A Memetic Algorithm Based on Probability Learning for Solving the Multidimensional Knapsack Problem

Zuocheng Li, Lixin Tang, Senior Member, IEEE, and Jiyin Liu

Abstract—The multidimensional knapsack problem (MKP) is a well-known combinatorial optimization problem with many real-life applications. In this article, a memetic algorithm based on probability learning (MA/PL) is proposed to solve MKP. The main highlights of this article are two-fold: 1) problem-dependent heuristics for MKP and 2) a novel framework of MA/PL. For the problem-dependent heuristics, we first propose two kinds of logarithmic utility functions (LUFs) based on the special structure of MKP, in which the profit value and weight vector of each item are considered simultaneously. Then, LUFs are applied to effectively guide the repair operator for infeasible solutions and the local search operator. For the framework of MA/PL, we propose two problem-dependent probability distributions to extract the special knowledge of MKP; that is, the marginal probability distribution (MPD) of each item and the joint probability distribution (JPD) of two conjoint items. Next, learning rules for MPD and JPD, which borrow ideas from competitive learning and binary Markov chain, are proposed. Thereafter, we generate MA/PL’s offspring by integrating MPD and JPD, such that the univariate probability information of each item as well as the dependency of conjoint items can be sufficiently used. Results of experiments on 179 benchmark instances and a real-life case study demonstrate the effectiveness and practical values of the proposed MKP.

Index Terms—Combinatorial optimization, Markov chain, competitive learning, memetic algorithm (MA), multidimensional knapsack problem (MKP), probability learning.

I. INTRODUCTION

THE MULTIDIMENSIONAL knapsack problem (MKP) is a generalized version of the conventional 0-1 knapsack problem. It is to select a subset of given items and put them into a multidimensional knapsack in order to maximize the total profit subject to the limited capacity of each dimension of the knapsack.

MKP is one of the most popular combinatorial optimization problems with rich practical applications, including cargo loading [1], cutting stock [2], portfolio selection [3], capital budgeting [4], and resource allocation in distributed systems [5].

Let $\mathcal{N}$ be the set of given items, $\mathcal{N} = \{1, \ldots, N\}$, $\mathcal{M}$ be the set of dimensions, $\mathcal{M} = \{1, \ldots, M\}$, $W_j$ be the capacity of dimension $j \in \mathcal{M}$, $p_i$ be the profit of item $i \in \mathcal{N}$, and $w_{ij}$ be the weight of item $i$ in dimension $j$. Then, MKP can be formulated as follows:

\[
\text{Max } f = \sum_{i \in \mathcal{N}} p_i x_i \tag{1}
\]

Subject to \[
\sum_{i \in \mathcal{N}} w_{ij} x_i \leq W_j \quad \forall j \in \mathcal{M} \tag{2}
\]

\[
x_i \in \{0, 1\} \quad \forall i \in \mathcal{N} \tag{3}
\]

where $x_i$ defined in (3) is a 0-1 decision variable indicating whether item $i$ is selected. The objective of model (1) is to maximize the total profit. Constraints (2) ensure that the constraint of each dimension must be satisfied.

MKP is known to be NP-hard [6], and so the size of the problem that can be solved by exact algorithms [7]–[9] is limited. Even the latest exact algorithms (such as [9]) still have difficulties for instances with $N = 250$ and $M = 30$. In this case, metaheuristics can be suitable solution methods because they can obtain near-optimal solutions within acceptable running time [10]. Memetic algorithm (MA) is a metaheuristic which uses a hybrid search model integrating evolutionary algorithms [11], [12] and local search to achieve a good balance between the global exploration and the local exploitation. It has been proved that MAs are effective for many combinatorial optimization problems [13]–[16] and, in particular, for MKP [17]–[19]. This motivates our MA search framework for MKP.

Learning-based mechanisms have been widely advocated in enhancing the search performances of MAs [20]–[22]. For most learning-based MAs, certain kinds of probability models...
are adopted to extract problem-specific knowledge and generate offspring. As Gupta and Ong [23] pointed out, accurate probability models are useful for improving the global exploration of MAs. In this sense, suitable probability models are critical to learning-based MAs, which in general can be represented in different forms, such as a probability distribution or other learning-based networks [24], depending on the problem being solved. The above works motivate us to propose an MA based on probability learning (MA/PL) to solve MKP, in which a problem-specific probability model is introduced.

There are various kinds of probability models in existing learning-based metaheuristics, such as estimation of distribution algorithm (EDA) [25]–[28], cross-entropy algorithm (CEA) [29], and ant colony optimization (ACO) [30]. In MA/PL, a simple probability model (i.e., probability distribution) is used. The probability model of MA/PL is different from EDA, CEA, and ACO in that it considers the problem-specific knowledge of MKP. Based on the structures of MKP, the probability model of MA/PL can reveal the inherent relationships among items. We not only consider individual items but also the dependency of two conjoint items. Two kinds of the probability distribution are used in MA/PL: 1) the marginal probability distribution (MPD) for each item to be selected and 2) the joint probability distribution (JPD) for two conjoint items to be selected simultaneously, whereas MPD reflects the univariate relationships of items, and JPD reflects the connections of conjoint items. Complementing each other, MPD and JPD work together to improve the global exploration ability of MA/PL. Very recently, Cai et al. [31], [32] proposed a novel dual population framework which used an external archive (ExA) to store elite solutions found during the search process. It can make good use of the information from ExA for generating high-quality solutions. In MA/PL, we introduce an ExA as well, which is used to build MPD and JPD.

Infeasible solutions might be generated during the search process of MA/PL. In this case, introducing suitable constraint handling techniques is crucial for the search performance [33]. In existing works, the concept of utility has been shown useful for designing repair operators for MKP. In [34] and [35], two novel utility-based repair operators for MKP, that is, RO2 and RO3, were proposed. RO2 and RO3 are guided by the pseudoutility ratio of each item that is related to the objective function and the constraint violation. It is not uncommon, however, that the values of the objective function and the constraint violation are in different magnitude. Thus, normalization is necessary. In this article, we advance existing utility-based repair operators and embed the logarithmic utility function (LUF) [36] into the proposed repair operator. LUF can obtain a composite value by weighting the preference scales of the objective function and the constraint violation. Thereby, we propose an LUF-based repair operator (RO_LUF). RO_LUF dynamically considers the profits of items and their contributions to the current overload of each dimension, so it is different from RO2 and RO3.

Local search is another important component of MAs. Given any feasible solution of MKP, local search can be viewed as an inverse process of the repair operator and so may be guided by an LUF similar to that used in the repair operator. We propose a simple yet effective local search based on LUF (LS_LUF), where neighboring solutions are generated according to another LUF that considers the profits of the items and spare capacity they need in each dimension. Such a problem-specific LS_LUF intends to enhance the local exploitation ability of MA/PL.

The framework of the proposed MA/PL is given in Fig. 1. As shown in Fig. 1, it starts with a randomly generated initial population. The individuals in the population are repaired and improved using RO_LUF and LS_LUF and then the best ones among them are selected to form the initial ExA. Next, the probability models MPD and JPD are built by using ExA, and used to generate a population of the next generation. The individuals in the new population are repaired and improved, the resulting feasible individuals are used to update ExA, and the iteration continues until some stopping criteria are satisfied.

We summarize the contributions of this article as follows.

1) Based on the features of MKP, we propose two LUFs considering the profit of each item and the overload/remaining capacity of each dimension. The LUFs are used to measure the tradeoff between the objective and the capacity constraint when performing the repair operation and the local search operation.

2) We propose the probability learning rules for MPD and JPD, which are derived from competitive learning and the binary Markov chain. Especially, JPD considers the dependency of two conjoint items in generating solutions of MKP. Integrating MPD and JPD strengthens the global search of MA/PL.

