389.8	1.3	28.7	155.1	52256	52188.3
250-10-5-0-4	**58234**	58234.1	1.5	23	104.4	58058	57906.9
250-10-5-0-5	99752*	99716.7	2.4	45.9	206.2	**99763***	99419.8
250-10-5-1-0	**26970**	26945.5	2.5	23.5	101.5	26941	26874.6
250-10-5-1-1	**26658**	26611.4	2.7	32.4	106.9	26575	26535.7
250-10-5-1-2	**25749**	25725.5	0.2	19.8	67.9	25667	25597.3
250-10-5-1-3	27153	27139.2	1.7	28.4	152.9	**27159**	27129.5
250-10-5-1-4	26815	26815.0	0	19	76.3	26815	26772.6
250-10-5-1-5	**46244**	46214.4	1.4	39.6	129.2	46195	46161.8
250-10-10-0-0	52441	52400.2	2.6	37	181.2	52441	52407.8
250-10-10-0-1	53745	53703.1	2.5	32	178.3	53745	53686.2
250-10-10-0-2	**46927**	46927.0	0	20	113	46839	46801.3
250-10-10-0-3	**54856**	54809.8	0.4	23.6	134.6	54816	54777.4
250-10-10-0-4	49699*	49668.7	3.4	34.1	207	49699*	49618.9
250-10-10-0-5	**93006***	92972.2	1.5	39.8	128.2	92904	92786.4
250-10-10-1-0	26696	26696.0	1	27.4	80.5	26696	26644.3
250-10-10-1-1	**25876**	25804.2	1.8	23.6	120.6	25818	25794.5
250-10-10-1-2	26517	26494.5	2.8	27.4	183.4	26517	26478.3
250-10-10-1-3	26684	26665.9	2.9	29	199.1	26684	26624.7
250-10-10-1-4	**26676**	26625.5	3.5	27	160.7	26631	26617.1
250-10-10-1-5	42629	42534.7	3	47	249.5	42629	42531.0

s.t.

$$\sum_{j \in N^{core}} a_{ij} x_j \leq b_i^{core}, \forall i \in \{1, 2, ..., m\} \tag{7}$$

$$\sum_{j \in N^{core}} a_{ij} x_j \geq b_i^{core}, \ \forall i \in \{m+1, m+2, ..., m+q\} \tag{8}$$

$$x_j \in \{0, 1\}, \ \forall j \in N^{core} \tag{9}$$

$$b_i^{core} = b_i - \sum_{j \in N_1^{adj}} a_{ij}, \ \forall i \in R \tag{10}$$

3.2. Neighbouring cores

In a typical LS algorithm, we define the neighbourhood of solution x, $\mathcal{N}(x)$, as shown in Equation 11, where solution $y \in X$ is a neighbouring solution to x based on $\mathcal{N}(x)$. Generally, neighbourhood $\mathcal{N}(x)$ is defined relative to a given metric (or quasi-metric) function $\delta(x, y)$ introduced in the solution space S, and a limit to this metric α. For example, the incumbent solution of a BIP, x_{BIP}^{incumb}, is a binary vector of size n. To find a neighbouring solution, $y \in \mathcal{N}(x_{BIP}^{incumb})$, we might define $\delta(y, x_{BIP}^{incumb})$ in terms of the hamming distance between solutions x_{BIP}^{incumb} and y. If we use $\alpha = 1$, then $y \in \mathcal{N}(x_{BIP}^{incumb})$ is a neighbouring solution that has the same values of variables in x_{BIP}^{incumb}, except only for one variable. For example, if $n = 3, \alpha = 1$ and $x_{BIP}^{incumb} = \{1, 1, 0\}$, then $\mathcal{N}(x_{BIP}^{incumb}) = \{\{0,1,0\}, \{1,0,0\}, \{1,1,1\}\}$. If, however, $\alpha = 2$, then $\mathcal{N}(x_{BIP}^{incumb}) = \{\{0,0,0\}, \{0,1,1\}, \{1,0,1\}\}$. We call the variables that has different values in solutions y and x catalyst variables, and α is the number of catalyst variables.

$$\mathcal{N}(x) = \{y \in X | \delta(x, y) \leq \alpha\} \tag{11}$$

To map the concept of neighbouring solutions to neighbouring CPs, we modify the BIP itself by fixing the catalyst variables to their values in y. We denote the resulting *BIP* by *BIPmod* and use Algorithm 1 to find the CP associated with *BIPmod*. Solution $x(CP(BIP, r)$ and any of its

Table 4
Comparison between CORE-LP-LS-TSTS and TSTS in solving MDMKP instances having $n = 500$.

