Dual Probability Learning Based Local Search for the Task Assignment Problem

Zuocheng Li, Lixin Tang, *Senior Member, IEEE*, and Jin-Kao Hao

***Abstract*— The task assignment problem (TAP) is concerned with assigning a set of tasks to a set of agents subject to the limited processing and memory capacities of each agent. The objective to be minimized is the total assignment cost and total communication cost. TAP is a relevant model for many practical applications, yet solving the problem is computationally challenging. Most of the current metaheuristic algorithms for TAP adopt population-based search frameworks, whose search behaviors are usually difficult to analyze and understand due to their complex features. In this work, unlike previous population-based solution methods, we concentrate on a single trajectory stochastic local search model to solve TAP. Especially, we consider TAP from the perspective of a grouping problem and introduce the first probability learning-based local search algorithm for the problem. The proposed algorithm relies on a dual probability learning procedure to discover promising search regions and a gain-based neighborhood search procedure to intensively exploit a given search region. We perform extensive computational experiments on a set of 180 benchmark instances with the proposed algorithm and the general mixed integer programming solver CPLEX. We assess the composing ingredients of the proposed algorithm to shed light on their impacts on the performance of the algorithm.**

***Note to Practitioners*—This work is motivated by the problem of program modules designing and task allocation in parallel and distribution systems. It can also be applied to deal with job (task) grouping problems in practical industrial applications. This article presents a novel and effective learning-based local search algorithm to obtain high-quality solutions for the considered problem. The results of numerical experiments and comparisons show that our algorithm can achieve good search performances on problem instances of different scales and difficulties. Afterward, we use the proposed solution method to solve a real-life open-order slab assignment problem, which is derived from the production planning of silicon steel for an iron and steel company. The learning techniques of the proposed algorithm are of general interest and can be used in search algorithms for solving other real-life optimization problems with grouping features. For future research, we will design solution methods based on these learning techniques to address other practical optimization problems.**

***Index Terms*— Learning based search, neighborhood search, task assignment.**

Manuscript received August 26, 2020; accepted October 4, 2020. Date of publication November 2, 2020; date of current version January 6, 2022. This article was recommended for publication by Associate Editor K. Paynabar and Editor Y. Ding upon evaluation of the reviewers' comments. This work was supported in part by the Major International Joint Research Project of the National Natural Science Foundation of China under Grant 71520107004, in part by the Fund for Innovative Research Groups of the National Natural Science Foundation of China under Grant 71621061, in part by the Major Program of the National Natural Science Foundation of China under Grant 71790614, in part by the National Natural Science Foundation of China under Grant 71602025, and in part by the 111 Project under Grant B16009. *(Corresponding author: Lixin Tang.)*

Zuocheng Li is with the Key Laboratory of Data Analytics and Optimization for Smart Industry (Northeastern University), Ministry of Education, Shenyang 110819, China (e-mail: zuocheng_li@163.com).

Lixin Tang is with the Institute of Industrial and Systems Engineering, Northeastern University, Shenyang 110819, China (e-mail: lixintang@mail.neu.edu.cn).

Jin-Kao Hao is with the Department of Computer Science, LERIA, Université d'Angers, 49045 Angers, France, and also with the Institut Universitaire de France, 75231 Paris, France (e-mail: jin-kao.hao@univ-angers.fr).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TASE.2020.3030397.

Digital Object Identifier 10.1109/TASE.2020.3030397

NOMENCLATURE

BKS	Best known solution of a TAP instance.
CI	Confidence interval.
DOE	Design of experiment.
DPLS	Dual probability learning-based local search approach.
ETC	Expected time to compute matrix.
GNS	Gain-based neighborhood search procedure.
HBMO	Honeybee mating optimization.
IDE	Improved differential evolution.
JPD	Joint probability distribution.
LPD	Local probability distribution.
MPD	Marginal probability distribution.
NGHS	Harmony search algorithm.
OPT	Optimal solution of certain instance.
OSAP	Open-order slab assignment problem.
PLS	Probability learning local search method.
TAP	Task assignment problem.
TIG	Task interaction graph.

I. INTRODUCTION

THE TAP [1] involves assigning N tasks to M agents subject to the constrained processing capability and memory capacity of each agent. The objective is to minimize the total cost consisting of two parts: 1) the total assignment cost of assigning tasks to agents and 2) the total communication cost between tasks that are assigned to different agents.

Formally, let $\mathbb{N} = \{1, \ldots, N\}$ be the set of tasks, $\mathbb{M} = \{1, \ldots, M\}$ the set of agents, c_{ij} the assignment cost of assigning task $i \in \mathbb{N}$ to agent $j \in \mathbb{M}$, $e_{i,k}$ the communication cost between tasks $i \in \mathbb{N}$ and $k \in \mathbb{N}$ when they are assigned to different agents, p_i the processing requirement of task i, q_i the memory requirement of task i, P_j the processing capacity of agent j, and Q_j the memory capacity of agent j. TAP can

be formulated as follows [2]–[4]:

$$\min \sum_{i=1}^{N}\sum_{j=1}^{M} c_{ij}x_{ij} + \sum_{i=1}^{N-1}\sum_{k=i+1}^{N} e_{ik}\left(1 - \sum_{l=1}^{M} x_{il}x_{kl}\right) \quad (1)$$

$$\text{s.t.} \sum_{j=1}^{M} x_{ij} = 1 \quad \forall i \in \mathbb{N} \quad (2)$$

$$\sum_{i=1}^{N} p_i x_{ij} \le P_j \quad \forall j \in \mathbb{M} \quad (3)$$

$$\sum_{i=1}^{N} q_i x_{ij} \le Q_j \quad \forall j \in \mathbb{M} \quad (4)$$

$$x_{ij} \in \{0, 1\} \quad \forall i \in \mathbb{N}, j \in \mathbb{M} \quad (5)$$

where x_{ij} is the decision variable indicating whether task i is assigned to agent j. Objective (1) is to minimize the total cost where the two terms correspond to the total assignment cost and the total communication cost, respectively. Constraints (2) require that each task can only be assigned to a unique agent. Constraints (3) and (4) ensure that the total processing and memory requirements of the tasks assigned to an agent must be less than the corresponding capacities of the agent. Constraints (5) define the binary decision variables.

TAP is encountered in a number of real-life applications, such as program modules designing [5], community-aware task allocation [6], multirobot task allocation [7], and naval task grouping [8]. TAP has also a number of variants with regard to different practical scenarios (see the review of Section II). Next, we share a real-life application that can be described by the model of TAP.

Over the past three years, the authors conducted a real-life application project for an iron and steel company in Hebei, China. Our work was to make the plant-wide production planning for silicon steel.[1] As shown in Fig. 1, the production of silicon steel involves multistage processes, i.e., ironmaking, steelmaking-continuous casting, hot rolling, and cold rolling. The company uses the make-to-order strategy, and the production planning depends closely on customer orders. Due to the batch features and the uncertainties of the production process, slabs coming out from the steelmaking-continuous casting may not perfectly match with orders. Thereby, open-order slabs, which do not belong to any orders, are inevitably created. The undesirable open-order slabs will increase the inventory cost and block the production process. Hence, we propose to consider an OSAP, which can be modeled as TAP. In OSAP, the sets of open-order slabs (tasks) and orders (agents) are identified first. Then, the cutting cost (assignment cost) of matching each open-order slab with each order is calculated. We also consider another kind of cutting cost (communication cost), concerned with two open-order slabs stemming from the same original slab. There is a limited weight (memory capacity) for each order. Moreover, the assigned slabs for each order that have not arrived at the inventory yard must be smaller than a given threshold value (processing capacity) to guarantee due dates and improve customer satisfaction. In this article, OSAP is used as a case study, as detailed in Section V.

[1]Silicon steel is usually used as magnetic materials for electric motors, transformers, electric instruments, and so on.

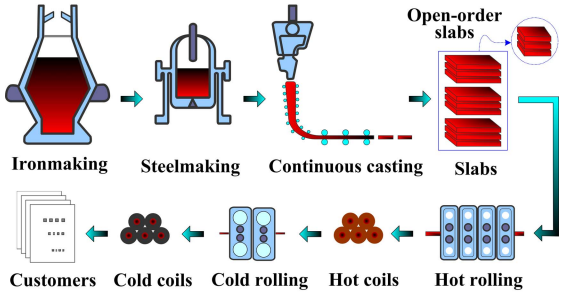


Fig. 1. Sketch of the production process of silicon steel.

In spite of its importance and application focuses, solving TAP is computationally challenging because the problem is known to be NP-hard even when only three agents are considered [9], [10]. In this sense, exact algorithms are limited to solve small-sized TAP instances. For large instances, metaheuristics whose goals are to find high-quality solutions within an acceptable computation time frame are preferred. Generally, metaheuristics for combinatorial optimization are classified as a single trajectory method and population-based method [11]. In the search space, the former builds a trajectory based on which a single solution is produced at each iteration. The latter, on the contrary, uses a search model to describe the emergence phenomenon of collective intelligence and maintains a set of solutions during the search. The performance of population-based methods depends strongly on search models. However, it is usually difficult to select the most suitable search models since there are still no universal conclusions on this issue [11]. Meanwhile, the designs of components for population-based methods are complex, which should account for both search behavior and problem-specific knowledge [12], [13]. Moreover, population-based methods are usually time-consuming because they generally require a large number of objective function evaluations, especially in some complex applications [14], [15]. As we observe from the literature review in Section II, most existing metaheuristic algorithms for TAP adopt population-based search models. As such, these algorithms are often highly complex, and their behaviors are difficult to analyze and understand. In this work, unlike population-based solution methods, we focus on the single trajectory stochastic local search framework [16] that has shown to be a powerful tool for solving a great number of hard combinatorial problems, including various assignment problems [17]–[19].

Machine learning has shown a powerful ability for improving different optimization methods [20]. It can structure valuable information and so guides metaheuristics to find better solutions [21]. In existing learning-based metaheuristics, the probability learning method, which adopts a probability model to depict the distribution of promising search regions, has shown good performance for solving a number of optimization problems [22]. In general, the probability learning method first estimates the probability model from high-quality solutions and then generates new solutions using sampling

operations. In this sense, it has relevant theoretical foundations from the probability theory perspectives. Especially, we investigate a DPLS approach that is inspired by and extends the recent PLS method [23], [24] designed for solving the class of so-called grouping problems [25].

Indeed, TAP can be considered as a special grouping problem where we want to create groups of tasks for the given agents subject to respecting the given constraints while optimizing the objective of the problem. From such a perspective, we are interested in finding useful information that can help us to determine task-to-agent assignment (i.e., which task should be assigned to which agent and which tasks should stay together). As shown in [23] and [24], PLS is a relevant method that employs learning techniques to acquire knowledge (in terms of probability distributions) from high-quality solutions sampled by a local optimization procedure and then uses the learned knowledge to create promising groups that are further improved by the local search procedure. Nevertheless, the probability distributions used in [23] and [24] are estimated based on the differences between the newly generated solution and the corresponding improved one of the local search procedure. Hence, such probability distributions tend only to concentrate on the local-level features of search space. In this work, we adapt the PLS method to TAP from the perspective of the grouping problem and introduce an additional global-level learning strategy to enhance the learning capacity of the proposed algorithm. Thus, it is different from the previous solution methods of [23] and [24]. The contributions of the work can be summarized as follows.

First, in terms of the solution method, we present a probability learning-based local search algorithm to solve TAP. Especially, the proposed DPLS relies on two key and complementary components: the dual probability learning procedure and the GNS. On the one hand, by combining global-level learning and local-level learning, the dual probability learning aims to acquire problem-specific knowledge related to promising search regions where high-quality solutions could be found. The learning process of DPLS not only exploits local information related to two correlated local optima, such as in PLS of [23] and [24], but also extracts global information from a long term memory composed of previously discovered elite solutions. On the other hand, GNS ensures effective exploitation of a given search region with a dedicated neighborhood search method. To examine neighbor solutions, both cost gain and constraint violation gain are simultaneously considered with the notion of Pareto dominance. In GNS, we treat two kinds of gains as conflicting goals when constructing the neighborhood, which can balance the gains regarding the objective and the constraint violation. We will compare it with the existing gain-based local search procedure of [2] in Section IV.

Second, from a perspective of computational performance and practical applications, we introduce a real-life case study that arises from a cooperative iron and steel company. This case study demonstrates the practical values of the proposed solution methods. Moreover, we report detailed computational results for a set of 180 TAP benchmark instances, which would be valuable for future research on TAP.

Third, the idea of combining global-level learning and local-level learning used in the dual probability learning procedure is of general interest and could be advantageously adopted by search algorithms for solving other grouping problems.

The rest of this article is organized as follows. Section II is dedicated to a literature review on TAP. The proposed DPLS algorithm is presented in Section III. Numerical results and comparisons on 180 benchmark instances are reported and discussed in Section IV. In Section V, we provide the details of the introduced real-life case study. In Section VI, we draw concluding comments.