3) We introduce a random route in MA/PL’s sampling operator, such that the overall dependency of items can be potentially revealed based on the dependency of two conjoint items.

The remainder of this article is organized as follows. Section II introduces related work on MA and MKP. In Section III, based on the solution representation of MKP, the proposed RO_LUF and LS_LUF are described. Section IV presents MA/PL in detail. Results of experiments on 179 benchmark instances and a real-life case study are presented in Sections V and VI. Finally, Section VII draws conclusions and proposes future work.

II. RELATED WORK

MAs have been successfully adopted to solve various optimization problems. A comprehensive review of MAs can be found in [37]. MA is naturally suitable for combinatorial
optimization problems, due to the fact that local search can provide basic building blocks in discrete search domains. There have been only a few studies that use MA to solve MKP. Özcan and Başaran [17] proposed a genetic algorithm (GA) with hill climbing heuristic. Rezoug et al. [18] proposed two GA-based MAs which incorporated a stochastic local search and a simulated annealing. Sörensen and Sevaux [19] reported an MA with a population management strategy. These works show that MAs can achieve good performance for MKP and so motivate the proposed MA-based search model.

Machine learning has shown powerful ability for enhancing the search performance of MAs. Ong and Keane [21] proposed a meta-Lamarckian learning to improve the local search of MAs. Rakshit et al. [22] reported an adaptive MA based on Q-learning for multirobot path planning. Indeed, the foremost highlight of learning-based MAs is the probability model. Different kinds of probability models have been proposed in previous learning-based metaheuristics. Liang et al. [25] proposed an EDA with a Gaussian probability model for the function optimization. In [26] and [27], EDA and MAs with Gaussian probability models were used to solve multimodal problems. Shim et al. [28] integrated an underlying EDA into the multiobjective search model. The above works show that suitable probability models should reflect the features of problems studied. These also motivate the proposed probability models of MA/PL.

Research on MKP started in the 1950s, from seminal works of Lorie and Savage [38], Markowitz and Manne [39], Loulou and Michaeides [40], and Magazine and Oğuz [41], [42]. In [43], a comprehensive review of studies on MKP is provided. Generally, solution methods for MKP can be classified as optimization-based algorithms and metaheuristics.

There are a number of optimization-based algorithms proposed for solving MKP which apply optimization methods as part of the solution process. Hill et al. [44] proposed a problem reduction heuristic for MKP, where they first formulated MKP as a Lagrangian dual problem and then presented a heuristic for solving the estimated core problem. Yoon [45] reported a Lagrangian method using the concept of Lagrangian capacity/relaxation for MKP. Rong et al. [46] solved a kind of multicriteria MKP with k-min objectives using hybrid algorithms. Other optimization-based methods, such as branch and bound [1], [7] and collaborative methods [8], were also proposed to solve MKP. Most of the above algorithms can only get optimal solutions for instances of small or medium sizes. Although some of them can obtain near-optimal solutions for large-sized ones, such as [8], [44], and [45], the computational time may increase dramatically when the scale of instances increases.

Metaheuristics are more suitable for solving large-scale MKP. Chih [47] reported a particle swarm optimization (PSO) with a self-adaptive check and a repair operator (SACRO) for MKP. Zhang et al. [48] solved MKP using a binary artificial algae algorithm (BAAA) derived from the artificial algae algorithm. Zouache et al. [49] presented a quantum-inspired firefly algorithm with PSO (QIFAPSO) for MKP. López et al. [50] proposed a hybrid algorithm with a binary PSO and genetic operators (HPSOGO) for MKP. Bhattacharjee and Sarmah [51] proposed two metaheuristics for MKP, that is, BCSA and BFA, based on the cuckoo search and firefly algorithm in continuous domains. Other metaheuristics were also used to solve MKP, such as binary differential search (TR-BDS) [52], binary harmony search (NBHS2) [53], k-means transition ranking cuckoo search (KMTR-Cuckoo) and black hole (KMTR-BH) [54], hybrid binary differential evolution with RO3 (HBDE-RO3) [35], modified multiverse optimization (MMVO) [60], and teaching-learning-based optimization (TLBO) [61]. The above papers showed the advantages of metaheuristics for MKP.

However, for MKP, studies that incorporate machine learning into metaheuristics are relatively unexplored. Wang et al. [34] presented a hybrid EDA (HEDA) for MKP, which adopted a probability matrix to record the probability distribution of elite solutions. Arin and Rabadi [55] proposed a randomized priority search model (Meta-RaPS) for MKP, whereby presented a Meta-RaPS EDA and a Meta-RaPS Q by integrating EDA and Q-learning into Meta-RaPS. Rojas-Morales et al. [56] presented an ant knapsack algorithm based on opposite learning for MKP. The above works show that there is still a gap between machine learning and metaheuristics for solving MKP.

III. PROPOSED PROBLEM-DEPENDENT HEURISTICS

In this section, we describe the solution representation and the problem-dependent heuristics, that is, RO_LUF and LS_LUF that are used as repair and local search operators in MA/PL.

A. Solution Representation

In MA/PL, a string of N bits is used to represent a solution. Value 1 of a bit represents that the item is selected while value 0 means not selected. Given any solution, it may or may not be feasible. If infeasible, we can repair it by taking some items out. For feasible solutions, we can try to improve it by adding more items in. In both operations, we propose to choose items based on LUFs. The idea of LUF stems from the utility concept in quality management areas [36]. We define an LUF as a metric that measures the effect of a given item on the total profit and constraints, and use LUFs to determine the priority of candidate items in RO_LUF and LS_LUF. Fig. 2 shows an infeasible solution π and candidate items in it for RO_LUF, as well as a feasible solution π′ and candidate items in it for LS_LUF.

![Fig. 2. Illustration of candidate items for RO_LUF and LS_LUF.](image-url)
B. Proposed Repair Operator Based on LUF (RO_LUF)

The repair operator for infeasible solutions can be seen as a readjustment of the load in each dimension. It is necessary to consider the overload in each dimension. Let \( \text{ol} = [\text{ol}_1, \ldots, \text{ol}_M] \) be the overloads in the \( M \) dimensions for an infeasible solution \( \pi = [\pi_1, \ldots, \pi_N] \). Each element of \( \text{ol} \) can be calculated as

\[
\text{ol}_j = \sum_{i \in N} w_{ij} \pi_i - W_j, \quad j \in M \tag{4}
\]

where the first term is the load in dimension \( j \). Obviously, a negative or zero value of \( \text{ol}_j \) indicates that the capacity constraint in dimension \( j \) is satisfied.

In case \( \text{ol}_j \) is positive for some \( j \), the solution is infeasible. To repair the solution, we take out items one by one until the solution becomes feasible. Denote \( C_r \) as the set of candidate items to be taken out (e.g., \( C_r = \{1, 2, 5, 6\} \) in Fig. 2). To determine the priority of the candidate items, we calculate an LUF solution becomes feasible. Denote \( \pi_1 \) that has the largest LUF, a feasible solution \( \pi \) can be calculated as

\[
\psi_i = \sum_{j \in M} \max\{\varepsilon, \text{ol}_j - w_{ij}\} \quad \forall i \in C_r \tag{5}
\]

where we set \( \varepsilon = 0.001 \) to ensure \( \psi_i > 0 \). Then, the value of LUF in RO_LUF is defined as

\[
\sigma_i = \lambda_1 \cdot \log\left(\max_{i \in C_r} \frac{p_i}{\psi_i}\right) + \lambda_2 \cdot \log\left(\max_{i \in C_r} \frac{\psi_i}{\psi_i}\right) \tag{6}
\]

where \( \lambda_1 \) and \( \lambda_2 \) are coefficients. Without loss of generality, we set \( \lambda_1 = \lambda_2 = 0.5 \) to represent the equal importance of the two attributes for given items. Obviously, when changing the bits of items in \( C_r \) from 1 to 0, an item that has larger LUF will result in smaller decrease of total profit while making more contribution in getting the capacities satisfied.