instance 500-30-30	BKV	CORE-LP-LS-TSTS Best,Avg	TSTS Best,Avg
-0-2-1	85188	**85311***(85311.0)	85194(85115.8)
-0-2-2	82073	**82334***(82334.0)	82071(81984.1)
-0-2-3	77393	**77450***(77450.0)	77141(77071.3)
-0-2-4	82304	**82347***(82347.0)	82217(82132.2)
-0-2-5	83525	**83648***(83549.4)	83500(83339.0)
-0-2-6	145967	**146154***(146018.2)	146004(145665.0)
-0-2-7	152246	**152320***(152271.2)	152015(151966.0)
-0-2-8	157687	**157788***(157523.5)	175552(157426.8)
-0-2-9	153751	**154078***(153982.3)	153766(153605.4)
-0-2-10	142173	**142237***(142237.0)	142084(141931.6)
-0-2-11	185226	**185250***(185032.0)	185056.0(184196.9)
-0-2-12	194614	194585(194492.3)	**194509**(194371.6)
-0-2-13	20246	**208425***(208363.3)	208292(208206.0)
-0-2-14	215849	**215944***(215877.2)	215844(215694.2)
-0-2-15	194224	**194310***(194222.3)	194193(194126.4)
-1-5-1	51666	51601(51532.4)	51561(51516.0)
-1-5-2	50101	**50172***(50013.2)	50009(49901.4)
-1-5-3	51226	51153(50824.3)	51140(50992.3)
-1-5-4	51637	**51681***(51662.3)	51602(51496.0)
-1-5-5	52078	**52224***(51983.2)	51963(51896.0)
-1-5-6	84052	**84072***(83983.5)	83947(83875.2)
-1-5-7	82850	**82870***(82585.3)	82859(82555.4)
-1-5-8	82722	**82763***(82603.8)	82735(82598.4)
-1-5-9	82825	**82831***(82562.4)	82560(82459.4)
-1-5-10	82845	**82958***(82876.3)	82580(82534.0)
-1-5-11	88887	88762(88762.0)	88831(88794.8)
-1-5-12	87254	**87287***(87199.0)	87280(87170.4)
-1-5-13	87315	87183(87139.2)	87228(87187.6)
-1-5-14	87583	87546(87493.5)	**87568**(87527.4)
-1-5-15	87956	87849(87789.3)	**87930**(87852.8)

neighbouring solutions $y(CP(BIPmod, r)$ should at least have α variables with different values. Unlike classical neighbourhoods, the differences between the two solutions could exceed α since we are creating a sub-region in which we solve a reduced optimization problem.

3.2.1. Minimizing neighbourhood size

To improve the efficiency of the CORE-LP-LS algorithm, we need to minimize the number of created CPs. For example, if we use a swapping operator, having two catalyst variables, for a problem of size n, then we will have $n \times n - 1$ CPs. Thus, it is essential not to create a large number of CPs; still, we need to increase the probability of finding x_{BIP}^{opt}. We, therefore, use a single flip operator to define the catalyst variables.

We use Algorithm 2 to explain how we define neighbouring CPs to solution x_{BIP}^{incumb}. We first partition x_{BIP}^{incumb} into two comprehensive and mutually exclusive sets: $I^1 = \{x_q : x_q = 1 \ in \ x_{BIP}^{incumb}\}$ and $I^0 = \{x_q : x_q = 0 \ in \ x_{BIP}^{incumb}\}$. Sets I^1 and I^0 include variables that have values of 1 and 0 in x_{BIP}^{incumb}, respectively. We then choose the set that has the least number of variables from I^1 and I^0, as shown in lines 2–3 of Algorithm 2. If set $|I^1| \leq |I^0|$ then we create new cores by flipping the values of variables in I^1 from 1 to 0. On the other hand, we flip the values of the variables in I^0 from 0 to 1 if $|I^1| \geq |I^0|$. Intuitively, choosing the set that has the least number of elements would minimize the number of created CPs that we need to solve.

After choosing the index of the set, for which we are going to apply the flip operator, the main loop for creating the set of CPs, *CPset*, begins. We, therefore, initialize *CPset* and the counter *iter* to $\varnothing$ and 1, respectively, as shown in lines 4 and 5 of algorithm 2. The number of CPs that we can create is equal to the number of variables in I^{index}, as shown in the loop parameters of line 7 in Algorithm 2. Note that for each CP, there is

Data: BIP and r
Result: N^{core} and x_{adj}
1 **begin**
2 $\Pi \xleftarrow{\pi_j, \forall j \in N}$ solve LP(BIP) ;
3 RV←Rank variables based on $|\pi_j|$ in an ascending order ;
4 $N^{core} \leftarrow$ choose the first r variables in RV ;
5 $N^{adj} \leftarrow N / N^{core}$;
6 **for** $j \in N^{adj}$ **do**
7 **if** $\pi_j < 0$ **then**
8 $x_{adj} = \{x_1, x_2, ..., x_j = 0 ..., x_{d=n-r}\}$;
9 **else**
10 $x_{adj} = \{x_1, x_2, ..., x_j = 1, ..., x_{d=n-r}\}$
11 End ;
12 End ;

Algorithm 1. Creating a core problem

Data: x_{BIP}^{incumb}, r
Result: $CPset = \{CP_1, CP_2, ..., CP_k\}$
1 **begin**
2 Partition x_{BIP}^{incumb} into I^0 and I^1 ;
3 index $\leftarrow \underset{0,1}{\text{argmin}}(|I^0|, |I^1|)$;
4 $CPset \leftarrow \emptyset$;
5 $iter = 1$;
6 **while** $iter \leq |I^{index}|$ **do**
7 $x_{iter} = \{x_{iter} | x_{iter} \in I^{index}\}$; x_{iter} at position $iter$ in I^{index};
8 $BIPmod \leftarrow$ fix x_{iter} value to $1 - index$ in BIP;
9 $CP_{iter}(BIPmod, r) \leftarrow$ define using Algorithm 1 ;
10 $CP_{iter}(BIPmod, r) \leftarrow$ enlarge using Algorithm 4 ;
11 $CPset = CPset \cup CP_{iter}$;
12 $iter = iter + 1$;
13 End ;
14 End ;

Algorithm 2. Creating set *CPset*

an associated x_{adj} that we save as well.

The main procedure to create *CPset* is described in lines 7–15 of Algorithm 2. For the variable in position *iter* of I^{index}, we first create *BIPmod*, in which we fix the chosen variable to its complementary value $1 -$ *index*. Again, we use Algorithm 1 to find $CP(BIPmod, r)$ and $x_{adj}(CP(BIPmod, r))$. We expand N^{core} to include more variables using the probing algorithm that we explain in the next section. The new CP is added to set *CPset*. Algorithm 2 is repeated each time a new x_{BIP}^{incumb} is found.