II. Related Work

TAP was first formalized by Glover *et al.* [1] in the 1970s. Since then, a number of studies on TAP (see [5], [9], [10], and [26]–[30]) have been reported. A comprehensive review of TAP can be found in [31]. In this section, we review some representative studies on models and algorithms for TAP.

TAP and its variants have been extensively used to model several real-life applications. Lee [26] presented a number of variants of TAP with respect to different practical scenarios. Hamam and Hindi [5] adopted TAP to formulate the program modules designing problem in communication networks. Wang and Jiang [6] established the community-aware task allocation model based on a TAP variant. Lee *et al*. [7] used TAP to formulate a kind of multirobot task allocation problem, so as to enhance the overall performance of the multirobot system. Karasakal *et al*. [8] employed the basic model of TAP to formulate the sector allocation problem in the naval task grouping area. TAP was also extended to address other practical applications, such as real-time task allocation in software engineering projects [32], reliability-oriented task assignment in distributed systems [33], and task assignment in mobile agent networks [34]. The abovementioned studies reveal the practical values of TAP, indicating that developing effective solution methods for solving the problem is both necessary and meaningful.

In general, existing solution methods for TAP are classified as exact algorithms and metaheuristic algorithms. Karasakal *et al*. [8] proposed a branch and bound algorithm for a naval task grouping application considered as a kind of TAP. Ernst *et al*. [9] presented exact algorithms to solve the uncapacitated and capacitated TAP and introduced a number of integer linear programming formulations and a column generation formulation. Li *et al*. [27] proposed a logarithmic method to reduce the numbers of binary variables and the inequality constraints in solving TAP. Their algorithms can achieve good performance for instances with 10, 20, and 30 tasks. Kaya and Uçar [30] presented an exact algorithm based on the A* search algorithm for the uncapacitated TAP. One notices that existing exact algorithms are usually limited to small-sized problems.

In recent years, various metaheuristic algorithms have been used to solve TAP approximately. Kang and He [2] proposed an HBMO algorithm for TAP, where they introduced a gain-based improvement approach to strengthen the local search ability of HBMO. Zou *et al*. [3] reported a harmony search

algorithm (NGHS), in which a normalized penalty function method was used to handle the constraints. Zou *et al.* [4] developed an IDE for TAP, wherein they presented an adaptive mechanism to adjust the mutation and crossover operators of IDE. Other metaheuristics, such as simulated annealing (SA) [5] and variable neighborhood search [29], were also proposed to solve TAP.

As reviewed above, most existing metaheuristic algorithms for TAP adopt the population-based search frameworks, whose search behaviors are guided by certain relative relationships among the solutions of the population. However, there may be short of analysis on the distributions of promising search regions during the search process. In this work, we advance the state of the art of solving TAP by proposing a new probability learning-based local search algorithm.

III. Dual-Probability-Learning-Based Local Search for TAP

In DPLS, we adopt an agent-based solution representation with an integer vector $\boldsymbol{\pi}$, where $\boldsymbol{\pi}_i = j$ indicates that task i is assigned to agent j. For example, $\boldsymbol{\pi} = [1, 3, 1, 3, 2]$ represents the assignment where tasks 1 and 3 are assigned to agent 1, tasks 2 and 4 are assigned to agent 3, and task 5 is assigned to agent 2. Based on this solution representation, the components of DPLS, including dual probability learning and GNS, explore the solution space of TAP. In the following, we present the general scheme of the proposed DPLS algorithm and then its components.

A. General Scheme

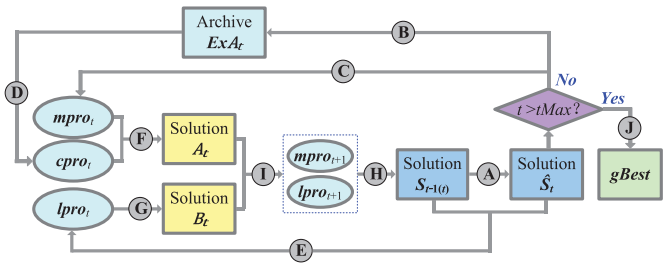
DPLS considers its search process as a special online active learning [35]. During the search process, problem-specific knowledge is learned online via its dual probability learning procedure (see Section III-B). The learned information, which is recorded in the adopted probability models (distributions), is then used to guide the algorithm toward promising search regions. In the dual probability learning, an MPD and a JPD are applied for global-level learning, whereas an LPD is employed for local-level learning. In addition to the component of the dual probability learning, DPLS uses a powerful local search method [i.e., GNS (see Section III-E)] to intensively examine a given region identified by the dual probability learning. As such, DPLS iteratively explores the given search space by alternating between the dual probability learning and GNS components to attain a suitable balance of search intensification and diversification.

Let $tMax$ be the maximum generation of DPLS, $\boldsymbol{mpro}_t = [mpro_{t,j,i}]_{M\times N}$ and $\boldsymbol{cpro}_t = [cpro_{t,i,k}]_{N\times N}$ the probability distributions of MPD and JPD at generation t, $\boldsymbol{lpro}_t = [lpro_{t,j,i}]_{M\times N}$ the probability distribution of LPD, $\boldsymbol{A}_t$ the solution generated by sampling MPD and JPD, $\boldsymbol{B}_t$ the solution generated by sampling LPD, $\boldsymbol{S}_t$ the solution selected from $\boldsymbol{A}_t$ and $\boldsymbol{B}_t$, $\hat{\boldsymbol{S}}_t$ the improved solution produced from $\boldsymbol{S}_t$ by using GNS, As the predetermined size of the external archive, $\boldsymbol{ExA}_t$ the external archive ($t \geq As$; if $t < As$, then $As = t$), and $\boldsymbol{gBest}$ the best feasible solution found so far by DPLS. Note that $cpro_{t,i,k}$ represents the probability of tasks i and k that are assigned to the same agent, while $mpro_{t,j,i}$ and $lpro_{t,j,i}$ represent that the probability of task i selects agent j. Then, the proposed DPLS algorithm for TAP is presented in Algorithm 1 with its flowchart shown in Fig. 2.

Algorithm 1 DPLS for Solving TAP

1: **Input**: parameters of TAP and DPLS
2: **Output**: $\boldsymbol{gBest}$
3: **For** $t := 1$ *to* $tMax$ **do**
4: **If** $t = 1$ **then**
5: /*initialization*/
6: Set $\boldsymbol{mpro}_{t,j,i} := 1/M, i \in \mathbb{N}, j \in \mathbb{M}$; //equal probability
7: Set $\boldsymbol{lpro}_{t,j,i} := 1/M, i \in \mathbb{N}, j \in \mathbb{M}$; //equal probability
8: Randomly generate $\boldsymbol{S}_t$ and initialize $\boldsymbol{ExA}_t$ with $\boldsymbol{S}_t$ alone; meanwhile, if $\boldsymbol{S}_t$ is feasible, then set $\boldsymbol{gBest} := \boldsymbol{S}_t$;
9: **Else**
10: /*main iteration*/
11: Perform GNS to obtain $\hat{\boldsymbol{S}}_t$ from $\boldsymbol{S}_{t-1}$; //Section III-E
12: Update $\boldsymbol{gBest}$ with a feasible and better $\hat{\boldsymbol{S}}_t$;
13: Update $\boldsymbol{ExA}_t$ by adding $\hat{\boldsymbol{S}}_t$ to it if $t \leq As$; otherwise, replacing the worst solution in $\boldsymbol{ExA}_t$ with $\hat{\boldsymbol{S}}_t$;
14: Perform global-level learning to learn (update) $\boldsymbol{mpro}_t$ using $\hat{\boldsymbol{S}}_t$; //Section III-B
15: Perform global-level learning to learn (update) $\boldsymbol{cpro}_t$ using $\boldsymbol{ExA}_t$; //Section III-B
16: Perform local-level learning to learn (update) $\boldsymbol{lpro}_t$ based on the differences between $\boldsymbol{S}_{t-1}$ and $\hat{\boldsymbol{S}}_t$; //Section III-B
17: Generate $\boldsymbol{A}_t$ from $\boldsymbol{mpro}_t$ and $\boldsymbol{cpro}_t$; //Section III-C
18: Generate $\boldsymbol{B}_t$ from $\boldsymbol{lpro}_t$; //Section III-C
19: Select solution $\boldsymbol{S}_t$ from $\{\boldsymbol{A}_t, \boldsymbol{B}_t\}$: if they are all feasible, select the one with smaller f_{tc}; if only one is feasible, select it; if they are all infeasible, select the one with smaller f_{cv};
20: Perform probability smoothing operator; //Section III-D
21: **End If**
22: **End For**
23: **If** $\{\boldsymbol{gBest}\} = \emptyset$ **then**
24: Perform repair operator for infeasible solutions; //Section III-F
25: **End If**
26: **Return**: $\boldsymbol{gBest}$

As shown in Algorithm 1, DPLS is driven by the interactions of the dual probability learning and GNS (lines 11–20). At each generation, MPD and JPD are learned (updated) by $\hat{\boldsymbol{S}}_t$ and solutions in $\boldsymbol{ExA}_t$ (lines 14 and 15). Since $\hat{\boldsymbol{S}}_t$ and $\boldsymbol{ExA}_t$ are dynamically updated along with the search process of DPLS, they always consist of high-quality solutions. MPD and JPD can, thus, reveal specific knowledge of the problem instance and search behavior of DPLS at a global level. It is probable, however, that $\boldsymbol{ExA}_t$ is not updated for a number of iterations of DPLS. In this case, MPD and JPD cannot be updated. Hence, to avoid the partial stagnation of DPLS, we do



A: GNS; **B**: Update $\boldsymbol{ExA}_t$; **C**: Learn $\boldsymbol{mpro}_t$; **D**: Learn $\boldsymbol{cpro}_t$; **E**: Learn $\boldsymbol{lpro}_t$; **F**: Generate $\boldsymbol{A}_t$; **G**: Generate $\boldsymbol{B}_t$; **H**: Select $\boldsymbol{S}_t$; **I**: Probability smoothing; **J**: Repair operator (optional).

Fig. 2. Flowchart of the proposed DPLS for TAP.

not use the solutions in $\boldsymbol{ExA}_t$ to update MPD. Note that MPD is not updated by using the best solution in $\boldsymbol{ExA}_t$ because we found in our preliminary experiments that such a way is very likely to lead to undesirable convergence of MPD. In addition, LPD is updated based on the differences of $\boldsymbol{S}_{t-1}$ and $\hat{\boldsymbol{S}}_t$ (line 16). Indeed, one can regard LPD as a "quasi-direction" of promising solutions in the current search region. Hence, LPD can capture more detailed information about the search regions that are being exploited, which is used to precisely guide GNS. Afterward, GNS is used to discover high-quality local optima with the given region (line 11), which is beneficial to reduce the sampling errors in producing $\boldsymbol{A}_t$ and $\boldsymbol{B}_t$ as well. In view of the general framework of DPLS, it highlights the integration of learning methods associated with the global and local levels and is thus different from the PLS algorithms of [23] and [24].

B. Dual Probability Learning

1) Global-Level Learning: The purpose of the global-level learning is to accumulate probability information of high-quality solutions found so far, so as to predict promising search regions. However, the learning goals for MPD and JPD are different since they account for different solution features of TAP. We update $\boldsymbol{mpro}_t$ as follows:

$$mpro_{t,J,i} = \begin{cases} mpro_{t-1,J,i} + LR, & \text{if } J = \hat{S}_{t,i} \\ mpro_{t-1,J,i}, & \text{otherwise} \end{cases}, \quad i \in \mathbb{M} \tag{6}$$

where LR is the learning rate. One observes that $\boldsymbol{mpro}_{t-1}$ and $\boldsymbol{mpro}_t$ ($t \geq 2$) are connected with LR and the differences between them are uniquely determined by $\hat{\boldsymbol{S}}_t$. Arguably, a larger value of LR results in stronger learning ability and faster convergence of $\boldsymbol{mpro}_t$, while a smaller LR value leads to relatively weaker learning ability.

Note that $mpro_{t,j,i}$ tends to become large if task i is frequently assigned to agent j regarding $\hat{\boldsymbol{S}}_t$ at each generation. Thus, $\boldsymbol{mpro}_t$ is suitable to determine "which task should go to which agent." Thereafter, we normalize $\boldsymbol{mpro}_t$ as follows:

$$mpro_{t,j,i} = mpro_{t,j,i}/(1 + LR) \quad \forall i \in \mathbb{N}, j \in \mathbb{M}. \tag{7}$$

On the other hand, the global-level learning also concerns the grouping preference about "which tasks should stay together," which is represented by $\boldsymbol{cpro}_t$ and updated by high-quality solutions in $\boldsymbol{ExA}_t$. Unlike $\boldsymbol{mpro}_t$, the elements of $\boldsymbol{cpro}_t$ are the probability values of any pair of tasks that are assigned to the same agent. The learning method of $\boldsymbol{cpro}_t$ is given in Algorithm 2.