In RO_LUF, the item in \( C_r \) with the largest LUF is taken out, the related bit is changed from 1 to 0, and this item is taken out of \( C_r \). If the solution is still infeasible, the values of \( \psi_i \) and \( \sigma_i \) are recalculated to identify the next item to be taken out. This process continues until the solution becomes feasible. The proposed RO_LUF is given in Algorithm 1.

In the illustration example in Fig. 2, RO_LUF is applied on the infeasible solution \( \pi \). After one iteration taking out item 1 that has the largest LUF, a feasible solution \( \pi' \) is obtained.

C. Proposed Local Search Based on LUF

As an inverse process of the repair operator, we here consider the spare capacity in each dimension. For a feasible solution \( \pi' = [\pi'_1, \ldots, \pi'_N] \), let \( \text{sc} = [\text{sc}_1, \ldots, \text{sc}_M] \) be the spare capacities in the \( M \) dimensions. Each element can be calculated as

\[
\text{sc}_j = W_j - \sum_{i \in N} w_{ij} \pi'_i \quad \forall j \in M. \tag{7}
\]

Let \( C_l \) be the set of items not in the knapsack (e.g., \( C_l = \{1, 3, 4\} \) in Fig. 2), such that adding each of them alone does not result in capacity violation

\[
C_l = \{i | \pi'_i = 0 \wedge \text{sc}_j - w_{ij} \geq 0 \quad \forall i \in N \text{ for all } j \in M\}. \tag{8}
\]

\begin{algorithm}
\caption{RO_LUF}
\begin{algorithmic}[1]
\STATE \textbf{Input:} infeasible solution \( \pi \)
\STATE \textbf{Output:} feasible solution \( \pi' \), objective function \( f(\pi') \)
\STATE \textbf{Initialize the set of candidate items} \( C_r \); \textbf{Eq. (8)}
\STATE \textbf{Initialize the overloads in the} \( M \) \textbf{dimensions} \( \text{ol} = [\text{ol}_1, \ldots, \text{ol}_M] \); \textbf{Eq. (4)}
\STATE \textbf{Calculate} \( \psi_i \) \textbf{of each item} \( i \in C_r \); \textbf{Eq. (5)}
\STATE \textbf{Do}
\STATE \textbf{Calculate} \( \text{ol}_j \) \textbf{of each item} \( i \in C_r \); \textbf{Eq. (6)}
\STATE \textbf{Select an item} \( s \) \textbf{from} \( C_r \) \textbf{with the largest LUF}; \textbf{break ties arbitrarily;}
\STATE \textbf{Set} \( \psi_i = 0 \); \textbf{change the bit of item} \( s \)
\STATE \textbf{Set} \( C_r = C_r \setminus \{s\} \); \textbf{update} \( C_r \)
\STATE \textbf{Set} \( \text{ol}_j = \text{ol}_j - w_{ij}, j = 1, \ldots, M \); \textbf{update} \( \text{ol}_j \)
\STATE \textbf{Calculate} \( \psi_i \) \textbf{of each item} \( i \in C_r \); \textbf{Eq. (5)}
\STATE \textbf{While} \( (\text{ol}_j \leq 0, j = 1, \ldots, M) \)
\STATE \textbf{Select} \( \pi' = \pi \) \textbf{and calculate} \( f(\pi') \)
\STATE \textbf{Return} \( \pi', f(\pi') \)
\end{algorithmic}
\end{algorithm}

We calculate another LUF for item \( i \in C_l \), again using two attributes, profit \( p_i \) and a specially defined \( \tau_i \)

\[
\tau_i = \sum_{j \in M} (\text{sc}_j - w_{ij}) + \varepsilon, \quad i \in C_l \tag{9}
\]

where \( \varepsilon = 0.001 \) is used to guarantee \( \tau_i > 0 \). Then, this LUF for item \( i \in C_l \) is calculated as

\[
\delta_i = \lambda_1 \cdot \log\left(\max_{i \in C_l} \frac{p_i}{\tau_i}\right) + \lambda_2 \cdot \log\left(\max_{i \in C_l} \frac{\tau_i}{\tau_i}\right) \tag{10}
\]

where \( \lambda_1 = \lambda_2 = 0.5 \) are two coefficients. In LS_LUF, we identify the item with the smallest LUF and change its bit from 0 to 1.

LS_LUF is also an iterative procedure, each time choosing one item to add in, then recalculating spare capacity of each dimension, updating \( C_l \), and recalculating \( \tau_i \) and \( \delta_i \) until \( C_l \) is empty. This procedure is given in Algorithm 2.

Note that we use LS_LUF to improve only the feasible solutions that are generated by repairing infeasible solutions with RO_LUF, such as \( \pi' \) generated by repairing \( \pi \) in Fig. 2. This is because, in our preliminary tests, we observed that executing LS_LUF for all feasible solutions produced by MA/PL's global search is very likely to cause a premature convergence.
IV. MA/PL FOR MKP

In MP/PL, we use two learning rules: 1) the learning rule based on the competitive learning for MPD and 2) the learning rule based on the proposed binary Markov chain for JPD. Using the learning rules, the dependency of items can be captured when a new population is produced by using a random sampling route.

A. ExA and Updating Mechanism

ExA is updated at each generation of MA/PL, so that it always contains high-quality solutions. Let PS be the population size of MA/PL, AL the size of ExA, \( \pi^P(gen) = \{\pi^P_1(gen), \ldots, \pi^P_{PS}(gen)\} \) the population of MA/PL at generation gen, and \( \pi^e(gen) = \{\pi^e_1(gen), \ldots, \pi^e_{AL}(gen)\} \) the set of ExA at generation gen. ExA can be initialized and updated by Algorithm 3.

In Algorithm 3, ExA is incrementally updated by replacing the worst individual in it with the best in the current population if the latter is better. Thus, the information of ExA that reflects elite solutions can be gradually enriched. No more than one individual from the current population is added to ExA to avoid premature convergence. MPD and JPD are all based on ExA, so it can be seen as a set of training data for our learning rules.

B. Learning Rule Based on Competitive Learning for MPD

The ideas of the proposed learning rule for MPD derive from competitive learning [57]. The competitive learning can make good use of the information of high-quality samples in learning probability models. We represent MPD as the union of the univariate marginal distributions of all items. Then, the existing learning rule in [34] and [57] is used. However, we learn MPD from ExA rather than from elite individuals in current population, which is different from [34]. Let \( MPD_{gen} \) be the union of the marginal distribution at generation gen. It is defined as

\[
MPD_{gen} = \left[\begin{array}{c}
mp_{0,1}(gen) \\
\vdots \\
mp_{0,N}(gen) \\
mp_{1,1}(gen) \\
\vdots \\
mp_{1,N}(gen)
\end{array}\right]_{2 \times N} (11)
\]

where \( \sum_{k=0}^{1} mp_{k,i}(gen) = 1 \) (i \( \in \mathbb{N} \)) and \( mp_{k,i}(gen) \) represents the probability value for the bit of item i to be k. Note that \( MPD_{gen} \) is initialized with \( mp_{k,i}(gen) = 0.5 \) (i \( \in \mathbb{N} \) and \( k \in \{0,1\} \)). Let LR be the learning rate of MA/PL. The competitive learning method for estimating MPD is given in Algorithm 4.