3.3. Searching for the best CP

The CORE-LP-LS algorithm is simply an LS algorithm where instead of searching neighbouring solution points to solution x^{incumb}, we search neighbouring CPs that are associated with the neighbouring solutions. We use Algorithm 3 to show how we implement a first-move LS strategy with any neighbourhood and CP definition; however, any other move strategy, such as best move, can be implemented. Algorithm 3 starts by finding an initial incumbent solution for the BIP, x_{BIP}^{incumb}, as shown in lines 2–5 of Algorithm 3.

We then generate the neighbouring CPs and save them to *CPset* using Algorithm 2. We also rank the CPs in *CPset* based on a user criterion. For example, we can rank CPs based on their sizes or based on the reduced cost of the catalyst variable the defined the CP. We, then start the search loop, as shown in lines 8–18. We find the BIP solution associated with the i^{th} CP, as shown in line 9. If this solution is better than z_{BIP}^{incumb}, then we update the incumbent value and solution, create a new *CPset* and rank it, and re-initialize the CP counter to 1, as shown in lines 10–16. This procedure is continued until a termination criterion is met. Possible termination criteria might include the number of investigated neighbouring solutions and time limits, to name a few.

4. Probing

A significant input that affects the success of the CORE-LP-LS algorithm, or any other core-based algorithm, is the choice of r. As a rule of thumb to use the CORE-LP-LS framework, r should guarantee that all variables that have 0 reduced costs are included in N^{core}. Moreover, the larger the size of r, the higher the probability of having better solutions; however, more time is needed to solve the CPs. Therefore, finding a good balance between time and solution quality is crucial.

As discussed in the previous section, we are not inspecting swapping neighbourhood, *swap(p,q)*, where we swap the values of variables x_q and x_p, we only flip the value of one variable. However, it is expected that variables are associated with each other such that we cannot obtain a better solution than x_{BIP}^{incumb} unless we simultaneously change two or more variable values. Consequently, if two variables need to change their values, compared to x_{BIP}^{incumb}, to find a better solution where one of them is included in N^{core} whereas the other is in N^{adj}, then it is impossible to move to the better solution if the variable in N^{adj} is fixed to the wrong value. So, the probing step tries to identify such a variable in N^{adj} and move it to N^{core}.

The probing step is an LP-based heuristic that compares reduced costs' changes that result from solving $LP(BIP^{incumb})$, and $LP(BIPmod)$, where BIP^{incumb} is the BIP that lead to identifying x_{BIP}^{incumb}. So x_{BIP}^{incumb} is the *BIP* itself if z^{incumb} is not updated ; otherwise, it is the *BIPmod* that updated z^{incumb}, as shown in Algorithm 3. We use the reduced costs' changes as a measure to assess which variables are to be moved from N^{adj} to N^{core}, i.e. increasing the number of core variables.

Algorithm 4 shows the details of the probing step and how we move variables from N^{adj} to N^{core}. An important input to Algorithm 4 is the reduced cost values that we obtain from LP(BIPmod). These reduced cost values are denoted by $\Pi_{BIPmod} = \{\pi_0^{BIPmod}, \pi_1^{BIPmod}, ..., \pi_n^{BIPmod}\}$. $\Pi_{ref} = \{\pi_0^{ref},$

Data: BIP and r
Result: x_{BIP}^{incumb}

begin
  $CP(BIP, r)$ and $x_{adj}(BIP, r) \leftarrow$ apply Algorithm 1 to BIP ;
  $x_{core}^{best} \leftarrow$ optimize the resulting $CP(BIP, r)$;
  $x_{BIP}^{incumb} = (x_{core}^{incumb}(CP(BIP, r)), x_{adj}(CP(BIP, r)))$;
  $z_{BIP}^{incumb} = f(x_{BIP}^{incumb}(BIP, r))$;
  $CPset \leftarrow$ using Algorithm 2 ;
  rank the CPs in $CPset$;
  $i = 1$ **while** *Termination criteria not met* **do**
    $z_{BIPmod}^{incumb} \leftarrow$ find BIP solution associated with $CP_i \in CPset$;
    if $z_{BIPmod}^{incumb} > z_{BIP}^{incumb}$ **then**
      $z^{incumb} = z_{BIPmod}^{incumb}$;
      $x^{incumb} = (x_{core}^{incumb}(CP_i(BIPmod, r)), x_{adj}(CP_i(BIPmod, r)))$;
      $CPset \leftarrow$ using Algorithm 2 ;
      rank the CPs in $CPset$;
      $i = 1$;
      continue
    ;
    $i++$;
    End ;
  End ;

Algorithm 3. LS heuristic implementing first move strategy to identify the optimal core

Data: *index*, *BIPmod*, N^{core}, N^{adj}, r, Q, and Π_{ref}
Result: N^{core} and x^{adj}
1 **begin**
2 Calculate $\delta\pi_j, \forall j \neq catalyst$ variable and $x_j^{incumb} = index$, based on Equation 12 ;
3 $N^Q \leftarrow$ the Q variables that have the highest $|\delta\pi_j|$ values ;
4 i=0 ;
5 **while** $i < |Q|$ **do**
6 *BIPmod2* $\leftarrow$ Modify *BIPmod* by fixing the i^{th} variable in N^Q to $index - 1$;
7 $N^{core2} \leftarrow$ apply Algorithm 1 to *BIPmod2* and r;
8 $N^{core} = N^{core} \cup N^{core2}$;
9 i=i+1 ;
10 End ;
11 $N^{adj} = N/N^{core}$ End ;