Algorithm 2 Learning Method of $\boldsymbol{cpro}_t$ (Global Level)

1: **Input**: $\boldsymbol{ExA}_t$
2: **Output**: $\boldsymbol{cpro}_t$
3: Set $tmpAs := t$, if $t < As$; otherwise, set $tmpAs := As$; //current size of $\boldsymbol{ExA}_t$
4: Set $\boldsymbol{cpro}_{t,i,k} = 1/2 \cdot (N^2 - N), \forall i \neq k \in \mathbb{N}$; //initialization
5: /*frequency of each pair of tasks that stay together*/
6: Set $tmpCa := 0$;
7: **For** $s := 1$ *to* $tmpAs$ **do**
8: **For** each pair $(i, k) \in \{(i', k') | k' > i', \forall i', k' \in \mathbb{N}\}$ **do**
9: **If** $ExA_{t,s,i} = ExA_{t,s,k}$ **then**
10: Set $cpro_{t,i,k} := cpro_{t,i,k} + 1$; //update the probability
11: Set $tmpCa := tmpCa + 1$; //number of pairwise tasks that stay together
12: **End If**
13: **End For**
14: **End For**
15: /*normalization of probability value*/
16: Set $scalVal := 1 + tmpCa$; //scaling factor for normalization
17: **For** each pair $(i, k) \in \{(i', k') | k' > i', \forall i', k' \in \mathbb{N}\}$ **do**
18: Set $cpro_{t,i,k} := cpro_{t,i,k}/scaleVal$; //normalization
19: Set $cpro_{t,k,i} := cpro_{t,i,k}$; //symmetric feature of probability
20: **End For**
21: **Return**: $\boldsymbol{cpro}_t$

In Algorithm 2, we first initialize $\boldsymbol{cpro}_t$ with equal probability (line 4). The elements of $\boldsymbol{cpro}_t$ are then approximated by the frequencies extracted from solutions in $\boldsymbol{ExA}_t$, which is a maximum likelihood estimation. It is, thus, reliable to use Algorithm 2 for capturing the grouping preferences of tasks. Note that the elements of $\boldsymbol{cpro}_t$ to be learned are only located in the upper triangular part excluding the diagonal ones, which is beneficial to the efficiency of the learning method (line 19).

2) Local-Level Learning: The purpose of local-level learning is to discover detailed information about the current search regions. Here, we adopt the learning method in [23] and [24] proposed for the graph coloring problem. Its original idea comes from reinforcement learning, consisting of reward, penalization, and compensation operators. Let α, β, and γ be the reward factor, penalization factor, and compensation factor. The learning method of $\boldsymbol{lpro}_t$ is given in Algorithm 3.

From Algorithm 3, we can see that $\boldsymbol{lpro}_t$ is constructed based on the differences between $\boldsymbol{S}_{t-1}$ and $\hat{\boldsymbol{S}}_t$. After executing the reward (line 5), the penalization (line 7), the compensation (line 8), and the normalization (line 11) operators, the final $\boldsymbol{lpro}_t$ can be obtained. Accordingly, $\boldsymbol{lpro}_t$ can be useful to predict promising areas that may contain high-quality local optima in the current search region.

C. Sampling Method for New Solutions

At each generation of DPLS, new solutions $\boldsymbol{A}_t$ and $\boldsymbol{B}_t$ are generated by sampling the probability distributions learned

Algorithm 3 Learning Method of $\boldsymbol{lpro}_t$ (Local Level)

1: **Input**: $\boldsymbol{S}_{t-1}$, $\hat{\boldsymbol{S}}_t$
2: **Output**: $\boldsymbol{lpro}_t$
3: **For** $i := 1$ *to* N **do**
4: **If** $S_{t-1,i} = \hat{S}_{t,i}$ **then**
5: Set $lpro_{t,S_{t-1,i},i} := \alpha + (1-\alpha) \cdot lpro_{t,S_{t-1,i},i}$; //reward
6: **Else**
7: Set $lpro_{t,S_{t-1,i},i} := (1-\gamma) \cdot (1-\beta) \cdot lpro_{t,S_{t-1,i},i}$;//penalization
8: Set $lpro_{t,\hat{S}_{t,i},i} := \gamma + (1-\gamma) \cdot \frac{\beta}{M-1} + (1-\gamma) \cdot (1-\beta) \cdot lpro_{t,\hat{S}_{t,i},i}$; //compensation
9: **For** $j := 1$ *to* M **do**
10: **If** $S_{t-1,i} = j$ and $\hat{S}_{t,i} = j$ **then**
11: Set $lpro_{t,j,i} := (1-\gamma) \cdot \frac{\beta}{M-1} + (1-\gamma) \cdot (1-\beta) \cdot lpro_{t,j,i}$; //normalization
12: **End If**
13: **End For**
14: **End If**
15: **End For**
16: **Return**: $\boldsymbol{lpro}_t$

in the dual probability learning. To be specific, $\boldsymbol{A}_t$ is sampled from the combination of $\boldsymbol{mpro}_t$ and $\boldsymbol{cpro}_t$, while $\boldsymbol{B}_t$ is sampled from $\boldsymbol{lpro}_t$. From the search model of DPLS (see Fig. 2), one can see that $\boldsymbol{A}_t$ and $\boldsymbol{B}_t$ are generated from two independent learning components of DPLS. They, thus, contribute differently to the overall search behavior of DPLS. Meanwhile, the used probability distributions $\boldsymbol{mpro}_t$, $\boldsymbol{cpro}_t$, and $\boldsymbol{lpro}_t$ are concerned with different considerations. Therefore, we will adopt different sampling methods for $\boldsymbol{A}_t$ and $\boldsymbol{B}_t$.

1) Proposed Sampling Method for $\boldsymbol{A}_t$: A hybrid sampling method is proposed to generate $\boldsymbol{A}_t$, which not only depends on $\boldsymbol{mpro}_t$ and $\boldsymbol{cpro}_t$ but also a random noise. Let *noise* be the random noise and $\boldsymbol{rv} = [rv_1, \ldots, rv_{M+1}]$ the roulette vector for a given task. The proposed sampling method for $\boldsymbol{A}_t$ is given in Algorithm 4.

In Algorithm 4, the sampling manner for each task is determined by *noise* (line 7). Such a method can make good use of the randomness and so enhances the diversification of the algorithm. If the *noise* condition is not satisfied, the sampling process is executed based on the integration of $\boldsymbol{mpro}_t$ and $\boldsymbol{cpro}_t$ (lines 11–29). That is, we first sample the ith task using the roulette vector (lines 11–21). Then, if there are still tasks needed to be sampled, we select a task nL having the largest probability for staying together with task i and assign nL to agent A_i (lines 22–29). By doing so, $\boldsymbol{A}_t$ can maintain a good tracking performance for the information from both $\boldsymbol{mpro}_t$ and $\boldsymbol{cpro}_t$, so as to enhance the intensification of the algorithm.

2) Sampling Method for $\boldsymbol{B}_t$: The sampling method proposed in [23] and [24] is adopted to generate $\boldsymbol{B}_t$, and the sampling manner for each task is determined by a random noise as well. The sampling process for $\boldsymbol{B}_t$ works as follows. For each task $i \in \mathbb{N}$, the noise condition is first checked (the same as line 7 in Algorithm 4). Then, we set $B_{t,i} = random\{1, \ldots, M\}$ if the random value $random[0, 1]$ is smaller than *noise*. Otherwise, we select an agent $mL \in \mathbb{M}$ satisfying $lpro_{t,mL,i} > lpro_{t,m,i}$ and $m \in \mathbb{M} \backslash \{mL\}$ and set $B_{t,i} = mL$. As shown in Algorithm 1, selecting a solution from $\{\boldsymbol{A}_t, \boldsymbol{B}_t\}$ to determine $\boldsymbol{S}_t$ is in fact a complementing strategy for the two kinds of probability distributions of DPLS. That is, the random noises adopted in both sampling methods for $\boldsymbol{A}_t$ and $\boldsymbol{B}_t$ are consistent with each other. Thus, we use the same value of noise as in Algorithm 4.

Algorithm 4 Proposed Sampling Method for $\boldsymbol{A}_t$

1: **Input**: $\boldsymbol{mpro}_t$ and $\boldsymbol{cpro}_t$
2: **Output**: $\boldsymbol{A}_t$
3: Set $flagTask_i := false, i \in \mathbb{N}$; //flags of already assigned tasks
4: Set $nTask := 0$; //total number of already assigned tasks
5: **For** $i := 1$ *to* N **do**
6: **If** $flagTask_i = false$ **then**
7: **If** $random[0, 1] < noise$ **then**
8: Set $A_{t,i} := random\{1, \ldots, M\}$, $flagTask_i := ture$;
9: Set $nTask := nTask + 1$;
10: **Else**
11: Set $rv_1 := 0$ and $rv_{M+1} := 1$; //construct roulette vector
12: **For** $j := 2$ *to* M **do**
13: Set $rv_j := rv_{j-1} + mpro_{t,j-1,i}$;
14: **End For**
15: Set $rnd := random[0, 1)$;
16: **For** $j := 1$ *to* M **do**
17: **If** $rnd \geq rv_j$ and $rnd < rv_{j+1}$ **then**
18: Set $A_{t,i} := j$ and $flagTask_i := true$;
19: Set $nTask := nTask + 1$; Break;
20: **End If**
21: **End For**
22: **If** $nTask < N$ **then**
23: Set $\boldsymbol{T} := \{i' | flagTask_{i'} = false, i' \in \mathbb{N}\}$; //tasks need to be assigned
24: **For** each task $n \in \boldsymbol{T}$ with $cpro_{t,n,i} \neq 0$ **do**
25: Select a task nL from $\boldsymbol{T}$ that satisfies $cpro_{t,nL,i} > cpro_{t,n,i}, n \in \boldsymbol{T} \backslash \{nL\}$;
26: Set $A_{t,nL} := A_{t,i}$ and $flagTask_{nL} := true$;
27: Set $nTask := nTask + 1$;
28: **End For**
29: **End If**
30: **End If**
31: **End If**
32: **End For**
33: **Return**: $\boldsymbol{A}_t$

D. Probability Smoothing Method for $\boldsymbol{mpro}_t$ and $\boldsymbol{lpro}_t$

It is very likely that the probability distributions gradually converge to stable states (i.e., some probability values are close to 0 or 1) along with the iteration of DPLS. Such a convergence may lead to undesirable local optima, especially in some flat search regions. We found in our preliminary observations that $\boldsymbol{cpro}_t$ does not have a tendency to converge to 0 or 1. Since the learning methods of $\boldsymbol{mpro}_t$ and $\boldsymbol{lpro}_t$

depend on $\hat{\boldsymbol{S}}_t$ that may speed up the convergence, it is necessary to execute smoothing operators for them. Due to the same features of $\boldsymbol{mpro}_t$ and $\boldsymbol{lpro}_t$, we adopt the smoothing procedure [23], [24] and parameters for them.

Let *thre* be the threshold of the smoothing operator and ρ the smoothing factor. Take $\boldsymbol{mpro}_t$ for example, and the smoothing procedure works as follows. For each task $i \in \mathbb{N}$, we check the probability of $mpro_{t,j,i}$ for each agent $j \in \mathbb{M}$. If $mpro_{t,j,i} \geq thre$, we record the probability $rec = mpro_{t,j,i}$ and set $mpro_{t,j,i} = \rho \cdot mpro_{t,j,i}$. Then, we normalize the probability values for $l \in \mathbb{M}$ as $mpro_{t,l,i} = mpro_{t,l,i}/(1 - (1-\rho)\cdot rec)$. The abovementioned procedure continues until the probability values $mpro_{t,j,i}$ $(j \in \mathbb{M})$ of the ith task are all less than *thre*. The smoothing operator allows the algorithm to forget some old memories of probability distributions and, thus, enhances the diversification of DPLS.

E. Proposed Gain-Based Neighborhood Search (GNS)

In this work, we reformulate TAP in a penalty function form without constraints (3) and (4) as follows:

$$\begin{aligned} &\min\ h = \{f_{tc} + f_{cv}\} \\ &\text{s.t. Constraints (2) and (5)} \end{aligned} \tag{8}$$

where f_{tc} is the total cost [i.e., the objective function in (1)] and f_{cv} is the penalty term of constraint violations defined as

$$f_{cv} = \sum_{j\in M}\left\{\max\left\{\sum_{i\in N} p_i x_{ij} - P_j, 0\right\}\right\}^2 + \sum_{j\in M}\left\{\max\left\{\sum_{i\in N} q_i x_{ij} - Q_j, 0\right\}\right\}^2. \tag{9}$$

If $f_{cv} > 0$, the solution does not satisfy constraints (3) or (4) and, thus, is infeasible. In this case, $h = f_{tc} + f_{cv}$ is used to assess the solution quality. That is, we always use the solutions with better values of h to perform the components of DPLS, regardless of their feasibility. This is also the basic idea of penalty function strategy for handling the constraints of metaheuristics [11]. However, it is often difficult to select a suitable penalty weight, which is strongly sensitive to the problem instances and the solution methods. For these reasons, the square function in (9) is adopted to cope with this difficulty.