From Algorithm 4, it can be seen that MPDs between two consecutive generations are connected with the learning rate LR, and the probability values of items are gradually moved toward TmpPro/AL obtained from elite solutions (line 6). This is similar to the weight update rule in competitive learning. In this case, MPD is not only determined by its previous statuses but also related to the information of the current ExA (lines 3–6). Thus, the information of elite solutions can be adequately used by MA/PL in order to predict promising search regions.

C. Learning Rule Based on Binary Markov Chain for JPD

The purpose of JPD is to get the dependency of two conjoint items. Our learning rule for JPD is motivated from the binary signal source in telegraph systems [58]. The telegraph system outputs a series of pulses with high (H) and low (L) symbols (i.e., an N-bit string) in a temporal order. It can be modeled by a Markov source with given memory length. The memory length defines the dependency structure of signals. If we only consider a dependency of two conjoint symbols, the memory length is 1 (i.e., first-order Markov source). The first-order Markov source can be analyzed by a binary Markov chain. As shown in Fig. 3, the dependency of two conjoint symbols can be captured using the binary Markov chain, such as the (i-1)th and the i\(^{th}\) signals.

It is interesting that the output of the first-order Markov source is equivalent to the solution representation of MKP (see Fig. 4), when regarding the “temporal order” as the
“spatial order.” Thus, it is reasonable to extract the bivariate dependency of conjoint items by using a binary Markov chain.

In Fig. 4, the probability distributions of points 2 to N are often different, which are determined by the features of MKP. Therefore, the nonstationary binary Markov chain is used in our learning rule, as shown in Fig. 5. There are \( N - 1 \) pairs of conjoint items for the order 2 \( \rightarrow \cdots \rightarrow N \). For each point of 2 to N, we first calculate the one-step transition probability matrix (OSTPM) from ExA and in turn obtain JPD based on OSTPM.

Let \( \text{pro}_{i,\rho}(i \in \{2, \ldots, N\} \text{ and } \rho \in \{1, \ldots, 4\}) \) be the OSTPM for the transitions of states on the \( i \)th point, where \( \text{pro}_{i,1} : (0 \rightarrow 0), \text{pro}_{i,2} : (0 \rightarrow 1), \text{pro}_{i,3} : (1 \rightarrow 1), \) and \( \text{pro}_{i,4} : (1 \rightarrow 0) \). Then, OSTPM for the \( i \)th point can be calculated using Algorithm 5.

Based on OSTPM, one can calculate JPD for each spatial point \( i \in \{2, \ldots, N\} \). Let \( \text{JPr}_i^{\rho}(\text{gen}) = [\text{JPr}_{i,\rho}(\text{gen})]_{2 \times 2} \) be the JPD for spatial point \( i \) at generation \( \text{gen} \). Then, it satisfies

\[
\sum_{k \in \{0,1\}} \sum_{l \in \{0,1\}} \text{JPr}_{i,k}^{\rho}(\text{gen}) = 1
\]

(12)

where \( \text{JPr}_{i,k}^{\rho}(\text{gen}) \) is the probability of \( \pi_{i-1} = k \) and \( \pi_i = l \) at generation \( \text{gen} \). Apparently, \( \text{JPr}_i^{\rho}(\text{gen}) \) can be seen as a discrete joint distribution of conjoint items \( i-1 \) and \( i \) in the order of \( 2 \rightarrow \cdots \rightarrow N \). Then, we can calculate \( \text{JPr}_{i,k}^{\rho}(\text{gen}) \) as

\[
\text{JPr}_{i,k}^{\rho}(\text{gen}) = \begin{cases} \left( \frac{1}{AL} - \sum_{l \in \{1,AL\}} \pi_{i-1}^{\rho}(\text{gen} - 1) \right) \cdot \text{pro}_{i,\rho} & \text{if } k = 0 \\ \left( \frac{1}{AL} - \sum_{l \in \{1,AL\}} \pi_{i-1}^{\rho}(\text{gen} - 1) \right) \cdot \text{pro}_{i,\rho} & \text{otherwise} \end{cases}
\]

(13)

where \( \rho \) is set to 1, 2, 3, and 4 for conditions \( (k = 0) \land (l = 0) \), \( (k = 0) \land (l = 1) \), \( (k = 1) \land (l = 1) \), and \( (k = 1) \land (l = 0) \), respectively.

D. Sampling Operator Based on MPD and JPD

Accurate probability models are beneficial to improve the global exploration of learning-based MAs [23]. Indeed, there are many probability models that can capture the precise dependency of all random variables, such as Bayesian networks. However, such probability models in general take more computational time. Hence, we only consider the dependency of two conjoint items in the proposed JPD. Besides, there may be a dependency between any pair of items. In this case, many such dependencies would not be considered if one fixed sampling route is always used. Thus, we propose to take a random sampling route, so that the dependency between any items can be potentially considered. In our sampling operator, we first generate a random permutation of integers \( 1, \ldots, N : \rho = \{r_1, \ldots, r_N\} \) and then use it as the sampling route, as shown in Fig. 6.

Let \( c\text{pro}(\pi_{ri} = l|\pi_{ri-1} = k), i \in \{2, \ldots, N\} \) be the conditional probability of \( \pi_{ri} = l \) given \( \pi_{ri-1} = k \) in terms of sampling route \( \rho \). We can calculate it via the conditional probability formula as

\[
c\text{pro}(\pi_{ri} = l|\pi_{ri-1} = k) = \text{JPr}_{i,k}^{\rho}(\text{gen}) / \sum_{k \in \{0,1\}} \text{JPr}_{i,k}^{\rho}(\text{gen}).
\]

(14)
Algorithm 6: Sampling Operator of MA/PL

Input: $\pi^r(gen - 1)$
Output: $\pi^r(gen)$

1. Estimate $MPr(gen)$ using competitive learning method; //Algorithm 4
2. For $k = 1$ to $PS$ do
3. Randomly generate a sampling route $r = [r_1, \ldots, r_N]$;
4. Set $\pi_{k,r}^r(gen) = 0$, for $i = 1, \ldots, N$; //initialize each individual with bit 0
5. For $i = 1$ to $N$ do
6. If $(i = 1)$ then //the first bit of sampling route $r$
7. If $(\text{random}[0,1] \leq mpri_{r,1}(gen))$ then //based on the MPD
8. $\pi_{k,r}^r(gen) = 1$;
9. End If
10. Else //the other bits of sampling route $r$
11. Calculate one-step transition probability matrix $prori_{r,\pi} = [1, \ldots, 4]$; //Algorithm 5
12. Calculate JPD $JPr^r(gen)$ based on $prori_{r,\pi}, i = 1, \ldots, 4$; //Eq. (13)
13. Obtain conditional probability $cpro(\pi_{r,\pi} = 1 | \pi_{r,\pi-1} = \pi_{k,r-1}(gen))$; //Eq. (14)
14. Set $\text{TmpPro} = cpro(\pi_{r,\pi} = 1 | \pi_{r,\pi-1} = \pi_{k,r-1}(gen)) \cdot JPr_{r,\pi}(gen)$;
15. Set $\text{TmpPro} = \text{TmpPro} / \sum_{\pi_{r,\pi}} [\pi_{r,\pi} = 1]$;
16. If $(\text{random}[0,1] \leq \text{TmpPro})$ then //based on integration of MPD and JPD
17. $\pi_{k,r}^r(gen) = 1$;
18. End If
19. End If
20. End For
21. End For
22. End For
23. Return $\pi^r(gen)$

Afterward, the offspring can be produced by integrating MPD and JPD. The sampling operator is given in Algorithm 6. In Algorithm 6, a sampling route is first generated (line 3), which is helpful in capturing more potential dependency among items. Then, the first bit of the sampling route is generated from MPD (lines 6–12), while other bits are generated by integrating JPD and MPD (lines 12–23). Thus, MPD and JPD can complement each other in the global search of MA/PL.