Algorithm 4. Probing algorithm

$\pi_1^{ref}, ..., \pi_n^{ref}\}$, on the other hand, is the reduced cost values of the variable that were obtained when solving BIP^{incumb}. Using Π_{BIPmod} and Π_{ref}, we calculate the changes in reduced cost for each variable $j \in N$, as shown in Equation 12, and save these changes to $\Delta\Pi = \{\delta\pi_1, \delta\pi_2, ..., \delta\pi_n\}$. Equation 12 has several cases. Two cases are used when the catalyst variable is in I^1, namely cases 1 and 3, while cases 2 and 4 are used when the catalyst variable is in I^0. We also distinguish the cases based on the value of π_j^{ref}. If $j \in N$ does not belong to the previous four cases, then it is assigned a large negative value, as shown in case 5 where M is a large number.

To explain the intuition behind Equation 12 and the probing heuristic, we consider the case of $index = 1, x_j^{incumb} = 1$ and $\pi_j^{ref} = 0$, then $\delta\pi_j$ is simply $-\pi_j^{BIPmod}$ as shown in case 1 of Equation 12. Assume that variables c and v satisfy this condition, and after fixing the catalyst variable to 0 and solving $LP(BIPmod)$, we have $\pi_c^{BIPmod} = -10$ and $\pi_v^{BIPmod} = 10$. We can interpret these two values as follow: variable c tries to change its value to 0 similar to the catalyst variable, while variable v keeps it value to 1 and have a stronger evidence to maintain its value of 1. Because of the reaction of variable c, we pick variable c to create *BIPmod2*, in which both variable c and the catalyst variable are fixed to 0. Based on the $LP(BIPmod2)$ solution, we create a new core based on $LP(BIPmod2)$, which we label by N^{core2}. Any variable in N^{core2} is moved to N^{core} if it is not already included in N^{core}.

We only consider Q variables that 1) have the highest $\delta\pi_j$ changes and 2) have $x_j^{incumb} = index$, as shown in lines 2 and 3 of Algorithm 4. We then create Q new BIPmod2 problems, in which we flip the values of two variables. One variable is the catalyst variable that defines CP, and the other is one of the Q variables, as shown in line 6 of Algorithm 4. We find N^{core} of each BIPmod2. Adjunct variables associated with $CP(BIPmod, r)$ that becomes core variable in $CP(BIPmod2, r)$ are added to the core variables of $CP(BIPmod)$, as shown in line 8 of Algorithm 4. The new N^{adj} set excludes any variable that becomes a core one.

$$\delta\pi_j = \begin{cases} -\pi_j^{BIPmod} & , \pi_j^{ref} = 0;\ x_j^{incumb} = 1;\ index = 1;\ j \neq catalyst \\ \pi_j^{BIPmod} & , \pi_j^{ref} = 0;\ x_j^{incumb} = 0;\ index = 0;\ j \neq catalyst \\ \dfrac{\pi_j^{ref} - \pi_j^{BIPmod}}{|\pi_j^{ref}|} & , \pi_j^{ref} \neq 0;\ x_j^{incumb} = 1;\ index = 1;\ j \neq catalyst \\ \dfrac{\pi_j^{BIPmod} - \pi_j^{ref}}{|\pi_j^{ref}|} & , \pi_j^{ref} \neq 0;\ x_j^{incumb} = 0;\ index = 0;\ j \neq catalyst \\ -M & otherwise \end{cases} \tag{12}$$

5. CORE-LP-LS effectiveness

Before implementing any heuristic or meta-heuristic algorithm to solve the CPs, we investigate first the effectiveness of the CORE-LP-LS framework by solving the core problems using Exact methods. In this section, we first describe the used benchmark instances followed by our algorithm implementation. We then show the experiment results.

5.1. Benchmark Instances

We use the first two benchmark instances that were used in [31]. The two sets can be downloaded from http://grafo.etsii.urjc.es/optsicom/binaryss/#instances. Each set has 48 instances. The first and second benchmark sets have $n = 100$ and $n = 250$, respectively. Instances in both sets have different number of R_{knap} and R_{cover} constraints. Thus, the names of the instances are written to show first the number of variables, followed by the number of R_{knap} constraints, and then R_{cover} constraints. For example, instance 250-5-2-0-1 indicates that

this instance has 250 variables, 5 R_{knap} constraints, and 2 R_{cover} constraints.

5.2. IP implementation

We use the IP solver of *CPLEX 12.10* to solve the CPs. We use the default settings of the branch and bound (B&B) algorithm in *CPLEX 12.10*; however, to reduce the algorithm execution time, we expedite the execution of the algorithm by following three rules:

- Do not solve the CP if its LP-relaxation value, UB, is less than the current z_{BIP}^{incumb}.
- After improving the UB using cutting planes, if UB is less than the current z_{BIP}^{incumb}, then we stop solving the CP. We implement an *information callback*[1] that is called every time a new UB is found.
- Terminate the algorithm if z_{BIP}^{incumb} is equal to or better than BKV of the solved instance, where BKVs are found from [31]. If we fail to find BKV, then we terminate the algorithm once we check all variables in I^{index}.
- To reduce the computation time, we ranks CPs based on $|N^{core}|$, i.e., we solve first CPs that have small number of variables.

We empirically choose $r = 40$ to do this experiment, which was enough to create core problems that lead to feasible solutions for instances having $n = 100$ and $n = 250$. For the initial problem, if solving the CP having $|N^{core}| = 40$ does not lead to a feasible solution, then we increase $|N^{core}|$ by five variables. This expansion was needed once, with instance 250-10-10-1-2. Note that we do not use probing with the first BIP. Moreover, we use $Q = 10$ in the probing algorithm. Table 1 shows the results for instances having 100 and 250 variables respectively.