Given that we adopt a penalty weight λ and build another penalty term f'_{cv}, the related conclusion is as follows:

$$\lim_{f_{cv}\to\lambda^2}\left\{f'_{cv} - f_{cv}\right\} = \lim_{f_{cv}\to\lambda^2}\left\{\lambda\cdot\sqrt{f_{cv}} - f_{cv}\right\} = 0 \tag{10}$$

where $f'_{cv} = f_{cv}$ at point (λ^2, λ^2). In addition, $f'_{cv} > f_{cv}$ if $f_{cv} < \lambda^2$, while $f'_{cv} < f_{cv}$ if $f_{cv} > \lambda^2$. Therefore, the square penalty term in (9) is reasonable to address the constrains of TAP. It would be worthwhile to design other effective constraint handling methods for TAP in future research.

Based on the abovementioned reformulation, to discover high-quality local optima, we propose a GNS-based local optimization method that is based on two kinds of gains regarding the cost and the constraint violation. We use the gain calculation method proposed in [2] with the notable difference that our method is based on the notion of Pareto dominance. Especially, in GNS, we treat two kinds of gains as conflicting goals when constructing the neighborhood. This is because, in TAP, smaller assignment or communication cost for tasks does not mean smaller processing or memory requirements. Besides, the two kinds of gains are often in different magnitude. Accordingly, the two kinds of gains are, in nature, suitable to be regarded as conflicting ones. Note that the proposed Pareto-dominance GNS aims at exploiting high-quality local optima and, thus, does not concern the diversity metrics of conventional multiobjective optimization solution methods [12]. In Section IV, we will put forward relevant assessments on the effectiveness of the presented Pareto-dominance GNS, comparing DPLS with its variant that uses another GNS based on the linear sum of the two gains.

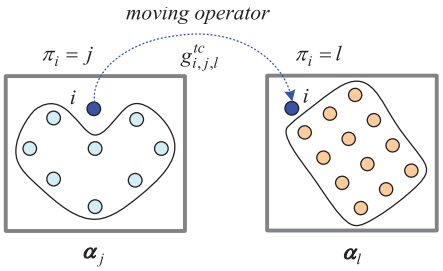


Fig. 3. Illustration of moving operator (moving task i from agents j to l).

Let $\boldsymbol{\alpha}_j$ $(j \in \mathbb{M})$ be the set of tasks assigned to agent j. Then, the gain of the cost associated with moving task $i \in \boldsymbol{\alpha}_j$ from agent j to agent $l \in \mathbb{M}\backslash\{j\}$ is defined as follows:

$$g^{tc}_{i,j,l} = \sum_{k'\in\boldsymbol{\alpha}_l} e_{ik'} - \sum_{k\in\boldsymbol{\alpha}_j\backslash\{i\}} e_{ik} + c_{ij} - c_{il} \quad \forall i \in \boldsymbol{\alpha}_j, l \in \mathbb{M}\backslash\{j\} \tag{11}$$

where we set $e_{ik'} = 0$ if $\boldsymbol{\alpha}_l = \emptyset$ and $e_{ik} = 0$ if $\boldsymbol{\alpha}_j\backslash\{i\} = \emptyset$. The moving operator of tasks in GNS is illustrated in Fig. 3.

The gain of the constraint violation $g^{cv}_{i,j,l}$ with regard to moving task i from agent j to l can be calculated according to (12)–(16) [2]. It is composed of eight subgains [see (12)] representing different scenarios of the constraint violations of agents j and l. Here, we combine the definitions of these subgains regarding the processing capacity and the memory capacity due to the same principles

$$g^{cv}_{i,j,l} = g^{cva}_{i,j,l} + g^{cvb}_{i,j,l} + g^{cvc}_{i,j,l} + g^{cvd}_{i,j,l} + g^{cve}_{i,j,l} + g^{cvf}_{i,j,l} + g^{cvg}_{i,j,l} + g^{cvh}_{i,j,l} \tag{12}$$

where

$$g^{cva(e)}_{i,j,l} = \left(\sum_{i'\in\boldsymbol{\alpha}_j} p(q)_{i'} - P(Q)_j\right)^2 - \left(\sum_{i'\in\boldsymbol{\alpha}_j\backslash\{i\}} p(q)_{i'} - P(Q)_j\right)^2 \quad \text{if } \sum_{i'\in\boldsymbol{\alpha}_j\backslash\{i\}} p(q)_{i'} > P(Q)_j, \quad \text{otherwise } 0 \tag{13}$$

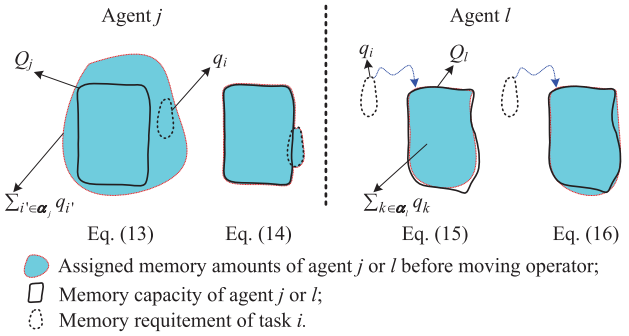


Fig. 4. Illustration of the gain of constraint violation (for memory constraints).

$$g_{i,j,l}^{cvb(f)} = \left(\sum_{i' \in \boldsymbol{\alpha}_j} p(q)_{i'} - P(Q)_j \right)^2 \text{ if } \sum_{i' \in \boldsymbol{\alpha}_j} p(q)_{i'} > P(Q)_j \text{ and } \sum_{i' \in \boldsymbol{\alpha}_j \setminus \{i\}} p(q)_{i'} \leq P(Q)_j, \quad \text{otherwise } 0 \tag{14}$$

$$g_{i,j,l}^{cvc(g)} = -\left(\sum_{k \in \boldsymbol{\alpha}_l \cup \{i\}} p(q)_k - P(Q)_l \right)^2 \text{ if } \sum_{k \in \boldsymbol{\alpha}_l} p(q)_k \leq P(Q)_l \text{ and } \sum_{k \in \boldsymbol{\alpha}_l \cup \{i\}} p(q)_k > P(Q)_l, \quad \text{otherwise } 0 \tag{15}$$

$$g_{i,j,l}^{cvd(h)} = \left(\sum_{k \in \boldsymbol{\alpha}_l} p(q)_k - P(Q)_l \right)^2 - \left(\sum_{k \in \boldsymbol{\alpha}_l \cup \{i\}} p(q)_k - P(Q)_l \right)^2 \text{ if } \sum_{k \in \boldsymbol{\alpha}_l} p(q)_k > P(Q)_l, \quad \text{otherwise } 0. \tag{16}$$

For memory constraints, we illustrate the subgains of (13)–(16) in Fig. 4. The first two figures show the conditions for (13) and (14) while the last two the conditions for (15) and (16). Note that, for calculating $g_{i,j,l}^{cv}$, we only consider $g_{i,j,l}^{cva}$ to $g_{i,j,l}^{cvh}$, and the others $g_{i',j',l'}^{cva}$ to $g_{i',j',l'}^{cvh}$ ($i' \neq i$, $j' \neq j$ and $l' \neq l$) are not changed. This is beneficial to the efficiency of GNS.

The proposed GNS procedure is shown in Algorithm 5. Two kinds of gains and nondominated set $\mathbb{P}$ are initialized first (lines 5 and 6). We build $\mathbb{P}$ based on the values of the two gains to be maximized where at least one gain is larger than 0. Then, whenever $\mathbb{P} \neq \emptyset$ (line 8), the main loop of GNS is repeated until $\mathbb{P} = \emptyset$ (lines 9–25). For each loop of GNS, a vector (i, j, l) is determined based on the proposed Rules 1–3 (lines 14–16). Since the two gains are seen as conflicting goals in Rules 1–3, the moving operator is executed to update $\boldsymbol{S}_{t-1}$ (line 17). Next, $\boldsymbol{S}_{t-1}$ is evaluated (line 19). It is very likely that $g_{i,j,l}^{tc} + g_{i,j,l}^{vc} \leq 0$ for each loop, so we always check

Algorithm 5 Proposed Pareto-Dominance-Based GNS

1: **Input**: $\boldsymbol{S}_{t-1}$
2: **Output**: $\hat{\boldsymbol{S}}_t$
3: /*initialization*/
4: Set $\hat{\boldsymbol{S}}_t := \boldsymbol{S}_{t-1}$ and $h(\hat{\boldsymbol{S}}_t) := h(\boldsymbol{S}_{t-1})$;
5: Calculate $g_{i,j,l}^{tc}$ and $g_{i,j,l}^{cv}$, $i \in \mathbb{N}$, $j = S_{t-1,i}$ and $l \in \mathbb{M} \setminus \{j\}$; //Eqs. (11) to (16)
6: Construct nondominated set $\mathbb{P}$ based on $N \times (M-1)$ pairs of gains $(g_{i,j,l}^{tc}, g_{i,j,l}^{cv})$ such that $(g_{i,j,l}^{tc} > 0) \vee (g_{i,j,l}^{cv} > 0)$, $i \in \mathbb{N}$, $j = S_{t-1,i}$ and $l \in \mathbb{M} \setminus \{j\}$);
7: /*neighborhood search*/
8: **If** $\mathbb{P} \neq \emptyset$ **then**
9: **Repeat**
10: **If** $|\mathbb{P}| = 1$ **then**
11: Perform moving operator for (i, j, l) and update $\boldsymbol{S}_{t-1}$;
12: **Else**
13: Determine a vector (i, j, l) based on following rules:
14: **Rule 1:** If there exist one or more pairs of gains in $\mathbb{P}$ satisfying $(g_{\bullet}^{tc} \geq 0) \wedge (g_{\bullet}^{cv} \geq 0)$, select (i, j, l) from such pairs with the largest $g_{\bullet}^{tc} + g_{\bullet}^{cv}$;
15: **Rule 2:** If conditions in Rule 1 are not satisfied, and there exist one or more pairs of gains in $\mathbb{P}$ satisfying $(g_{\bullet}^{tc} < 0) \wedge (g_{\bullet}^{cv} > 0)$, select (i, j, l) from such pairs with the largest $g_{\bullet}^{cv}$;
16: **Rule 3:** If conditions in Rules 1 and 2 are not satisfied, and there exist one or more pairs in $\mathbb{P}$ of gains satisfying $(g_{\bullet}^{tc} > 0) \wedge (g_{\bullet}^{cv} < 0)$, select (i, j, l) from such pairs with the largest $g_{\bullet}^{tc}$;
17: Perform moving operator for (i, j, l) and update $\boldsymbol{S}_{t-1}$;
18: **End If**
19: Update objective $h(\boldsymbol{S}_{t-1}) := h(\boldsymbol{S}_{t-1}) - (g_{i,j,l}^{tc} + g_{i,j,l}^{vc})$;
20: **If** $h(\boldsymbol{S}_{t-1}) < h(\hat{\boldsymbol{S}}_t)$ **then**
21: Set $\hat{\boldsymbol{S}}_t := \boldsymbol{S}_{t-1}$ and $h(\hat{\boldsymbol{S}}_t) := h(\boldsymbol{S}_{t-1})$; //update $\hat{\boldsymbol{S}}_t$
22: **End If**
23: Calculate $g_{i,j,l}^{tc}$ and $g_{i,j,l}^{cv}$, $i \in \mathbb{N}$, $j = S_{t-1,i}$ and $l \in \mathbb{M} \setminus \{j\}$; //Eqs. (11) to (16)
24: Construct set $\mathbb{P}$; //use the same method in Line 6
25: **Until** $(\mathbb{P} = \emptyset)$
26: **End If**
27: **Return**: $\hat{\boldsymbol{S}}_t$

the relationship between $h(\boldsymbol{S}_{t-1})$ and $h(\hat{\boldsymbol{S}}_t)$ before updating $\hat{\boldsymbol{S}}_t$ (lines 20–22). In view of the abovementioned procedures, the existing GNS method of [2], which is based on the linear sum of the two gains, can be seen as a special case of our Pareto-dominance GNS.

F. Constraint Handling Method for Infeasible Solutions

A problem-specific repair operator for infeasible solutions is proposed, which will be invoked if and only if no feasible solution can be found at the end of the search process of DPLS. In this case, during the search process of DPLS, the solution quality is evaluated based on the newly introduced objective function in (8). That is, we do not remove infeasible solutions

when updating probability distributions in the dual probability learning stage. Hence, the search process of DPLS may start from the infeasible side of the search space of TAP. The proposed repair operator is given in Algorithm 6.