E. Overall Procedure of the Proposed MA/PL

Let $\text{genMax}$ be the maximum generations and $\text{gbest}(\text{genMax})$ be the best feasible individual found so far. The overall procedure of MA/PL is given in Algorithm 7. It shows that MPD and JPD are all estimated from ExA (lines 5 and 10), which is useful to get more information of promising search regions. Moreover, we use RO_LUF and LS_LUF to repair and improve new solutions (lines 4 and 9). With this precise model, MA/PL can hopefully achieve a good balance between exploration and exploitation.

F. Time Complexity of the Proposed MA/PL

The time complexities of calculating the objective function, initializing population, and sampling new individuals are all $O(PS \cdot N)$. In the worst case, the time complexities of RO_LUF and LS_LUF are all $O(PS \cdot N \cdot (3 \cdot N + M))$. The time complexities of initializing and updating ExA are all $O(AL)$. The time complexities of learning and initializing MPD are $O(N)$. The time complexity of learning JPD is $O((N - 1) \cdot AL + N)$. Thus, the total time complexity is $\text{genMax} \cdot \{O(PS \cdot N^2 + PS \cdot N \cdot M) + O(N \cdot AL)\}$.

V. EXPERIMENTAL RESULTS AND COMPARISONS

A. Experimental Setup and Parameter Settings

To verify the performance of the proposed MA/PL, we test it on four sets of well-known benchmark data for MKP. The sets are related to different sizes. Test set I contains 11 large-scale instances with $M = 15$ to 100 and $N = 100$ to 2500 (available from http://hces.bus.olemiss.edu/tools.html). Test set II contains 20 medium-scale instances with $M = 5$ and 10 and $N = 100$ (available from ORLIB [59]). Test set III contains 58 small-scale instances from $M = 2$ to 30 and $N = 10$ to 105 (available from ORLIB). Test set IV consists of 90 large-scale instances with $M = 5$ to 30 and $N = 500$ (available from ORLIB). Therefore, there are a total of 179 instances. MA/PL is coded in C++ and run on a PC with Intel Core-i7 3.4-GHz processor and 16-GB memory.

MA/PL is independently run 30 times for each instance. Results are reported with eight metrics, that is, BEST, AVG, SD, SR, Min.Dev, Ave.Dev, Max.Dev, and Var.Dev. BEST, AVG, SD, and SR are the best value, mean value, standard deviation, and success rate. Min.Dev, Ave.Dev, and Max.Dev are the minimum, average, and maximum percentage deviations from best known solutions (BKS). Var.Dev is the variance of percentage deviations. We note that the values of BKS for all the instances are collected from the most currently published works for MKP (available at http://dx.doi.org/10.13140/RG.2.2.14761.95841). In Tables I–XIV, each number in the row #Best is the total number of instances for which the relevant algorithm obtained the best result in terms of different quality metrics. Besides, we highlight the best quality metric of each instance found by relevant algorithms in bold, and mark the column of BEST of certain instances with symbol “*” if BKS is attained. To check the significance of the differences between MA/PL and other
algorithms, the p-values based on the nonparametric Friedman test at the 95% confidence level (CI) are reported as well.

In Sections V-C and V-D, we adopt the results of existing algorithms from relevant references. The average CPU times (s) (i.e., CPU (s)) of MA/PL are listed for information. Due to the differences in programming environments and computer configurations, it is hard to accurately compare CPU times of other algorithms. However, we find that CPU times of MA/PL are all acceptable and thus running times will not be further discussed.

The parameters of MA/PL include the population size \(PS\), the size of ExA \(AL\), and the learning rate \(LR\). To obtain the appropriate parameters, relevant experiments based on the design of experiment (DOE) are carried out. We set the parameters of MA/PL as follows: \(PS = 150\), \(AL = 20\), and \(LR = 0.01\). More details

Due to the differences in programming environments and computer configurations, it is hard to accurately compare CPU times of other algorithms. However, we find that CPU times of MA/PL are all acceptable and thus running times will not be further discussed.

The parameters of MA/PL include the population size \(PS\), the size of ExA \(AL\), and the learning rate \(LR\). To obtain the appropriate parameters, relevant experiments based on the design of experiment (DOE) are carried out. We set the parameters of MA/PL as follows: \(PS = 150\), \(AL = 20\), and \(LR = 0.01\). More details

Due to the differences in programming environments and computer configurations, it is hard to accurately compare CPU times of other algorithms. However, we find that CPU times of MA/PL are all acceptable and thus running times will not be further discussed.

The parameters of MA/PL include the population size \(PS\), the size of ExA \(AL\), and the learning rate \(LR\). To obtain the appropriate parameters, relevant experiments based on the design of experiment (DOE) are carried out. We set the parameters of MA/PL as follows: \(PS = 150\), \(AL = 20\), and \(LR = 0.01\). More details
TABLE V

| Instance  | HEDA1 Ave. | Dev | Var. Dev | HEDA2 Ave. | Dev | Var. Dev | MA/PL Ave. | Dev | Var. Dev | CPU (s) |
|-----------|------------|-----|----------|------------|-----|----------|------------|-----|----------|--------|
| Mk_gk01   | 0.32       | 0.08| 0.94     | 0.66       | 0.23| 0.00     | 20.07      |     |          |        |
| Mk_gk02   | 0.20       | 0.12| 0.98     | 0.40       | 0.25| 0.00     | 22.77      |     |          |        |
| Mk_gk03   | 0.20       | 0.10| 1.15     | 0.78       | 0.40| 0.00     | 40.83      |     |          |        |
| Mk_gk04   | 0.34       | 0.29| 1.27     | 0.71       | 0.51| 0.00     | 52.73      |     |          |        |
| Mk_gk05   | 0.28       | 0.22| 1.50     | 0.75       | 0.23| 0.00     | 57.19      |     |          |        |
| Mk_gk06   | 0.44       | 0.23| 1.44     | 1.16       | 0.35| 0.00     | 78.95      |     |          |        |
| Mk_gk07   | 0.72       | 0.82| 1.85     | 0.99       | 0.28| 0.00     | 211.11     |     |          |        |
| Mk_gk08   | 0.77       | 1.10| 1.73     | 0.29       | 0.40| 0.00     | 348.42     |     |          |        |
| Knap5     | –          | –   | 0.00     | 0.00       | 0.00| 0.00     | 2.02       |     |          |        |
| Knap10    | –          | –   | 0.00     | 0.00       | 0.00| 0.00     | 2.72       |     |          |        |
| Knap15    | –          | –   | 0.00     | 0.00       | 0.00| 0.00     | 3.42       |     |          |        |
| #Best     | 2          | 0   | 4        | 4          | 12  | 15       |            |     |          |        |
| p-value   | 0.157      |     | 0.004    |            |     |          |            |     |          |        |