5.3. Experiment

We summarize the results of our experiment to solve the two sets of benchmark instances in Table 1. We describe here columns 1–4 that show the results of instances having 100 variable; however, columns 5–8 are the same, except they are used to describe the results of instances having 250 variables. After the instance name in columns 1, we report the instance BKV in column 2, followed by the best framework value (BFV) obtained, where framework stands for the CORE-LP-LS framework. Thus, the best CORE-LP-LS-IP solutions are denoted by BFV-IP in Table 1. To give an insight about the solved CPs, we report some statistics about the CPs in column 4. We report the minimum, maximum, and the average number of core variables in the solved CPs that were needed to execute the CORE-LP-LS-IP algorithm. Moreover, we report how many times the CORE-LP-LS-IP algorithm updates the incumbent solution, and the number of solved CPs using the exact IP algorithm. For example, for instance 100-5-2-0-0, show the following values in column 4: 40, 40, 40, 1, 36. These values mean that all CPs had 40 variables, x^{incumb} was changed once and the algorithm terminated after solving 36 CPs.

Inferior and superior results relative to the BKVs are underlined and written in bold, respectively. Table 1 shows that we are able to find all the BKVs for instances having $n = 100$, except for two instances. Similarly, for the 250 variable instances, we missed four BKVs; however, we were able to identify 9 new BKVs. The BKVs of the six missed instances were obtained by increasing r to 60 or increasing Q to 20.

A solution showing no changes in x_{BIP}^{incumb} means that the BKV was obtained from solving the first CP. Due to probing, the maximum CP sizes solved were 59 and 62 for the 100-variable and 250-variable instances, respectively.

[1] https://www.ibm.com/support/pages/ilog-cplex-manuals

6. CORE-LP-LS Efficiency

Although we solve several CPs that are smaller in size than the BIP, using exact methods to solve these CPs is still time-consuming, especially if we do not solve them in parallel. Thus, we replace the exact B&B algorithm that was used in the previous experiments with the TS-based algorithm of [31], which can be considered as the state-of-the-art algorithm to solve MDMKP instances. We compare the new algorithm, which we call CORE-LP-LS-TSTS to the TSTS algorithm. We try to find if embedding the TSTP algorithm within the CORE-LP-LS framework would improve the TSTS results, and not increase its computation time.

The processor used to conduct all the experiments reported in this paper is the 2.20 GHz Core(TM) i7-3632QM. We do not benchmark our processor to the one used in [31]; instead, we run the TSTS algorithm using our processor. The TSTS source code is available from http://www.info.univ-angers.fr/pub/hao/mdmkp.html.

In addition to the two sets of instances solved in the previous section, [31] have also solve instance having $n = 500$ that can be downloaded from http://people.brunel.ac.uk/~mastjjb/jeb/info.html. Instances in this set are characterized by having $m = q = 30$. CPs' sizes of this set ranged between 50 and 190.

6.1. CORE-LP-LS-TSTS implementation

The TS-based algorithm of [31], TSTS, uses two improvement operations, namely flip and swap. The algorithm searches for a solution of any size in the first stage, while in the second stage, the algorithm searches solutions having the size of stage one solution. In both stages, the two search operations, flip and swap, are used. Please note that by solution size, we mean the number of variables included in the optimal solution. For more details about the algorithm, readers are referred to [31].

To replace B&B with TSTS, we implement the following:

First CP. The first CP is solved using a B&B algorithm for instances having $n = 100$ and $n = 250$, while we use the TSTS algorithm for instances having $n = 500$. Using two algorithms in the CORE-LP-LS framework proves the flexibility of the algorithm by implementing various solution techniques. For example, users can use B&B, TSTS, or any other algorithm depending on the CPs' sizes.

Starting TSTS Solutions for following CPs. For all CPs, except the first one that we solve to find the first x_{BIP}^{incumb}, the starting solution of any CP is found by modifying x_{BIP}^{incumb}. Basically, variables in N^{adj} are fixed to the values found by Algorithm 1, whereas variables in N^{core} are copied from x_{BIP}^{incumb}, except for the catalyst variable.

Termination Criterion Per CP. The termination criterion used in [31] is based on time. In our implementation, we terminate phase I and phases II of the TSTS algorithm if the solution does not improve for 10,000 iterations.

Termination Criterion for the CORE-LP-LS-TSTS algorithm. We use two termination criterion. For instances having $n = 100$ and $n = 250$, we solve 50% of the CPs resulting from the last x_{BIP}^{incumb}. For example, assume that $|I^{\#}| = 50$, then we solve the first 25 CPs. Now, if x_{BIP}^{incumb} was updated after solving 10 CPs, and based on the new x_{BIP}^{incumb}, we have $|I^{\#}| = 60$ then we solve another 30 CPs. We terminate the CORE-LP-LS-TSTS algorithm if after solving the 30 CPs, x_{BIP}^{incumb} is not updated. For instances having $n = 500$, we use time as a termination criterion.

Hash Vectors. We use the same hash functions that were used in [31]. In [31] all the neighbours of a solution are checked. In our implementation, we only check a neighbourhood limited by N^{core}. In a first implementation of the algorithm, we used to re-initialize the hash vectors when we solve a new CP; however, this implementation lead to inferior results compared to the current implementation

where we initialize the hash vectors once at the start of the algorithm.