Algorithm 6 Repair Operator for Infeasible Solutions

1: **Input**: $\boldsymbol{ExA}_{tMax}$
2: **Output**: $\boldsymbol{gBest}$
3: **For** $s := 1$ *to* As **do**
4: Set $SC_j := 0$ for each $j \in \mathbb{M}$; //initial spare capacities
5: **For** $i := 1$ *to* N **do**
6: Set $jj = ExA_{tMax,s,i}$ and $SC_{jj} := SC_{jj} + p_i + q_i$;
7: **End For**
8: Set $SC_j := P_j + Q_j - SC_j$ for each $j \in \mathbb{M}$; //spare capacities
9: **Repeat**
10: **For** $j := 1$ *to* M **do**
11: **If** $SC_j < 0$ **then**
12: Set $CT := \{i | ExA_{tMax,s,i} = j, i \in \mathbb{N}\}$;
13: Rank solutions in CT according to the value of $p_{ii} + q_{ii}$ $(ii \in CT)$;
14: **For** $ii := 1$ *to* $|CT|$ **do**
15: Find an agent jj from $\mathbb{M}$ that has the largest spare capacity;
16: Set $ExA_{tMax,s,ii} := jj$; //move task ii to jj
17: Set $SC_j := SC_j + p_{ii} + q_{ii}$ and $SC_{jj} := SC_{jj} - p_{ii} - q_{ii}$; **If** $SC_j \geq 0$ **then** Break;
18: **End For**
19: **End If**
20: **End For**
21: Set $nif := |\{j | SC_j < 0, j \in \mathbb{M}\}|$; //#infeasible agents
22: **Until** $(nif = 0)$
23: Calculate the objective function for solution $\boldsymbol{ExA}_{tMax,s}$;
24: **End For**
25: Find $\tilde{\boldsymbol{\pi}}$ with the smallest objective function from $\boldsymbol{ExA}_{tMax}$;
26: Set $\boldsymbol{gBest} := \tilde{\boldsymbol{\pi}}$;
27: **Return**: $\boldsymbol{gBest}$

As shown in Algorithm 6, the repair operator is an iterative procedure (lines 9–23), starting from the initialization of the spare capacity of each agent (line 8). The primary idea is to balance the spare capacity of each agent, so as to transform an infeasible solution in $\boldsymbol{ExA}_{tMax}$ into a feasible solution. We only consider infeasible agents (i.e., the spare capacities are less than 0, line 11) and move the related task to the agent having the largest spare capacity (line 15). The spare capacity of each agent and the total number of infeasible agents are dynamically updated (lines 17 and 22). We repeat these operators until a feasible solution is produced.

IV. Numerical Results and Comparisons

A. Test Instances and Experimental Design

To assess the effectiveness of the proposed DPLS, we generate 180 benchmark instances of TAP based on the work of Kaya and Uçar [30]. In their work, the relationships among tasks were described using an undirected TIG $G = (\mathbb{N}, \mathbb{E})$, where a node of $\mathbb{N}$ represents a task and $\mathbb{E}$ is the set of edges representing the communication between each pair of the task. Meanwhile, ETC $= [c_{ij}]_{N \times M}$, whose elements are the assignment cost, was also introduced. If a pair of tasks are not contained in $\mathbb{E}$, the communication cost between them is always 0 even when they stay in different agents. Hence, TIG and ETC determine the solving difficulty of the instances of TAP. As suggested in [30], an inconsistent ETC matrix is better than a consistent matrix due to the heterogeneous features of tasks. In fact, the consistent ETC matrix is a special case of the inconsistent one. The inconsistent ETC matrix can provide more general solution features of TAP, which is helpful in showing the synthetical performance of the TAP algorithms studied. Thus, to generate the test instances, we use an inconsistent ETC matrix that consists of four types, i.e., ETC $=\{0, 1, 2, 3\}$. Moreover, a communication-to-computation ratio r_{com} is introduced to determine the different impacts of the communication cost and the assignment cost on the objective function of TAP. More details about the abovementioned definitions and the generation methods of $e_{i,k}$ and $c_{i,j}$ can be found in [30].

Nevertheless, different from the approach proposed in [30], the processing capacity and the memory capacity of agents need to be considered in this work. To do so, we first generate the memory requirements q_i and the processing requirements p_i for each task as follows: $q_i = \text{random}\{1, \ldots, 100\}$ and $p_i = \text{random}\{5, \ldots, 25\}$. Such different distributions of p_i and q_i are to distinguish the two kinds of attributes of tasks. Then, the processing capacity and the memory capacity for each agent are generated as follows: $P_j = ((\sum_{i \in N} p_i)/M)/\eta$ and $Q_j = ((\sum_{i \in N} q_i)/M)/\zeta$. The parameters that we use are $\eta = \zeta = 0.8$ that represents the utilization of the average processing and memory requirements expected for each agent.

In addition to the topologies of the TIGs, five sparse DWT matrices,[2] which originally describe a set of unweighted structural problems, are adopted, including dwt59, dwt66, dwt162, dwt245, and dwt 310. Based on the abovementioned definitions and generation methods of the parameters of TAP, we generate benchmark instances for combinations $N \times M = \{59, 66, 162, 245, 310\} \times \{4, 6, 8\}$ concerned with ETC $= \{0, 1, 2, 3\}$ and $r_{\text{com}} = \{0.7, 1.0, 1.4\}$. Therefore, there are a total of 180 benchmark instances.[3]

Numerical results and comparisons on the aforementioned 180 benchmark instances are carried out. For each instance, DPLS and other compared algorithms are independently run 21 times. The results are reported as *BV*, *AV*, and *SD* that represent the best objective value, mean objective value, and standard deviation of the obtained solutions. In addition, we use #*Best* to represent the numbers of instances that an algorithm can obtain the best values of the quality indicators. To assess the statistical significance of the differences between DPLS and each compared algorithm, we conduct the nonparametric Friedman test at a 95% CI and report the *p*-values. All the algorithms are coded in C++ (Visual Studio 2008) and

[2]Topologies of DWT matrices are available from the MATRIX MARKET: https://sparse.tamu.edu/

[3]Instances are available at http://dx.doi.org/10.13140/RG.2.2.25431.62887

TABLE I
LEVELS OF PARAMETERS

Parameter	Level				
	1	2	3	4	5
α	0.05	0.10	0.15	0.20	0.25
β	0.05	0.10	0.15	0.20	0.25
γ	0.10	0.20	0.30	0.40	0.50
ρ	0.15	0.20	0.25	0.30	0.35
$noise$	0.10	0.20	0.30	0.40	0.50
$thre$	0.75	0.80	0.85	0.90	0.95
LR	0.05	0.10	0.15	0.20	0.25
As	5	10	15	20	25

executed on a PC with Intel Core i7-6700 3.4-GHz CPU and 4-GB memory.

B. Parameter Analysis and Settings

The parameters of DPLS are the reward factor α, the penalization factor β, the compensation factor γ, the smoothing factor ρ, the sampling *noise*, the smoothing threshold *thre*, the learning rate *LR*, and the size of the external archive *As*. To find an appropriate combination of parameters, we conduct the DOE, in which the orthogonal array $L_{49}(5^8)$ with 49 combinations of levels is used. The first necessity of DOE is to determine the factor levels based on the ranges of parameters. It is not uncommon, however, that certain extreme values in the ranges of parameters may cause sharp deterioration of the performance of DPLS. In this sense, we find two extreme values for each parameter, which are used as the boundary values of factor levels. Note that the determinations of such boundary values are based on preliminary observations on a number of instances with different sizes. Thereafter, the factor levels can be generated by using an arithmetic progression. Take *LR* for example, and we first identify the extreme values of *LR* with 0 and 0.3 and then give its factor levels: 0.05, 0.10, 0.15, 0.20, and 0.25. The levels of all parameters (factors) of DPLS are given in Table I. Using a medium-sized instance t162p4r1.0ETC2, for each parameter combination, we independently run DPLS 21 times with the stopping condition $tMax = 5000$ and use the values of *AV* as response values.[4] The level trends of parameters are given in Fig. 5. Moreover, we carry out the analysis of variance (ANOVA) at the 95% CI, as shown in Table II. As it shows, the p-values for all the parameters are larger than 0.05, indicating that DPLS is relatively less sensitive to its parameters and, in turn, can retain better robustness for solving different kinds of TAP instances.

Note that the selections of parameters should balance the effectiveness and efficiency of DPLS. In particular, we observe the level trends of parameters that use average CPU time as response values.[5] For example, in Fig. 5, the levels of β display a better value of margin mean corresponding to the level 5, while the related level trend of CPU time shows a relatively small difference between levels 1 and 5. Thus, the selection of β can be 0.25. Based on the abovementioned guidelines,

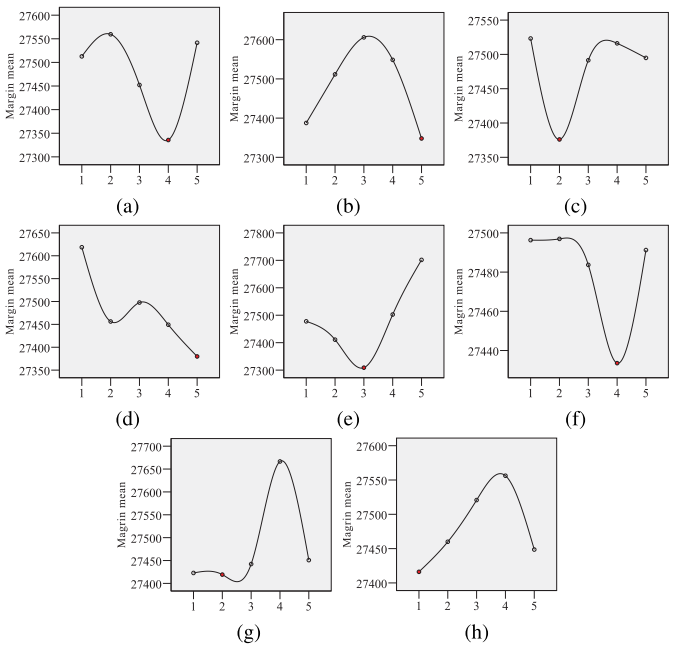


Fig. 5. Level trend of each parameter (using *AV* as response values). (a) α. (b) β. (c) γ. (d) ρ. (e) *noise*. (f) *thre*. (g) *LR*. (h) *As*.

TABLE II
ANOVA RESULTS AND PARAMETER SELECTIONS

Parameter	ANOVA			Selection	
	Mean square	F-value	p-value	Value	Rank
α	67092.973	0.923	0.475	0.20	5
β	98965.918	1.361	0.291	0.25	2
γ	46389.433	0.638	0.643	0.20	6
ρ	85565.490	1.177	0.358	0.35	4
$noise$	153632.030	2.113	0.127	0.30	1
$thre$	5648.331	0.078	0.988	0.90	8
LR	85892.153	1.181	0.356	0.10	3
As	28773.647	0.396	0.809	5	7

we give the selections of parameters in Table II. We also perform relevant experiments for the parameter settings of DPLS considering small-sized t59p6r1.4e1 and large-sized t310p4r1.0e0 and are able to confirm these settings.

C. Comparisons of DPLS and State-of-the-Art Algorithms

In this section, we compare DPLS with three state-of-the-art reference algorithms, i.e., IDE [4], HBMO [2], and NGHS [3]. To be fair, we first run DPLS for each instance ($tMax = 5000$) and record the CPU time, and then, the three algorithms are run with the same CPU time.[6] In terms of *BV*, *AV*, and *SD*, the values of #*Best* for algorithm pairs "DPLS versus IDE," "DPLS versus NGHS," and "DPLS versus HBMO" are summarized in Fig. 6. In addition, the significance of the difference between DPLS and the three algorithms is given in Table III.

From Table III and Fig. 6, we can see that DPLS outperforms the other three algorithms in terms of *BV*, *AV*, and *SD*. The corresponding p-values are all less than 0.05, except

[4]Details of the orthogonal array and the corresponding response values are available at http://dx.doi.org/10.13140/RG.2.2.17462.45124

[5]Level trends of parameters concerned with CPU time are available at http://dx.doi.org/10.13140/RG.2.2.35182.64322

[6]Details of comparison results associated with 180 individual instances are available at http://dx.doi.org/10.13140/RG.2.2.10496.87047

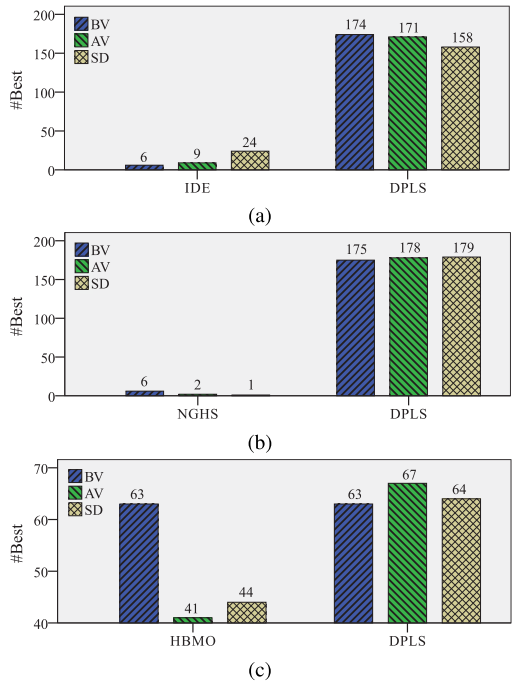


Fig. 6. Summary of results of IDE, NGHS, HBMO, and DPLS (180 instances). (a) DPLS versus IDE. (b) DPLS versus NGHS. (c) DPLS versus HBMO.