TABLE VI

| Instance  | RaPS-EDA Ave. | Dev | Var. Dev | RaPS-Q Ave. | Dev | Var. Dev | MA/PL Ave. | Dev | Var. Dev | CPU (s) |
|-----------|---------------|-----|----------|-------------|-----|----------|------------|-----|----------|--------|
| H1        | 0.00          |     | 0.00     | 0.00        |     | 0.00     | 3.73       |     |          |        |
| H2        | 0.00          | 0.19| 0.00     | 0.00        | 4.68| 0.00     | 3.68       |     |          |        |
| Pb1       | 0.00          | 0.52| 0.00     | 3.86        |     | 0.00     | 4.47       |     |          |        |
| Pet6      | 0.04          | 0.20| 0.02     | 6.04        |     | 0.00     | 7.98       |     |          |        |
| Pet7      | 0.00          | 0.19| 0.00     | 7.98        |     | 0.00     | 10.64      |     |          |        |
| Sento2    | 0.00          | 0.02| 0.00     | 10.64       |     | 0.00     | 14.37      |     |          |        |
| Weing6    | 0.00          | 0.06| 0.00     | 4.13        |     | 0.00     | 16.30      |     |          |        |
| Weing7    | 0.00          | 0.05| 0.00     | 16.30       |     | 0.00     | 18.12      |     |          |        |
| Weing8    | 0.05          | 0.31| 0.00     | 18.12       |     | 0.00     | 20.13      |     |          |        |
| Weish18   | 0.00          | 0.13| 0.00     | 10.86       |     | 0.00     | 13.51      |     |          |        |
| Weish22   | 0.00          | 0.07| 0.00     | 12.72       |     | 0.00     | 15.52      |     |          |        |
| Weish23   | 0.00          | 0.02| 0.00     | 12.83       |     | 0.00     | 15.98      |     |          |        |
| Weish24   | 0.00          | 0.20| 0.00     | 12.61       |     | 0.00     | 18.56      |     |          |        |
| Weish25   | 0.00          | 0.09| 0.00     | 12.67       |     | 0.00     | 18.12      |     |          |        |
| Weish26   | 0.00          | 0.09| 0.00     | 14.39       |     | 0.00     | 20.13      |     |          |        |
| #Best     | 14            | 0   | 16       | 14.39       |     | 0.00     |            |     |          |        |
| p-value   | 0.157         |     | 0.000    |            |     |          |            |     |          |        |

about the parameter settings of MA/PL can be found at http://dx.doi.org/10.13140/RG.2.2.23150.56644.

B. Effectiveness of MA/PL’s Components

1) Effectiveness of Proposed Learning Rules and Local Search: To study the impact of the proposed learning rules, random sampling route, and local search, we introduce the following variants.

1) MA/PL_V1: It is different from MA/PL in that the learning method for JPD, the random sampling route and LS_LUF are removed. That is, MA/PL_V1’s search behavior is only guided by the learning and sampling operators of MPD.

2) MA/PL_V2: It is the same as MA/PL_V1 except that the learning method for JPD is added.

3) MA/PL_V3: It is obtained by replacing the fixed sampling route in MA/PL_V2 by the random sampling route.

Obviously, the comparisons of these variants and MA/PL can reveal the effectiveness of the learning rule for JPD, the random sampling route, and LS_LUF. For fair comparison, we first run MA/PL with \( genMax = 5000 \) and record the CPU time. Then, the other variants are run for the same CPU time. Note that parameters in the variants are the same as in MA/PL. Comparison results are listed in Table I. In terms of the percentage deviations from BKS, that is, \( Dev(\%) \), the related violin plots are given in Fig. 7.

From Table I, we can see that MA/PL_V2 outperforms MA/PL_V1, clearly showing that the integration of MPD and JPD enhances the searchability. That is, considering the dependency of conjoint items leads to better global exploration. Meanwhile, MA/PL_V3 outperforms MA/PL_V2 for almost all the instances in terms of AVG. Especially, there are a total of 13 instances for which the SD values of MA/PL_V3 are smaller than those of MA/PL_V2. Thus, by capturing the potential dependencies among all items, the random sampling route improves the effectiveness and the robustness of MA/PL. Moreover, MA/PL outperforms MA/PL_V3 which demonstrates the effectiveness of the proposed LS_LUF. From Fig. 7, it is clear that the distributions of MA/PL_V3 are better than MA/PL_V1 and MA/PL are better than MA/PL_V3, in terms of both the dispersions and the interquartile ranges.

In addition, we also observe the 95% CI for the solutions of MA/PL and the three variants for the other instances, that is, H2, Knap 28, MK_gk04, Pb6, Pet7, and Sento2. The related CI’s are shown in Fig. 8. We can see that MA/PL can obtain relatively more consistent solutions than the other three variants. These CI’s also reveal the performance of MA/PL’s components.

To further reveal the effectiveness of the proposed LS_LUF, two variants of MA/PL that use advanced local search methods in existing works for MKP are introduced. The two variants are MA/PL_NLS and MA/PL_ILS, which are the same as MA/PL except that LS_LUF is replaced by the neighborhood local search [62] and the intensity-based local search [34]. The same CPU times used in Table I are adopted for each algorithm. Results are given in Table II. In addition, in Fig. 9, we give the 95% CI for the solutions of MA/PL and the two variants for the same instances that are shown in Fig. 8. The results of Table II and Fig. 9 reveal the competitiveness of the proposed LS_LUF.
2) Effectiveness of the Proposed Repair Operator: To verify the effectiveness and efficiency of the proposed RO_LUF, we compare MA/PL with two variants, namely, MA/PL_RO2 and MA/PL_RO3. They are the same as MA/PL except that RO_LUF is replaced by RO2 [34] and RO3 [35]. RO2 and RO3 are all the most effective repair heuristics for MKP in existing works. Note that the same stopping criterion \((\text{genMax} = 5000)\) is used for two variants and MA/PL. Relevant results on Test set II are reported in Table III. Meanwhile, the average CPU times for three algorithms are given in Fig. 10.

Table III shows that MA/PL is better than MA/PL_RO2/3 for almost all the instances, regarding BEST and AVG. Hence, RO_LUF is beneficial to enhance the overall solution quality of MA/PL. Besides, as shown in Fig. 10, MA/PL takes less CPU time than MA/PL_RO2/3 although the same stopping criterion is adopted. These results prove the effectiveness and efficiency of the proposed RO_LUF. In terms of Dev (\%), the violin plots of MA/PL and MA/PL_RO2/3 are given in Fig. 11, showing that MA/PL can obtain more robust solutions than the other variants.

3) Discussions on Coefficients for RO_LUF and LS_LUF: The coefficients \(\lambda_1\) and \(\lambda_2\) applied for calculating LUFs play an important role on the performance of RO_LUF and LS_LUF. To study the impacts of \(\lambda_1\) and \(\lambda_2\), we introduce four variants of MA/PL, that is, MA/PL_A, MA/PL_B, MA/PL_C, and MA/PL_D, with respect to different combinations of \((\lambda_1, \lambda_2)\). The values of \((\lambda_1, \lambda_2)\) for these four variants are \((0.1, 0.9), (0.3, 0.7), (0.7, 0.3),\) and \((0.9, 0.1)\). We then assess the effectiveness and the efficiency of MA/PL and the four variants. The comparative results in terms of AVG are reported in Table IV. In addition, we give trends of SD in Fig. 12. Note that the four variants are executed with \(\text{genMax} = 5000\) which is the same as MA/PL.

Table IV shows that the values of AVG of MA/PL are much better than those of the four variants for almost all the used instances. The results verify that in RO_LUF and LS_LUF the equal weight of the two attributes for candidate items is helpful in enhancing the effectiveness of MA/PL. Meanwhile, in Fig. 12, MA/PL has relatively smaller SD for each instance, so setting the equal coefficient to the two attributes is beneficial to enhance the robustness of MA/PL as well. This can be due to the fact that for the proposed LUFs the different magnitude of the two attributes has already been normalized before using...
them. Thereby, the equal coefficient of them is more reliable to reflect the problem-specific relationships of the two attributes.