6.2. Comparison Experiment

Tables 2 and 3 compare the CORE-LP-LS-TSTS to the stand-alone TSTS algorithm for instances having $n = 100$ and $n = 250$, respectively. Both tables start by the instance name. For the CORE-LP-LS-TSTS solutions, we report the best and average solutions first, followed by the average numbers of core changes, the number of solved CPs and computation time.

To compare CORE-LP-LS-TSTS to TSTS, we solve each instance, using both algorithms, ten times using ten different seeds. As a termination criterion for the TSTS algorithm, we increase the average CORE-LP-LS-TSTS computation time needed to solve each instance by 20%. Thus, computation time is biased towards TSTS.

We use boldface numbers to show if one of the two compared algorithms reached a better solution than the other one. Moreover, we use superscript * to show new BKVs.

For instances having $n = 500$ and $m = q = 30$, we do not solve the first CP using an exact method; instead, we solve it using TSTS. Moreover, we allow the algorithm to start with a small core size. If stage 1 of the TSTS algorithm fails to find a feasible solution, then we increase the core size by five variables and try again to find a starting feasible solution.

Unlike the experiments reported in Tables 2 and 3, we check if algorithm CORE-LP-LS-TSTS is capable of finding the BKVs. Thus, for each instance, we allow the algorithm to run for 600 s and report the best and average results found from 10 runs. On the other hand, we allow the TSTS algorithm to run for the same time and report the best and average solutions of 10 runs as well. In column 2, after the instance name, we cite the BKVs that were reported in [31].

6.3. Analysis

We compare first CORE-LP-LS-TSTS with TSTS in terms of best results obtained by both algorithms. Table 5 summarize the results shown in Tables 2–4. We report first in column 1 the set name, followed by the number of instances in each set in column 2. Column 3 shows if any of the best results exceeds the BKVs reported in [31]. In columns 4–5, we show the number of instances for which the CORE-LP-LS-TSTS algorithm achieves better and worse results, respectively, when compared to the TSTS results. Finally, in columns 6–7, we show the number of better and worse average results when comparing CORE-LP-LS-TSTS with TSTS.

Table 5 shows that the number of better results obtained by the CORE-LP-LS-TSTS algorithm increases with the increase in the instance size. For instances having $n = 100$, the CORE-LP-LS-TSTS algorithm got better results for only three instances, compared to 7 better instances that were found by the TSTS algorithm. However, when comparing instances having $n = 250$ or $n = 500$, then the CORE-LP-LS-TSTS algorithm performs better than the TSTS algorithm. For example, better results were obtained for 23 out of the 30 instances for the third set that has $n = 500$. Moreover, the number of new BKV that the CORE-LP-LS-TSTS algorithm finds for instances having $n = 100$, $n = 250$, and $n = 500$ are 0, 5, and 22, respectively. Note that for instances having $n = 250$, the number of new BKVs found by the CORE-LP-LS-IP algorithm was 9, of which 4 of these BKVS were also identified by the CORE-LP-LS-TSTS algorithm. Thus, we were able to obtain six new BKVs for instances having $n = 250$ using the CORE-LP-LS framework.

For average solutions, TSTS was slightly better than CORE-LP-LS-TSTS only for instances having $n = 100$. For the other two sets of instances, the average results obtained by CORE-LP-LS-TSTS are better than TSTS. The total results show that out of the 126 solved instances, CORE-LP-LS-TSTS was able to get 79 better average solutions compared to 21 for TSTS. Based on [9], average solutions are a better measure to assess algorithms since best results are over-optimistic concerning an algorithm performance.

We can attribute better average solutions to the following. First, solving a problem formed from 20% − 40% of the original problem variables reduces the solution variability because the search region is smaller. Moreover, the BKVs of several instances were found by solving the first CP, i.e., instances for which the number of core changes is 0 in Tables 2 and 3. Similarly, for instances having $n = 500$, better solutions than the current BKVs were obtained from the first CP or after inspecting a small number of CPs.

In our implementation, we did not use parallel processors. Moreover, we used a first-move LS strategy to reduce computation time. Considering the number of solved CPs in Tables 2 and 3, the average number of CPs did not exceed 50 CP. Also, for instances having $n = 500$, we only solved less than 10 CPs to report the solutions in Table 4. Algorithm efficiency can improve if we solve the CPs in parallel because we can inspect several CPs at the same time. However, the TSTS algorithm does not allow the use of parallel processors due to the use of hash vectors.

The TSTS algorithm uses hash vectors and functions to avoid re-inspecting some solution points. As stated earlier, re-initializing the hash vectors with every CP deteriorated the performance of the CORE-LP-LS-TSTS algorithm; thus, we initialize them once at the start. The neighbourhoods of solutions marked as visited in the TSTS algorithm consider the whole neighbourhood of the solution; however, in the CORE-LP-LS-TSTS algorithm, solutions are marked visited without checking the whole neighbourhood of the solution, only the neighbourhood in N^{core}. This was the reason for not finding the current or new BKVs for some of the instances in Table 4.

7. Probing importance

The probing step would increase the CPs' sizes, but does this increase in the core sizes improves the algorithm performance? In this experiment, we check if probing is essential to the effectiveness of our algorithm. Consequently, we choose 10 instances from Table 1, for which the size of the solved CPs exceeded 50 variables, and we skip the probing step, i.e., we keep the sizes of all CPS as 40 variable

Table 6 shows the results of this experiment. Columns 2 and 3 of Table 6 shows the solution values found by the CORE-LP-LS-IP algorithm with and without probing, respectively. Columns 4 and 5 show the

Table 5
Comparison between CORE-LP-LS-TSTS and TSTS in terms of best and average results reported in Tables 2–4.