TABLE III
SIGNIFICANCE OF THE DIFFERENCE BETWEEN DPLS AND STATE-OF-THE-ART ALGORITHMS

Algorithm pair	*p*-values of two indicators			
	BV	Significance	*AV*	Significance
DPLS vs. IDE	0.000	Yes	0.000	Yes
DPLS vs. NGHS	0.000	Yes	0.000	Yes
DPLS vs. HBMO	0.089	No	0.002	Yes

that the *BV* of "DPLS versus HBMO" is 0.089. One can also find that DPLS and HBMO have the same value of #*Best* with regard to *BV*. Therefore, it is necessary to further discuss the competitiveness of DPLS and HBMO. Particularly, we calculate two *Gap* (%) values to the BKS[7]: the best relative error (*BRE*) and the average relative error (*ARE*) as follows:

$$BRE = ((BV - BKS)/BKS)) \times 100\% \quad (17)$$

$$ARE = ((AV - BKS)/BKS)) \times 100\%. \quad (18)$$

In terms of *BRE* and *ARE*, we give violin plots for HBMO and DPLS in Fig. 7. Note that these violin plots correspond only to the instances with $M = 6$ and 8. This is because, for all the instances with $M = 4$, DPLS achieves a more competitive performance than the other compared algorithms. From Fig. 7, one observes that the statistic distributions of DPLS are better than HBMO. Apparently, DPLS has better median lines and more reasonable dispersions than those of HBMO, indicating that DPLS can obtain more robust solutions than HBMO.

To demonstrate the stability of DPLS, for each combination of N, M, and r_{com}, we consider the average value of *SD* (i.e., SD_{avg}) of each algorithm with regard to ETC = 0, 1, 2, and 3.

[7]See more details about the BKS of TAP instances in Section IV-D.

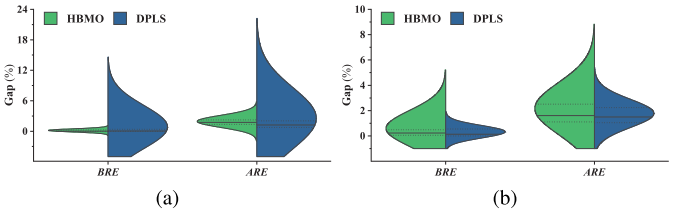


Fig. 7. Violin plots for solutions of HBMO and DPLS. (a) $M = 6$. (b) $M = 8$.

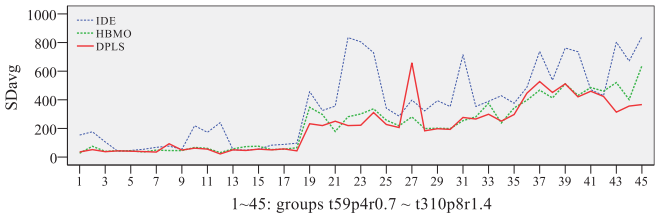


Fig. 8. Trends of SD_{avg} for IDE, HBMO, and DPLS (45 groups).

Thus, there are a total of 45 groups of SD_{avg} values (i.e., t59p4r0.7 to t310p8r1.4) for each algorithm. We here only consider IDE, HBMO, and DPLS because DPLS obviously outperforms NGHS associated with *BV*, *AV*, and *SD* (see Fig. 6). The trend of SD_{avg} is given in Fig. 8. As shown in Fig. 8, the DPLS obtains smaller SD_{avg} values than HBMO (37 out of 45 groups) and IDE (43 out of 45 groups). These results verify the competitive stability of DPLS for solving TAP.

To observe the statistic distributions of the solutions of DPLS and the three compared algorithms, we draw the box plots for instances t66p61.0e1, t162p61.0e1, t245p61.0e1, and t310p61.0e1 in Fig. 9. Meanwhile, we give the related 95% CIs of these algorithms in Fig. 10. It can be clearly found from Figs. 9 and 10 that DPLS has a more reasonable distribution than the other compared algorithms, regarding t66p61.0e1, t245p61.0e1, and t310p61.0e1. However, the median line of DPLS is similar to that of HBMO with respect to t162p61.0e1. Thereby, the Q–Q plots that reveal the detailed information about the solution distributions of HBMO and DPLS are given in Fig. 11. As shown in Fig. 11, DPLS achieves a good match between the observed values (objective functions) and the relevant expected values, which is better than HBMO. In view of the abovementioned observations, DPLS can obtain better statistic distributions of solutions than the compared algorithms.

Moreover, we give the trends of T_{avg}(s) for each instance in Fig. 12. We can see from Fig. 12 that the values of T_{avg}(s) increase along with the enlargement of the problem size. However, the values of T_{avg}(s) ranged from 5 to 250 s for instances with $N = 59$, 66, and 162 while 200 to 1300 s for instances with $N = 245$ and 310. Thus, the values of T_{avg}(s) are acceptable and, in turn, reveal the efficiency of DPLS. Due to the effectiveness and efficiency, it would be highly desirable to use DPLS for addressing relevant real-life applications that can be modeled as TAP.

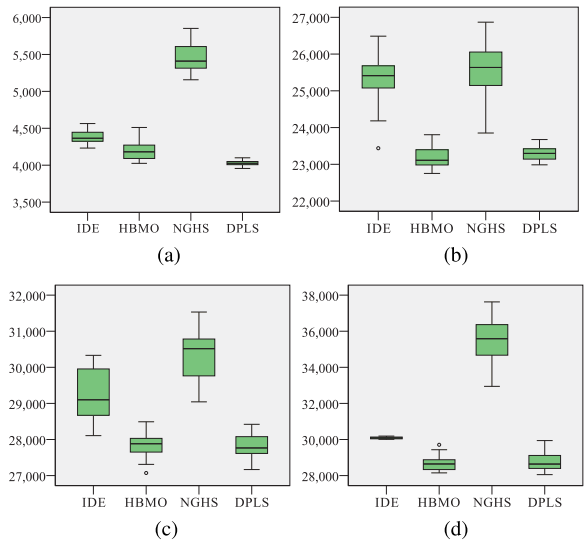


Fig. 9. Box plots for IDE, HBMO, NGHS, and DPLS. (a) t66p61.0e1. (b) t162p61.0e1. (c) t245p61.0e1. (d) t310p61.0e1.

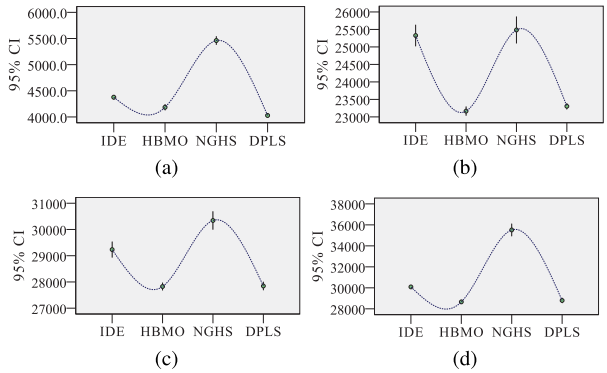


Fig. 10. 95% CI's for IDE, HBMO, NGHS, and DPLS. (a) t66p61.0e1. (b) t162p61.0e1. (c) t245p61.0e1. (d) t310p61.0e1.

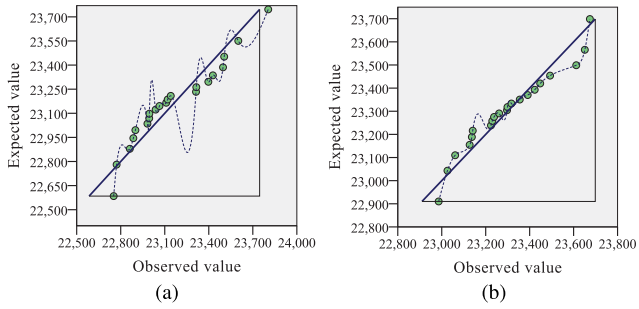


Fig. 11. Q–Q plots for (a) HBMO and (b) DPLS (t162p61.0e1).

D. Additional Assessments of DPLS

We provide additional assessments of DPLS concerning the following issues: 1) the number of BKS obtained by DPLS; 2) effectiveness of DPLS for difficult instances; and 3) the effectiveness of DPLS's components. Thereby, we can verify the overall performance of DPLS and the contributions of the main components of DPLS.

1) Number of BKS Obtained by DPLS: Since no previous results are available for the 180 benchmark instances, we report the values of BKS as a useful supplement. For each instance, we solve the mathematical model of TAP (see Section I) using the CPLEX solver. BKS for each instance is reported as the optimal solution if it is solved by CPLEX. Otherwise, the values of BKS are reported as the best objective functions found by IDE, HBMO, NGHS, or DPLS. The stopping condition of CPLEX is the determined running time of 2.5 h.[8] Among all 180 benchmark instances, the numbers of BKS and optimal solutions (denoted as OPT) that can be found by IDE, HBMO, NGHS, and DPLS are given in Fig. 13. One can clearly see that DPLS finds 97 instances among all 180 instances that can be reported as BKS, wherein 80 of them are optimal solutions, which are much better than IDE, HBMO, and NGHS.

Moreover, we observe that CPLEX can obtain 134 optimal solutions out of 180 instances. Note that, for the other 46 large-sized instances, CPLEX stops with the error message "out of memory." This fact shows that CPLEX is limited to small or medium instances, and for large instances, metaheuristics, such as DPLS, are a useful alternative. For the 134 instances with known optima, DPLS attains 80 optimal

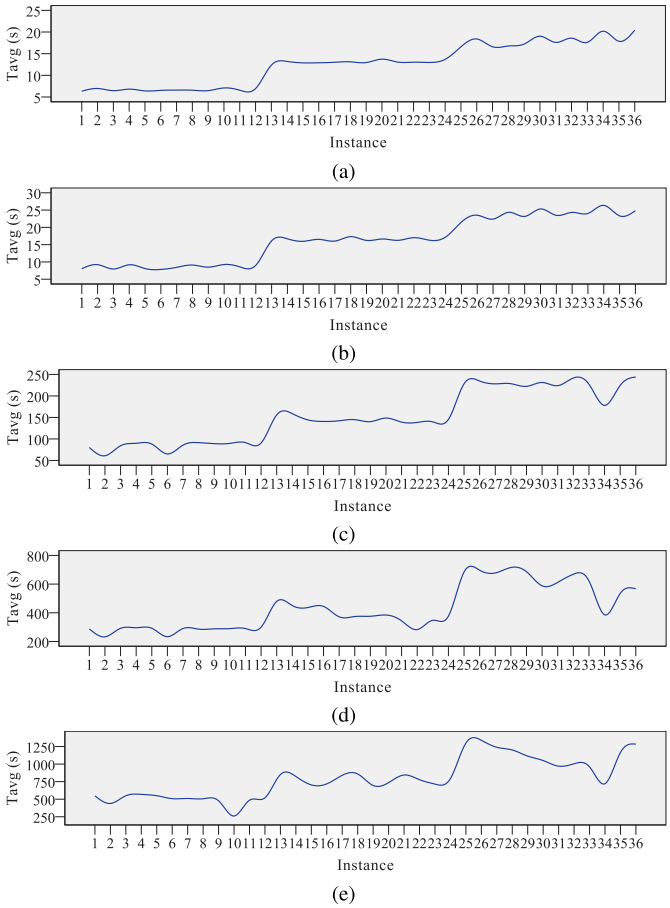


Fig. 12. Trends of T_{avg}(s) for each instance. (a) 1−36: t59p4r0.7e0−t59p8r1.4e3. (b) 1−36: t66p4r0.7e0−t66p8r1.4e3. (c) 1−36: t162p4r0.7e0−t162p8r1.4e3. (d) 1−36: t245p4r0.7e0−t245p8r1.4e3. (e) 1−36: t310p4r0.7e0−t310p8r1.4e3.

[8]The values of BKS for the 180 benchmark instances are available at http://dx.doi.org/10.13140/RG.2.2.29371.23847

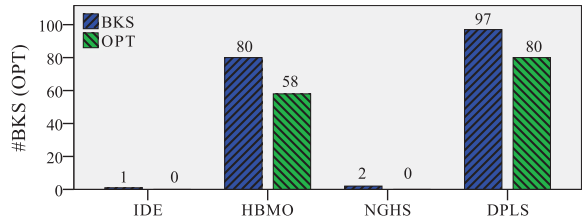


Fig. 13. Numbers of BKS and OPT for IDE, HBMO, NGHS, and DPLS.