Moreover, we draw the trend of CPU (s) for each algorithm in Fig. 13. It is clear that algorithms with larger $\lambda_2$ tend to take less CPU time. This is because, during the search process of MA/PL, a larger $\lambda_2$ that corresponds to a smaller $\lambda_1$ will emphasize the role of the constraint violation but weaken the impact of the objective. Thereby, for MA/PL with a larger $\lambda_2$, the time complexities of RO LU F and LS LU F can be much lower than the worst case $O(PS \cdot N \cdot (3 \cdot N + M))$, compared with ones using smaller $\lambda_2$. Especially, the trend line of MA/PL in Fig. 13 is located in the relatively middle area among all the lines. However, the CPU time for MA/PL is quite acceptable as we can see in this section. Therefore, we set $(\lambda_1, \lambda_2) = (0.5, 0.5)$ considering both the effectiveness and efficiency of MA/PL.

### C. Comparisons With Learning-Based Metaheuristics

We compare MA/PL with four learning-based metaheuristics for MKP, that is, HEDA1 [34], HEDA2 [34], Meta-RaPS EDA [55], and Meta-RaPS Q [55]. HEDA1 and HEDA2 use the EDA-based learning scheme, whereas HEDA2 uses RO2 as the repair operator and HEDA1 uses a repair operator based on LP. Meta-RaPS EDA and Meta-RaPS Q adopt a model of memoryless metaheuristic (i.e., Meta-RaPS). Meta-RaPS EDA and Meta-RaPS Q employ EDA and Q-learning as the underlying algorithms. These algorithms are the most currently effective solution methods for MKP. The stopping criterion of MA/PL is $\text{genMax} = 5000$.

#### D. Comparisons With Other Existing Algorithms

In this section, we compare MA/PL with nine existing algorithms for MKP with respect to Test sets I–IV. These existing algorithms are listed in Table VII, which are all the most currently effective solution methods for MKP. The stopping criterion of MA/PL is $\text{genMax} = 5000$.

| Algorithm     | Published year | Test set I | Test set II | Test set III | Test set IV |
|---------------|----------------|------------|-------------|--------------|-------------|
| NBHS2 [53]   | 2015           | ×          | ×           | ×            | ×           |
| TR-BDS [52]  | 2016           | ×          | ×           | √            | ×           |
| QIFAPSO [49] | 2016           | ×          | √           | ×            | ×           |
| BCSA [51]    | 2017           | √          | ×           | √            | ×           |
| HPSOGO [50]  | 2018           | ×          | ×           | √            | ×           |
| KMT-RaCkoo [54] | 2018       | ×          | ×           | ×            | ×           |
| KMTB-H [54]  | 2018           | ×          | ×           | √            | ×           |
| HBDE-RO3 [35] | 2019          | ×          | √           | √            | ×           |
| MMVO [60]    | 2019           | ×          | √           | ×            | ×           |

Note: “×” (“•”) means the test set is relevant (irrelevant) to the algorithm.

#### C. Comparisons With Learning-Based Metaheuristics

From Table V, we can see that MA/PL outperforms HEDA1 and HEDA2. Although MA/PL’s learning mechanism is similar to HEDA1 and HEDA2, its probability distribution is estimated based on ExA, rather than the elite individuals in the population. Such a learning method of MA/PL can make good use of the information of elite individuals and so speeds up the convergence. Besides, the integration of MPD and JPD can capture valuable probability information to guide the global exploration of MA/PL. We can see from Table VI that MA/PL outperforms Meta-RaPS EDA and Meta-RaPS Q. For all the instances where any of other methods found BKS, MA/PL also found BKS, while for all other instances MA/PL found better solutions. Thus, our learning method is effective for MKP. It may be worthwhile to consider embedding our learning method into other metaheuristics for MKP in future research.
TABLE VIII

COMPARISONS OF BHS2, BCSA, TR-BDS, AND MA/PL (VERSUS TEST SET I)

| Instance | BKS | NBHS2 Min. Dev | NBHS2 Ave. Dev | NBHS2 Var. Dev | BCSA Min. Dev | BCSA Ave. Dev | BCSA Var. Dev | TR-BDS SD | MA/PL SD | MA/PL Min. Dev | MA/PL Ave. Dev | MA/PL Var. Dev | CPU (s) |
|----------|-----|----------------|----------------|----------------|----------------|----------------|----------------|----------|----------|----------------|----------------|----------------|---------|
| Mk_gl01 | 3766 | 0.19 | 0.29 | 0.09 | 0.23 | 0.09 | 3688.26 | 6.02 | 3757.03 | 1.14 | 0.19 | 0.23 | <0.01 | 20.07 |
| Mk_gl02 | 3558 | 0.18 | 0.30 | 0.12 | 0.28 | 0.08 | 3878.88 | 8.06 | 3948.00 | 2.75 | 0.08 | 0.25 | <0.01 | 22.77 |
| Mk_gl03 | 5650 | 0.16 | 0.44 | 0.12 | 0.17 | 0.07 | 5512.06 | 7.55 | 5633.37 | 4.09 | 0.28 | 0.40 | <0.01 | 40.83 |
| Mk_gl04 | 5764 | 0.12 | 0.41 | 0.12 | 0.21 | 0.05 | 5623.70 | 9.16 | 5746.20 | 3.64 | 0.16 | 0.31 | <0.01 | 52.73 |
| Mk_gl05 | 7557 | 0.16 | 0.39 | 0.11 | 0.17 | 0.08 | 7349.14 | 11.41 | 7539.70 | 4.69 | 0.11 | 0.23 | <0.01 | 57.19 |
| Mk_gl06 | 7672 | 0.20 | 0.42 | 0.17 | 0.35 | 0.12 | 7488.88 | 9.95 | 7645.10 | 4.23 | 0.25 | 0.35 | <0.01 | 78.95 |
| Mk_gl07 | 19215 | 0.15 | 0.23 | 0.06 | 0.70 | 0.12 | 18588.20 | 22.74 | 19161.27 | 6.02 | 0.23 | 0.28 | <0.01 | 211.11 |
| Mk_gl08 | 18801 | 0.36 | 0.53 | 0.09 | 0.77 | 0.11 | 18299.68 | 14.88 | 18725.33 | 9.40 | 0.27 | 0.40 | <0.01 | 348.42 |
| Mk_gl09 | 58085 | 0.18 | 0.26 | 0.06 | 0.98 | 0.08 | 56035.20 | 38.46 | 57598.70 | 25.95 | 0.72 | 0.24 | <0.01 | 1204.68 |
| Mk_gl10 | 57292 | 0.38 | 0.53 | 0.06 | – | – | 55719.48 | 25.50 | 56825.43 | 23.07 | 0.33 | 0.81 | <0.01 | 1779.39 |
| Mk_gl11 | 95231 | 0.46 | 0.64 | 0.09 | – | – | 93132.92 | 35.92 | 94124.03 | 0.94 | 0.01 | 0.01 | <0.01 | 2411.12 |

4) Computational Results for Test Set IV: Results concerned with Test set IV are given in Tables XII–XIV. In Table XII, the AVG values of MA/PL are better than those of other algorithms. Meanwhile, the SD of TR-BDS, BCSA, and MA/PL are given in Table XI. It is worth noting that Table X MA/PL can get the BKS values for all instances of Test set III. In addition, the SD values of MA/PL are all obviously smaller than those of HBDE-RO3 and HPSOGO. Table XI shows that MA/PL can obtain more competitive performance than TR-BDS and BCSA. The above facts prove the effectiveness and robustness of our MA/PL for solving the small-scale instances of MKP. More details about the comparative results associated with Test set III are available at http://dx.doi.org/10.13140/RG.2.2.23779.71208.