Set	Number Instances	New BKVs	CORE-LP-LS-TSTS vs TSTS			
			Best		Average	
			Better	Worse	Better	Worse
100	48	0	3	7	8	14
250	48	5	22	6	47	1
500	30	22	23	7	24	6
Total	126	27	48	20	79	21

Table 6
Effect of not using probing on the CORE-LP-LS-IP algorithm effectiveness.

instance	Best probing	Best no probing	Number of x_{BIP}^{incumb} Changes	Number of solved CPs
100-10-5-0-0	21852	21852	2	41
100-10-5-0-3	20596	*20594*	3	48
100-10-5-1-0	10018	10018	1	5
100-10-5-1-3	10544	10544	1	22
100-10-10-0-0	22054	*22029*	1	29
250-5-5-0-1	60795	*60785*	2	143
250-10-5-0-1	59619	*59604*	5	207
250-10-5-1-2	25737	*25719*	0	67
250-10-10-0-1	53745	*53720*	2	97
250-10-10-1-3	26684	*26608*	1	82

number of x_{BIP}^{incumb} updates and the number of solved CPs, respectively. Table 6 clearly shows that probing has improved the quality of results; the solution values of 7 out of the ten instances have deteriorated without probing.

8. Conclusions and future work

The CORE-LP-LS framework is a solution methodology by which we can avoid solving a large BIP and instead solve several CPs derived from the BIP. Each CP has associated adjunct variables that have fixed values. Classifying variables as core or adjunct ones depends on the reduced costs that we obtain from the LP-relaxation solutions. A CP solution, along with the adjunct variables' values, leads to a BIP solution, and we use an LS heuristic to move from a CP to a better one.

We define neighbouring CPs by forcing one or more variables, catalyst variables, to have values different than their values in the BIP solution associate with the CP solution. We modify the BIP to take into account the catalyst variables' new values. We then create a new CP based on the modified BIP. We also enlarge the core variables' set using a probing heuristics and solve the resulting CP. If the resulting CP is better than the current incumbent solution, then we update it like any LS algorithm. The probing step is experimentally found to improve the effectiveness of the CORE-LP-LS framework.

The framework is flexible. We used three variants of this framework. One that uses an IP solver, CORE-LP-LS-IP. Another variant that uses TSTS algorithm, CORE-LP-LS-TSTS. We also use a hybrid variant in which we solve the first core problem using an IP solver, while we solve the rest of the core problems using TSTS.

To check the effectiveness, efficiency, and probing importance, we solve a set of 126 instances having different sizes. The framework outperforms previous algorithms when solving large instances. We were able to identify 28 new best solutions that were not identified before, mostly for large instances. Moreover, even if other algorithms are better in terms of finding the best solution value by running the algorithm several times, the average solutions obtained by the CORE-LP-LS framework are better. Thus, the CORE-LP-LS framework variants provided competitive algorithms to the state-of-the-art one.

For future work, we need to develop a parallel implementation version of this algorithm. A parallel implementation enables us to use better LS strategies like best-move strategy instead of the first-move strategy used in this work since computation time would decrease. Trying other neighbourhoods definitions is necessary, especially if we want to use our algorithm to solve other problems.

More analysis is also needed to map solutions' landscapes to the problem parameters [39]. Landscape analysis usually studies how meta-heuristic and heuristic solutions navigate through the search regions. The CORE-LP-LS framework involves such navigation as defined by the neighbourhood; however, defining CPs and searching an area, instead of evaluating a solution point, needs further investigation.