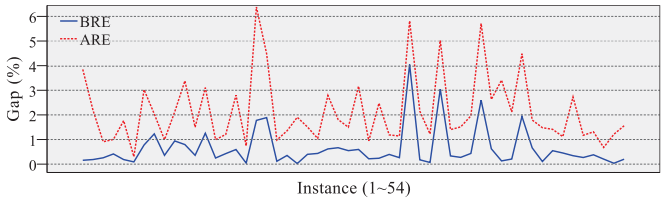


Fig. 14. Trends of *BRE* and *ARE* of DPLS for 54 instances.

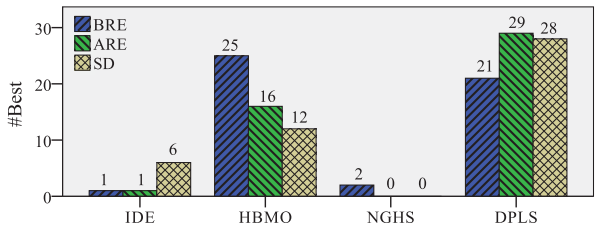


Fig. 15. Summary of comparison results for difficult instances.

TABLE IV
SIGNIFICANCE OF DIFFERENCE FOR DIFFICULT INSTANCES

Algorithm pair	*p*-values of two indicators			
	BRE	Significance	*ARE*	Significance
DPLS vs. IDE	0.000	Yes	0.000	Yes
DPLS vs. NGHS	0.000	Yes	0.000	Yes
DPLS vs. HBMO	0.763	No	0.039	Yes

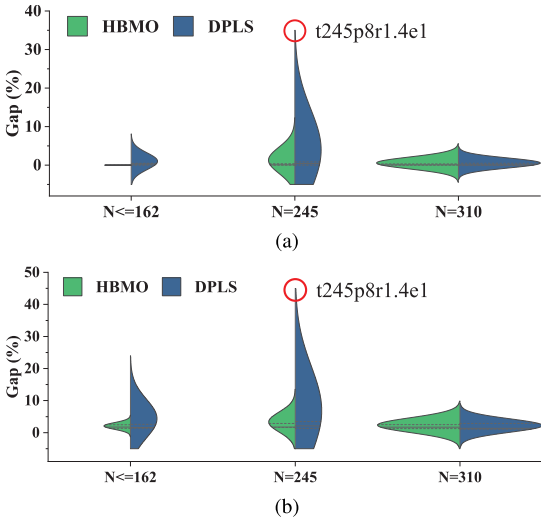


Fig. 16. Violin plots for solutions of HBMO and DPLS (difficult instances). (a) *BRE*. (b) *ARE*.

solutions, with computation times T_{avg}(s) that are significantly shorter than CPLEX (see Fig. 12). Furthermore, for the other 54 instances[9] for which DPLS fails to attain optimal solutions, we calculate *BRE* and *ARE*. The trends of the values of *BRE* and *ARE* of DPLS are given in Fig. 14. It is clear that the gaps to the optima for most of these 54 instances are relatively small: less than 1% for *BRE* and less than 3% for *ARE*. On the basis of the abovementioned analysis, DPLS has a large potential to find high-quality solutions for TAP.

2) Effectiveness of DPLS for Difficult Instances: As mentioned earlier, CPLEX fails to solve the 46 large-sized instances, which can be considered to be the most difficult instances. We now conduct comparisons of IDE, NGHS, HBMO, and DPLS on these 46 instances, so as to verify the effectiveness of DPLS for solving difficult instances compared with the reference algorithms.[10] In terms of *BRE*, *ARE*, and *SD*, we summarize the values of *#Best* obtained by DPLS and the three compared algorithms in Fig. 15. Moreover, the significance of the difference between DPLS and the compared algorithms is given in Table IV.

As shown in Fig. 15 and Table IV, DPLS performs better than the reference algorithms with regard to *ARE* and *SD*. The values of *#Best* of DPLS for *ARE* and *SD* are 29 and 28 that are larger than those of the compared algorithms. For *BRE*, HBMO has a slightly better value of *#Best*. However, the related *p*-value is 0.763 (> 0.05), which means that there is no significant difference between HBMO and DPLS regarding *BRE*. Thereby, we conduct further discussions on the performance of HBMO and DPLS for difficult instances. In particular, 46 instances are divided into three groups, including "$N \leq 162$," "$N = 245$," and "$N = 310$." In terms of *BRE* and *ARE*, the violin plots for these groups of solutions of HBMO and DPLS are given in Fig. 16.

From Fig. 16, we find that DPLS obtains better values of *BRE* and *ARE* than HBMO, concerned with the groups "$N \leq 162$" and "$N = 310$." However, HBMO has a slightly better performance than DPLS for the group "$N = 245$." Such worse values of *BRE* and *ARE* for DPLS are mainly due to the extreme points of the instance t245p8r1.4e1. Moreover, we give the violin plots for the solutions of HBMO and DPLS considering large-sized instances with $N = 310$ and $M = 8$ in Fig. 17 and so confirm the competitiveness of the proposed DPLS. Furthermore, the *BRE* values of DPLS for these large-sized instances are given in Fig. 18. It can be seen from Fig. 18 that the *BRE* values of DPLS are less than 0.5% for almost all these large-sized instances. The abovementioned results prove the effectiveness of DPLS for difficult instances of TAP. Thus, DPLS can be a potential method to address real-life applications in complex conditions.

3) Effectiveness of DPLS's Components: The dual probability learning and GNS are the two main components of DPLS. To assess their effectiveness, we introduce three variants of DPLS as follows. 1) DPLS_V1 is a DPLS variant where

[9]See the list of these 54 instances at http://dx.doi.org/10.13140/RG.2.2.28532.37763

[10]Detailed comparisons of IDE, NGHS, HBMO, and DPLS for 46 difficult instances are available at http://dx.doi.org/10.13140/RG.2.2.32569.39522

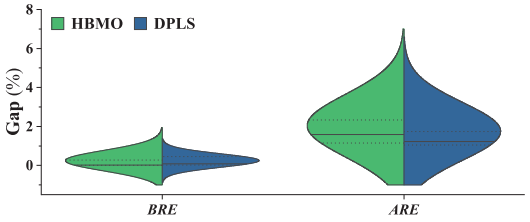


Fig. 17. Violin plots for solutions of HBMO and DPLS ($N = 310$ and $M = 8$).

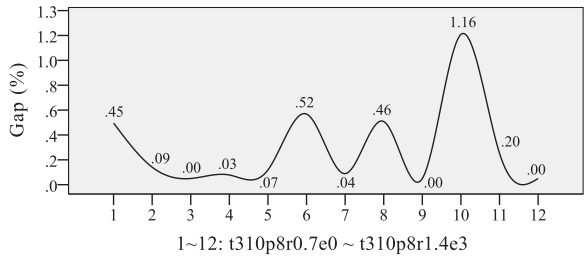


Fig. 18. *BRE* values of DPLS for large-sized instances ($N = 310$ and $M = 8$).

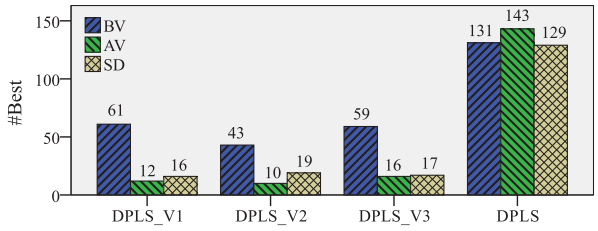


Fig. 19. Summary of results of three variants and DPLS (180 instances).

TABLE V
SIGNIFICANCE OF DIFFERENCE BETWEEN DPLS AND THREE VARIANTS

Algorithm pair	*p*-values of two indicators			
	BV	Significance	*AV*	Significance
DPLS vs. DPLS_V1	0.000	Yes	0.000	Yes
DPLS vs. DPLS_V2	0.000	Yes	0.000	Yes
DPLS vs. DPLS_V3	0.000	Yes	0.000	Yes

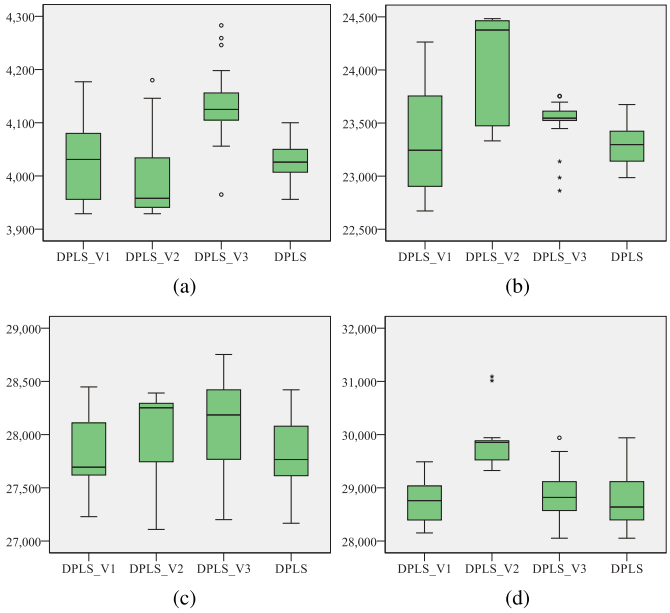


Fig. 20. Box plots for DPLS_V1, DPLS_V2, DPLS_V3, and DPLS (a) t66p61.0e1. (b) t162p61.0e1. (c) t245p61.0e1. (d) t310p61.0e1.

we remove the global-level learning and the dual probability learning only contains the local-level learning to guide GNS; 2) DPLS_V2 is a DPLS variant where the global-level learning is used only and the local-level learning is disabled; and 3) DPLS_V3 is a DPLS variant where the Pareto-dominance method in Algorithm 5 is replaced by the linear sum of two gains in GNS. Note that DPLS_V1 can also be regarded as a variant of the solution methods of [23] and [24], which are concerned only with the local-level learning and GNS of DPLS. Hence, comparing DPLS with DPLS_V1 and DPLS_V2, the effectiveness of the interactions between the two learning levels of DPLS can be verified. In addition, we perform the comparison of DPLS_V3 and DPLS to study the competitiveness of the proposed Pareto-dominance GNS and the existing linear-sum-based approach [2]. For this comparison, DPLS_V1, DPLS_V2, and DPLS_V3 adopt the same CPU time as DPLS. We summarize the comparison results of the three variants and DPLS in Fig. 19.[11] Moreover, the significances of the differences between DPLS and the three variants are given in Table V.

As shown in Fig. 19, for the values of *#Best*, DPLS reports the values of 131, 143, and 129 out of 180 instances regarding *BV*, *AV*, and *SD* that are much better than those of the three compared variants. Meanwhile, we observe from Table V that the *p*-values for all the three algorithm pairs are all less than 0.05. The abovementioned results prove that the proposed interaction mechanisms between the two learning levels of DPLS and the Pareto-dominance GNS contribute significantly to the performance of DPLS.

Moreover, we draw the box plots for DPLS_V1, DPLS_V2, DPLS_V3, and DPLS for four instances in Fig. 20, showing that the statistical distributions of the solutions of DPLS are reasonable. In view of the abovementioned analysis, we conclude that the proposed dual probability learning and Pareto-dominance GNS are useful to enhance the performance of DPLS.

V. REAL-LIFE CASE STUDY

In this section, we apply the proposed DPLS to solve the real-life OSAP for the production of silicon steel mentioned in Section I. The sketch of OSAP is illustrated in Fig. 21. Note that we here refer to the term "open-order slabs" as "slabs" for short. For OSAP, we first identify a number of original slabs with a relatively large weight according to current customer orders and production statuses. Then, these original slabs are virtually cut into a set of slabs (with small weight) to be assigned. Such a virtual cutting operator guarantees that some orders with very small weight can still have available slabs to be assigned. Next, the cutting cost (communication cost) of

[11] Detailed comparisons of DPLS and the three variants for the 180 instances are available at http://dx.doi.org/10.13140/RG.2.2.34666.54723

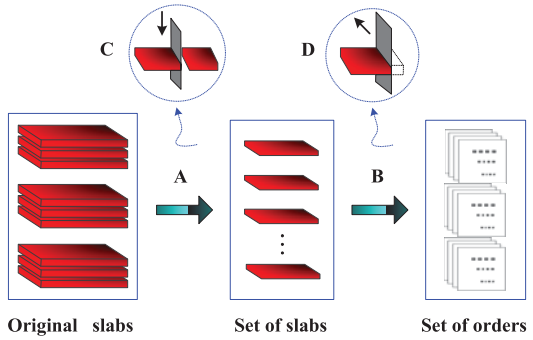


Fig. 21. Sketch of OSAP for the production of silicon steel.

TABLE VI
COMPARISON RESULTS OF HBMO AND DPLS (CASE STUDY)

Algorithm	*BV*	*AV*	*SD*	T_{avg}(s)
HBMO	5094290	5094295.21	6.44	195.68
DPLS	5094286	5094286.00	0.00	195.13

two slabs from the same original slabs is calculated, which is determined by both the width and the thickness of slabs. To satisfy the width requirements of orders, we introduce another cutting cost (assignment cost) resulted from the loss of materials. Accordingly, we can obtain the factory data of OSAP.[12] Afterward, slabs are assigned to customer orders subject to the two considered capacities (see Section I).