Fig. 14. Violin plots for KMTR-Cuckoo, KMTR-BH, and MA/PL.
VI. REAL-LIFE CASE STUDY

Recently, the authors conducted a real-life project for an iron and steel company in Qian’an, Hebei, China. In this project, we focus on the production planning for the cold rolling of the iron and steel production process. Cold rolling is the most complex process in the iron and steel industry. The primary procedure of the cold rolling production planning is the order selection (OS), which involves selecting suitable orders (items) from a given order pool, on the basis of the capability of processing units (dimensions) and the due dates of orders. The sketch of OS is given in Fig. 15. We consider five bottleneck units in OS, that is, acid pickling line (APL), pickling line tandem (PLT), continuous annealing 1 and 2 (CA1 and CA2), and 20-high cold rolling mill (RCM). Note that in a given planning interval, each unit has a limited capacity (constraint) that is determined by the maintenance activity and the already assigned load.

To improve customer satisfaction, orders that are closer to their due dates should be given higher priority to be selected. We thus model OS as an MKP where the objective to be maximized is the total inverse of the time difference between the current time and the due date of each order. Besides, constraints considered here are the limited capacity of each unit in the current planning interval. In this section, OS is used as a case study and the related factory data are collected (available from http://dx.doi.org/10.13140/RG.2.2.21263.12964).

In this case study, MA/PL is independently run 30 times with the stopping condition $\text{genMax} = 5000$. Besides, we especially invite two experienced schedulers (i.e., Schedulers A and B) from the company to manually make the decision of OS by using the same factory data as MA/PL. The results of MA/PL and the manual
method are given in Table XV (details can be found at http://dx.doi.org/10.13140/RG.2.2.10777.36960).

From Table XV, it is clear that MA/PL outperforms, in terms of BEST and AVG, the manual method. Especially, MA/PL can select 306 orders with SD = 0.00, which is much better than the two schedulers. Moreover, the value of CPU (s) of MA/PL is 77.14 s, which is acceptable for the efficiency requirements of the company. Furthermore, we give the trend of objective found so far at each generation of MA/PL in Fig. 16, showing that MA/PL has a good convergence feature to solve OS. Toward that end, MA/PL can be seen as an effective and efficient solution method for practical applications.

VII. CONCLUSION

This article proposed an MA based on probability learning (i.e., MA/PL) to solve MKP. The primary highlights of this work lie in two aspects, that is, the problem-dependent heuristics and the learning-based search framework of MA/PL.

Based on the features of MKP, we proposed two problem-dependent heuristics for the repair operation and local search of MA/PL, that is, RO_LUF and LS_LUF. These heuristics fully considered the tradeoff between the objective and the violation of constraints according to the presented LUFs. The framework of MA/PL adopted two kinds of probability distribution, that is, MPD and JPD. MPD was concerned with the univariate probability distributions of items, whereas JPD focused on the bivariate dependency of conjoint items in a given spatial order. The learning scheme of MPD and JPD were all based on elite solutions recorded in ExA. To be specific, MPD was first estimated by the canonical competitive
learning. And then JPD was constructed using the proposed learning rule based on a special binary Markov chain. By integrating and complementing MPD and JPD, more accurate probability models can be constructed accordingly. Moreover, we introduced a random sampling route based on the solution structures of MKP to reveal the overall dependency among all items by means of the bivariate dependency of two joint items. Thereby, promising offspring of MA/PL can be produced by the integration of MPD and JPD. Results of experiments and comparisons on 179 well-known benchmark instances and a real-life case study verified the effectiveness and practical values of the proposed MA/PL.

For further research, effective metaheuristics may be designed based on machine learning and mathematical programming to address other types of knapsack problems, such as the multiobjective multidimensional knapsack problem (MOMKP), the quadratic knapsack problem (QKP), and the multichoice knapsack problem (MCKP).

REFERENCES

[1] S. Wei, “A branch and bound method for the multiobjective zero-one knapsack problem,” J. Oper. Res. Soc., vol. 30, no. 4, pp. 369–378, 1979.
[2] P. C. Gilmore and R. E. Gomor, “The theory and computation of knapsack functions,” Oper. Res., vol. 14, no. 6, pp. 1045–1074, 1966.
[3] Y. Shou, W. Xiang, Y. Li, and W. Yao, “A multiagent evolutionary algorithm for the resource-constrained project portfolio selection and scheduling problem,” Math. Probl. Eng., vol. 2014, no. 4, pp. 1–9, 2014.
[4] H. M. Weingartner and D. N. Ness, “Methods for the solution of the multidimensional 0/1 knapsack problem,” Oper. Res., vol. 15, no. 1, pp. 83–103, 1967.
[5] B. Guvish and I. H. Pirkul, “Allocation of databases and processors in a distributed data processing,” in Management of Distributed Data Processing, J. Akola, Ed. Amsterdam, The Netherlands: North-Holland, 1982, pp. 215–231.
[6] P. C. Chu and J. E. Beasley, “A genetic algorithm for the multidimensional knapsack problem,” J. Heuristics, vol. 4, no. 1, pp. 63–86, 1998.
[7] R. Breu and C. A. Burdet, “Branch and bound experiments in zero-one programming,” Approaches to Integer Programming (Mathematical Programming Studies), vol. 2. Berlin, Germany: Springer, 2009, pp. 1–50.
[8] J. Puchinger, G. Raidl, R. Nether, and U. Pferschy, “The multidimensional knapsack problem: Structure and algorithms,” INFORMS J. Comput., vol. 22, no. 2, pp. 250–265, 2010.
[9] R. Mansini and M. G. Speranza, “CORAL: An exact algorithm for the multidimensional knapsack problem,” INFORMS J. Comput., vol. 24, no. 3, pp. 399–415, 2012.
[10] X. Cai, H. Sun, Q. Zhang, and Y. Huang, “A grid weighted sum Pareto local search for combinatorial multi and many-objective optimization,” IEEE Trans. Cybern., vol. 49, no. 9, pp. 3586–3598, Sep. 2019.
[11] X. Cai et al., “The collaborative local search based on dynamic-constrained decomposition with grids for combinatorial multiobjective optimization,” IEEE Trans. Cybern., early access, Aug. 15, 2015, doi: 10.1109/TCYB.2019.2931434.
[12] F. Caraffini, A. V. Kononova, and D. Corne, “Infeasibility and structural bias in differential evolution,” Inf. Sci., vol. 496, pp. 161–179, Sep. 2019.
[13] L. Wang, S. Y. Wang, and Y. Xu, “An effective hybrid EDA-based algorithm for solving multidimensional knapsack problem,” Expert Syst. Appl., vol. 39, no. 5, pp. 5593–5599, 2012.
[14] Y. He, X. Zhang, and W. Li, “An efficient binary differential evolution algorithm for the multidimensional knapsack problem,” Eng. Comput., to be published.
[15] P. Kumar, P. B. Barua, and J. L. Gaindhar, “Quality optimization (multi-characteristics) through Taguchi’s technique and utility concept,” Qual. Rel. Eng. Int., vol. 16, no. 6, pp. 475–485, 2015.
[16] X. Chen, Y.-S. Ong, M.-H. Lim, and K. C. Tan, “A multi-facet survey on memetic computation,” IEEE Trans. Evol. Comput., vol. 15, no. 5, pp. 591–607, Oct. 2011.
[17] J. H. Lorie and L. J. Savage, “Three problems in rationing capital,” J. Bus., vol. 28, no. 4, pp. 229–239, 1955.
[18] H. M. Markowitz and A. S. Manne, “On the solution of discrete programming problems,” Econometrica, vol. 25, no. 1, pp. 84–110, 1957.
[19] R. Loulou and E. Michailescu, “New greedy-like heuristics for the multidimensional 0–1 knapsack problem,” Oper. Res., vol. 27, no. 6, pp. 1101–1114, 1979.
[20] M. I. Magazine and O. Oguz, “A fully polynomial approximation algorithm for the 0–1 knapsack problem,” Eur. J. Oper. Res., vol. 8, no. 3, pp. 270–273, 1981.