Author Statement

I have done everything related to this paper

Declaration of Competing Interest

The author declares that there is no conflict of interest

References

[1] Al-Shihabi S. A hybrid of max–min ant system and linear programming for the k-covering problem. Computers & Operations Research 2016;76:1–11.
[2] Al-Shihabi S, Ólafsson S. A hybrid of nested partition, binary ant system, and linear programming for the multidimensional knapsack problem. Computers & Operations Research 2010;37(2):247–55.
[3] Angelelli E, Mansini R, Speranza MG. Kernel search: A general heuristic for the multi-dimensional knapsack problem. Computers & Operations Research 2010;37(11):2017–26.
[4] Angelelli E, Mansini R, Speranza MG. Kernel search: A new heuristic framework for portfolio selection. Computational Optimization and Applications 2012;51(1):345–61.
[5] Arntzen H, Hvattum LM, Løkketangen A. Adaptive memory search for multidemand multidimensional knapsack problems. Computers & operations research 2006;33(9):2508–25.
[6] Balas E, Zemel E. An algorithm for large zero-one knapsack problems. operations Research 1980;28(5):1130–54.
[7] Beaujon GJ, Marin SP, McDonald GC. Balancing and optimizing a portfolio of r&d projects. Naval Research Logistics (NRL) 2001;48(1):18–40.
[8] Belvaux G, Wolsey LA. bcprod: A specialized branch-and-cut system for lot-sizing problems. Management Science 2000;46(5):724–38.
[9] Birattari M, Dorigo M. How to assess and report the performance of a stochastic algorithm on a benchmark problem: mean or best result on a number of runs? Optimization letters 2007;1(3):309–11.
[10] Boyer V, Elkihel M, El Baz D. Heuristics for the 0–1 multidimensional knapsack problem. European Journal of Operational Research 2009;199(3):658–64.
[11] Cappanera P, Gallo G, Maffioli F. Discrete facility location and routing of obnoxious activities. Discrete Applied Mathematics 2003;133(1-3):3–28.
[12] Cappanera P, Trubian M. A local-search-based heuristic for the demand-constrained multidimensional knapsack problem. INFORMS Journal on Computing 2005;17(1):82–98.
[13] Chen H. Fix-and-optimize and variable neighborhood search approaches for multi-level capacitated lot sizing problems. Omega 2015;56:25–36.
[14] Chu PC, Beasley JE. A genetic algorithm for the multidimensional knapsack problem. Journal of heuristics 1998;4(1):63–86.
[15] Danna E, Rothberg E, Le Pape C. Exploring relaxation induced neighborhoods to improve mip solutions. Mathematical Programming 2005;102(1):71–90.
[16] Federgruen A, Meissner J, Tzur M. Progressive interval heuristics for multi-item capacitated lot-sizing problems. Operations Research 2007;55(3):490–502.
[17] Fischetti M, Lodi A. Local branching. Mathematical programming 2003;98(1):23–47.
[18] Fréville A. The multidimensional 0–1 knapsack problem: An overview. European Journal of Operational Research 2004;155(1):1–21.
[19] Gortázar F, Duarte A, Laguna M, Martí R. Black box scatter search for general classes of binary optimization problems. Computers & Operations Research 2010;37(11):1977–86.
[20] Guastaroba G, Speranza MG. Kernel search: An application to the index tracking problem. European Journal of Operational Research 2012;217(1):54–68.
[21] Hanafi S, Freville A. An efficient tabu search approach for the 0–1 multidimensional knapsack problem. European Journal of Operational Research 1998;106(2-3):659–75.
[22] Hanafi S, Wilbaut C. Improved convergent heuristics for the 0-1 multidimensional knapsack problem. Annals of Operations Research 2011;183(1):125–42.
[23] Hansen P, Mladenović N. Variable neighborhood search. Handbook of metaheuristics. Springer; 2003. p. 145–84.
[24] Helber S, Sahling F. A fix-and-optimize approach for the multi-level capacitated lot sizing problem. International Journal of Production Economics 2010;123(2):247–56.
[25] Hill RR, Cho YK, Moore JT. Problem reduction heuristic for the 0–1 multidimensional knapsack problem. Computers & Operations Research 2012;39(1):19–26.
[26] Huston S, Puchinger J, Stuckey P. The core concept for 0/1 integer programming. Fourteenth Computing: The Australasian Theory Symposium (CATS2008). 2008.
[27] Hvattum LM, Arntzen H, Løkketangen A, Glover F. Alternating control tree search for knapsack/covering problems. Journal of Heuristics 2010;16(3):239–58.
[28] Hvattum LM, Løkketangen A. Experiments using scatter search for the multidemand multidimensional knapsack problem. Metaheuristics. Springer; 2007. p. 3–24.
[29] Kellerer H, Pferschy U, Pisinger D. Multidimensional knapsack problems. Knapsack problems. Springer; 2004. p. 235–83.
[30] Laabadi S, Naimi M, El Amri H, Achchab B, et al. The 0/1 multidimensional knapsack problem and its variants: a survey of practical models and heuristic approaches. American Journal of Operations Research 2018;8(05):395.
[31] Lai X, Hao J-K, Yue D. Two-stage solution-based tabu search for the multidemand multidimensional knapsack problem. European Journal of Operational Research 2019;274(1):35–48.
[32] Magazine MJ, Chern M-S. A note on approximation schemes for multidimensional knapsack problems. Mathematics of Operations Research 1984;9(2):244–7.
[33] Mansini R, Speranza MG. Coral: An exact algorithm for the multidimensional knapsack problem. INFORMS Journal on Computing 2012;24(3):399–415.
[34] Martello S, Toth P. A new algorithm for the 0-1 knapsack problem. Management Science 1988;34(5):633–44.
[35] Martins JP, Ribas BC. A randomized heuristic repair for the multidimensional knapsack problem. Optimization Letters 2020:1–19.
[36] Pirkul H. A heuristic solution procedure for the multiconstraint zero-one knapsack problem. Naval Research Logistics (NRL) 1987;34(2):161–72.
[37] Pisinger D. An expanding-core algorithm for the exact 0–1 knapsack problem. European Journal of Operational Research 1995;87(1):175–87.
[38] Pisinger D. Core problems in knapsack algorithms. Operations Research 1999;47(4):570–5.
[39] Pitzer E, Affenzeller M. A comprehensive survey on fitness landscape analysis. Recent advances in intelligent engineering systems. Springer; 2012. p. 161–91.

[40] Plastria F. Static competitive facility location: an overview of optimisation approaches. European Journal of Operational Research 2001;129(3):461–70.
[41] Puchinger J, Raidl GR, Pferschy U. The core concept for the multidimensional knapsack problem. European Conference on Evolutionary Computation in Combinatorial Optimization. Springer; 2006. p. 195–208.
[42] Romero Morales MD, Carrizosa Priego EJ. Semi-obnoxious location models: A global optimization approach. European Journal of Operational Research, 102 (2), 295-301 1997.
[43] Sahling F, Buschkühl L, Tempelmeier H, Helber S. Solving a multi-level capacitated lot sizing problem with multi-period setup carry-over via a fix-and-optimize heuristic. Computers & Operations Research 2009;36(9):2546–53.
[44] Soyster A, Lev B, Slivka W. Zero-one programming with many variables and few constraints. European Journal of Operational Research 1978;2(3):195–201.
[45] Toledo CFM, da Silva Arantes M, Hossomi MYB, França PM, Akartunalı K. A relax-and-fix with fix-and-optimize heuristic applied to multi-level lot-sizing problems. Journal of Heuristics 2015;21(5):687–717.
[46] Wilbaut C, Hanafi S. New convergent heuristics for 0–1 mixed integer programming. European journal of operational research 2009;195(1):62–74.