In this case study, we compare DPLS with HBMO that performs relatively better than IDE and NGHS for the 180 benchmark instances. We independently run DPLS 21 times with $tMax = 5000$ and calculate the related T_{avg}(s). Then, we run HBMO 21 times and, for each run, check the CPU time at each iteration. HBMO is terminated if the current CPU time reaches T_{avg}(s). The comparison results of HBMO and DPLS are reported in Table VI.

As shown in Table VI, DPLS outperforms HBMO with respect to *BV*, *AV*, and *SD*. Meanwhile, the value of T_{avg}(s) of DPLS is less than 200 s, which is quite acceptable for management practices. Moreover, for the best solution of DPLS, we give the load (ton) and utility (%) of each order regarding memory and processing requirements that have been assigned in Table VII. It shows that the best value of the memory utility is 96.18%, and most of the memory utility values (seven out of ten orders) are larger than 70%. As for the values of the processing utility, they are all smaller than the capacity of each order (the best value is 0.0%). Based on our previous survey, the abovementioned results can highly satisfy the real-life production conditions, as well as customer satisfaction.

VI. CONCLUSION AND FUTURE WORK

We proposed a DPLS to solve the challenging TAP. To the best of our knowledge, this is the first local search algorithm based on probability learning for TAP. The dual probability learning component of DPLS combines global-level learning and local-level learning to identify promising search regions that are examined by the gain-based neighborhood search component. Due to the tight interactions of these two original and complementary components, DPLS achieves a suitable balance between intensification and diversification of the given search space.

To assess the performance of the proposed algorithm, we introduced a set of 180 benchmark instances with different features and reported computational results for them with our algorithm in comparison with three reference algorithms and the general CPLEX solver. These results can serve as references for the performance assessment of new TAP algorithms. Moreover, a real-life case study from an iron and steel company is introduced to show the practical values of our DPLS.

The idea of the proposed DPLS that adopts a dual probability learning mechanism is of general interest. It would be interesting to apply the same idea to solve other TAPs, such as equilibrium TAP (ETAP), reliability-oriented TAP (RTAP), and multiobjective TAP (MOTAP).

[12]The applied factory data of OSAP in this case study are available at http://dx.doi.org/10.13140/RG.2.2.36553.98403

TABLE VII
LOAD AND UTILITY OF CUSTOMER ORDERS (DPLS)

Load (ton)	Customer order									
	1	2	3	4	5	6	7	8	9	10
Memory	67	67	76	368	142	342	132	245	163	208
Processing	18	0	26	186	0	127	132	160	163	71
Utility (%)	**Customer order**									
	1	2	3	4	5	6	7	8	9	10
Memory	53.90	74.52	88.51	94.74	92.09	96.18	41.95	70.67	49.68	95.26
Processing	28.96	0.00	60.56	95.77	0.00	71.43	83.90	92.30	99.37	65.03

ACKNOWLEDGMENT

The authors are grateful to the reviewers for the useful comments that helped them to significantly improve this article.

REFERENCES

[1] F. Glover, J. Hultz, and D. Klingman, "Improved computer-based planning Techniques. Part II," *Interfaces*, vol. 9, no. 4, pp. 12–20, Aug. 1979.
[2] Q. Kang and H. He, "Honeybee mating optimization algorithm for task assignment in heterogeneous computing systems," *Intell. Autom. Soft Comput.*, vol. 19, no. 1, pp. 69–84, Feb. 2013.
[3] D. Zou, L. Gao, S. Li, J. Wu, and X. Wang, "A novel global harmony search algorithm for task assignment problem," *J. Syst. Softw.*, vol. 83, no. 10, pp. 1678–1688, Oct. 2010.
[4] D. Zou, H. Liu, L. Gao, and S. Li, "An improved differential evolution algorithm for the task assignment problem," *Eng. Appl. Artif. Intell.*, vol. 24, no. 4, pp. 616–624, Jun. 2011.
[5] Y. Hamam and K. S. Hindi, "Assignment of program modules to processors: A simulated annealing approach," *Eur. J. Oper. Res.*, vol. 122, no. 2, pp. 509–513, Apr. 2000.
[6] W. Wang and Y. Jiang, "Community-aware task allocation for social networked multiagent systems," *IEEE Trans. Cybern.*, vol. 44, no. 9, pp. 1529–1543, Sep. 2014.
[7] D.-H. Lee, S. A. Zaheer, and J.-H. Kim, "A resource-oriented, decentralized auction algorithm for multirobot task allocation," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 4, pp. 1469–1481, Oct. 2015.
[8] O. Karasakal, L. Kandiller, and N. E. Özdemirel, "A branch and bound algorithm for sector allocation of a naval task group," *Nav. Res. Logistics*, vol. 58, no. 7, pp. 655–669, 2011.

[9] A. Ernst, H. Jiang, and M. Krishnamoorthy, "Exact solutions to task allocation problems," *Manage. Sci.*, vol. 52, no. 10, pp. 1634–1646, Oct. 2006.
[10] G. S. Rao, H. S. Stone, and T. C. Hu, "Assignment of tasks in a distributed processor system with limited memory," *IEEE Trans. Comput.*, vol. C-28, no. 4, pp. 291–299, Apr. 1979.
[11] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Comput. Surveys*, vol. 35, no. 3, pp. 268–308, Sep. 2003.
[12] L. Tang, X. Wang, and Z. Dong, "Adaptive multiobjective differential evolution with reference axis vicinity mechanism," *IEEE Trans. Cybern.*, vol. 49, no. 9, pp. 3571–3585, Sep. 2019.
[13] L. Tang, Y. Dong, and J. Liu, "Differential evolution with an individual-dependent mechanism," *IEEE Trans. Evol. Comput.*, vol. 19, no. 4, pp. 560–574, Aug. 2015.
[14] P. J. Angeline, "Evolutionary algorithms and emergent intelligence," Ph.D. dissertation, Dept. Comput. Inf. Sci., Ohio State Univ., Columbus, OH, USA, 1993.
[15] L. Tang, Y. Zhao, and J. Liu, "An improved differential evolution algorithm for practical dynamic scheduling in steelmaking-continuous casting production," *IEEE Trans. Evol. Comput.*, vol. 18, no. 2, pp. 209–225, Apr. 2014.
[16] H. H. Hoos and T. Stützle, *Stochastic Local Search: Foundations and Applications*. Amsterdam, The Netherlands: Elsevier, 2004.
[17] U. Benlic and J.-K. Hao, "Breakout local search for the quadratic assignment problem," *Appl. Math. Comput.*, vol. 219, no. 9, pp. 4800–4815, Jan. 2013.
[18] T. Stützle, "Iterated local search for the quadratic assignment problem," *Eur. J. Oper. Res.*, vol. 174, no. 3, pp. 1519–1539, Nov. 2006.
[19] S. Mitrović-Minić and A. P. Punnen, "Local search intensified: Very large-scale variable neighborhood search for the multi-resource generalized assignment problem," *Discrete Optim.*, vol. 6, no. 4, pp. 370–377, Nov. 2009.
[20] L. Tang and Y. Meng, "Data analytics and optimization for smart industry," *Frontiers Eng. Manage.*, early access, Aug. 26, 2020, doi: 10.1007/s42524-020-0126-0.
[21] I. H. Osman and G. Laporte, "Metaheuristics: A bibliography," *Ann. Oper. Res.*, vol. 63, pp. 513–628, Oct. 1996.
[22] M. Pelikan, D. E. Goldberg, and F. G. Lobo, "A survey of optimization by building and using probabilistic models," *Comput. Optim. Appl.*, vol. 21, no. 1, pp. 5–20, 2002.
[23] Y. Zhou, J.-K. Hao, and B. Duval, "Reinforcement learning based local search for grouping problems: A case study on graph coloring," *Expert Syst. Appl.*, vol. 64, pp. 412–422, Dec. 2016.
[24] Y. Zhou, B. Duval, and J.-K. Hao, "Improving probability learning based local search for graph coloring," *Appl. Soft Comput.*, vol. 65, pp. 542–553, Apr. 2018.
[25] E. Falkenauer, "A hybrid grouping genetic algorithm for bin packing," *J. Heuristics*, vol. 2, no. 1, pp. 5–30, 1996.
[26] D. C. Lee, "Some compartmentalized secure task assignment models for distributed systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 12, pp. 1414–1424, Dec. 2006.
[27] H.-L. Li, Y.-H. Huang, and S.-C. Fang, "A logarithmic method for reducing binary variables and inequality constraints in solving task assignment problems," *INFORMS J. Comput.*, vol. 25, no. 4, pp. 643–653, Nov. 2013.
[28] V. M. Lo, "Heuristic algorithms for task assignment in distributed systems," *IEEE Trans. Comput.*, vol. C-37, no. 11, pp. 1384–1397, Nov. 1988.
[29] A. Lusa and C. N. Potts, "A variable neighbourhood search algorithm for the constrained task allocation problem," *J. Oper. Res. Soc.*, vol. 59, no. 6, pp. 812–822, Jun. 2008.
[30] K. Kaya and B. Uçar, "Exact algorithms for a task assignment problem," *Parallel Process. Lett.*, vol. 19, no. 3, pp. 451–465, Sep. 2009.
[31] Y. Jiang, "A survey of task allocation and load balancing in distributed systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 585–599, Feb. 2016.
[32] P. Emberson and I. Bate, "Stressing search with scenarios for flexible solutions to real-time task allocation problems," *IEEE Trans. Softw. Eng.*, vol. 36, no. 5, pp. 704–718, Sep. 2010.
[33] G. Attiya and Y. Hamam, "Task allocation for maximizing reliability of distributed systems: A simulated annealing approach," *J. Parallel Distrib. Comput.*, vol. 66, no. 10, pp. 1259–1266, Oct. 2006.
[34] H. Sayyaadi and M. Moarref, "A distributed algorithm for proportional task allocation in networks of mobile agents," *IEEE Trans. Autom. Control*, vol. 56, no. 2, pp. 405–410, Feb. 2011.
[35] Y. Baram, R. El-Yaniv, and K. Luz, "Online choice of active learning algorithms," *J. Mach. Learn. Res.*, vol. 5, pp. 255–291, Mar. 2004.

Zuocheng Li is currently pursuing the Ph.D. degree with the Key Laboratory of Data Analytics and Optimization for Smart Industry (Northeastern University), Ministry of Education, Shenyang, China.

He has participated in two real-life application projects with cooperative enterprises about optimization algorithms in decision supporting systems. His research interests include modeling and solution methods for combinatorial optimization problems, machine learning, and their applications.

Lixin Tang (Senior Member, IEEE) received the B.Eng. degree in industrial automation, the M.Eng. degree in systems engineering, and the Ph.D. degree in control theory and application from Northeastern University, Shenyang, China, in 1988, 1991, and 1996, respectively.

He is currently a fellow of the Chinese Academy of Engineering, the Director of the Key Laboratory of Data Analytics and Optimization for Smart Industry, Ministry of Education, and the Head of the Centre for Artificial Intelligence and Data Science, Northeastern University. His research articles have appeared in academic journals, such as *Operations Research*, *INFORMS Journal on Computing*, *IIE Transactions*, IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, IEEE TRANSACTIONS ON POWER SYSTEMS, and IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY. His research interests cover industrial big data science, data analytics and machine learning, reinforcement learning and dynamic optimization, computational intelligent optimization, plant-wide production and logistics planning, production and logistics batching and scheduling, and engineering applications in manufacturing (steel, petroleum–chemical, and nonferrous), energy, resources industry, and logistics systems.

Dr. Tang serves as an Associate Editor for *IISE Transactions*, IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, IEEE TRANSACTIONS ON CYBERNETICS, *Journal of Scheduling*, *International Journal of Production Research*, and *Journal of the Operational Research Society* and an Area Editor for the *Asia-Pacific Journal of Operational Research*. He also serves on the Editorial Board of *Annals of Operations Research*.

Jin-Kao Hao received the B.S. degree in computer science from the National University of Defense Technology, Changsha, China, in 1982, the M.S. degree in computer science from the National Institute of Applied Sciences, Lyon, France, in 1987, the Ph.D. degree in constraint programming from the University of Franche-Comté, Besançon, France, in 1991, and the Professorship Diploma [Habilitation Diriger des Recherches (HDR)] from the University of Science and Technology of Montpellier, Montpellier, France, in 1998.

He has been a Full Professor with the Computer Science Department, University of Angers, Angers, France, since 1999. He has been a Senior Fellow of the Institut Universitaire de France, Paris, France, since 2015. He has authored or coauthored more than 250 peer-reviewed publications and coedited nine books in Springers LNCS Series. His research lies in the design of effective algorithms and intelligent computational methods for solving large-scale combinatorial search problems. He is also interested in various application areas, including data science, complex networks, and transportation.

Dr. Hao has served as an Invited Member of more than 200 Program Committees of international conferences. He is also an Associate Editor and Editorial Board member of eight international journals.