Expert Systems With Applications 97 (2018) 193–204

Contents lists available at ScienceDirect

Expert Systems With Applications

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

Multi-mode resource leveling in projects with mode-dependent generalized precedence relations

Hongbo Li, Xuebing Dong*

School of Management, Shanghai University, Shangda Road 99, Shanghai 200444, China

A R T I C L E I N F O

Article history:
Received 7 September 2017
Revised 15 December 2017
Accepted 16 December 2017
Available online 16 December 2017

Keywords:
Project management
Project scheduling
Resource leveling
Generalized precedence relations
Multi-mode case
Estimation of distribution algorithm

A B S T R A C T

In real-life project management, resource leveling is an important technique to ensure the effective use of resources, in which activities (a) can often be executed in alternative modes and (b) are constrained by precedence relations with minimum and maximum time lags that can be modeled using generalized precedence relations (GPRs). In addition, the values of the time lags tend to depend on activity modes. The resource leveling problem with multiple modes and mode-dependent GPRs (MRLP-GPR) is a generalization of the classic NP-hard resource leveling problem. To our knowledge, no literature exists regarding the MRLP-GPR.

We propose several heuristics for the MRLP-GPR built upon two solution approaches: (a) a steepest descent algorithm and a fast descent algorithm that are based on a decomposition approach and (b) a hybrid estimation of distribution algorithm (EDA), which is based on an integration approach. Extensive computational experiments on a large number of benchmark instances are conducted to evaluate the proposed heuristics. A comparison of the results shows that our EDA outperforms or is competitive with three baseline heuristics (a random search algorithm and two variants of a genetic algorithm that is the best-performing metaheuristic for the single-mode resource leveling problem with GPRs). Our results can serve as a benchmark for future research. Our model and solution algorithms provide an automatic tool for the project manager's multi-mode resource leveling decision-making.

1. Introduction

For projects in practice, resource leveling is an important technique to ensure the effective use of resources. To a large extent, many negative effects can be avoided by resource leveling, such as hastily allocating temporary resources, frequently hiring and dismissing employees, and idle resources caused by reserving a high security level (Li, Xiong, Liu, & Li, 2017). This condition motivates a study of the resource leveling problem (RLP) that aims to minimize the resource utilization fluctuations through scheduling activities under precedence constraints and the project deadline constraint (Li & Demeulemeester, 2016; Li, Xu, & Demeulemeester, 2015; Markou, Koulinas, & Vavatsikos, 2017; Neumann, Schwindt, & Zimmermann, 2003). Here, we propose three heuristics for the RLP with multiple modes and mode-dependent generalized precedence relations (GPRs).

In real-life project management, activities can often be executed in alternative modes, indicating that activity duration can be changed by varying the amount of resources allocated (Heilmann, 2001). However, few studies have considered multiple modes in the RLP. Coughlan, Lübbecke, and Schulz (2015) developed a branch-price-and-cut algorithm for the multi-mode RLP (MRLP) with the objective of minimizing the resource availability cost. However, their problem setting is relatively simplistic; for example, they assume that multiple resource types exist but that each activity requires only one resource type to execute. Behrouz, Hojjat, and Esmaeil (2012) designed a branch-and-bound procedure for the MRLP. Based on constraint programming, Menesi and Hegazy (2015) proposed a bi-objective multi-mode project scheduling model in which one of the objectives is to minimize the peak resource demand. Guo, Li, Zhang, and Ye (2012) presented a particle swarm optimization algorithm for a multi-objective multi-mode project scheduling problem. One of their objective functions was to minimize the resource usage variation.

The above-mentioned studies on the MRLP focus only on critical path method (CPM)-type precedence relations, i.e., the finish-start precedence relations with zero time lag. To our knowledge, a more generalized precedence relation (also known as minimum and maximum time lags) has not been considered in the MRLP. The advantage of the GPR is that it can easily describe many com-

* Corresponding author.
E-mail address: dongxuebing116@sina.com (X. Dong).

https://doi.org/10.1016/j.eswa.2017.12.030
0957-4174/© 2017 Published by Elsevier Ltd.

plex precedence constraints, such as the project/activity due date, parallel execution of activities, time window constraints, and so on (Dorndorf, Pesch, & Phan-Huy, 2000).

Currently, the resource leveling literature that considers GPR focuses only on solving the single-mode RLP-GPR. For exact methods, a branch-and-bound algorithm (Gather, Zimmermann, & Bartels, 2011; Neumann & Zimmermann, 2000) and a mixed-integer programming algorithm (Kreter, Rieck, & Zimmermann, 2014; Rieck & Zimmermann, 2015; Rieck, Zimmermann, & Gather, 2012) have been proposed to handle the RLP-GPR. For heuristic approaches, Neumann and Zimmermann (1999, 2000) have devised polynomial heuristics and priority-rule-based heuristics for the RLP-GPR. Metaheuristics based on a tabu search algorithm (Neumann & Zimmermann, 2000), an iterated greedy algorithm (Ballestín, Schwindt, & Zimmermann, 2007), and a genetic algorithm (Li et al., 2017) have also been developed.

When multiple modes and GPRs come into play, the time lags tend to become mode-dependent (De Reyck & Herroelen, 1999). Mode-dependent time lags mean that the time lag between any two activities depends on the modes (durations) of both activities. For example, let us consider two activities in a software project: “testing module A” (activity 1) and its successor “testing module B” (activity 2). Assume that there are two modes for activity 1: module A can be tested by two software engineers with one hour (mode 1) or by one software engineer with two hours (mode 2). Activity 2 has one mode. Since activity 2 is the successor of activity 1, it can be started as soon as activity 1 is finished. Therefore, if activity 1 is executed in mode 1, activity 2 can be started one hour after the start of activity 1, implying that the time lag between the start of activities 1 and 2 is one hour. Similarly, if activity 1 is executed in mode 2, then the time lag between the start of activities 1 and 2 is two hours. Mode-dependent time lags have been considered in some studies on the multi-mode resource-constrained project scheduling problem with generalized precedence relations (MRCPSP-GPR), and exact and heuristic methods have been developed (Ballestín, Barrios, & Valls, 2013; Barrios, Ballestín, & Valls, 2011; Heilmann, 2001, 2003; Jędrzejowicz & Ratajczak-Ropel, 2011).

However, to our knowledge, no literature exists on the RLP with multiple modes and mode-dependent generalized precedence relations (MRLP-GPR). MRLP-GPR is a generalization of RLPs that have been proven to be NP-hard (Neumann et al., 2003). Specifically, MRLP-GPR is a generalization of the MRLP in which only CPM-type precedence relations are considered. In addition, MRLP-GPR is a generalization of the RLP-GPR in which each activity has only one mode.

In this paper, the MRLP-GPR is studied for the first time and efficient and effective heuristics are developed. The following novel features and design ensure the efficiency and effectiveness of the proposed heuristics:

(a) By analyzing the character of the MRLP-GPR, we propose two solution approaches: decomposition and integration. These approaches reflect how we deal with two sub-problems (a mode assignment problem and the RLP-GPR) of the MRLP-GPR.
(b) Based on the decomposition approach, we devise a steepest descent algorithm and a fast descent algorithm. Both of the descent algorithms are equipped with a relatively fast and simple priority-based heuristic. This heuristic has a computational complexity of $O(n^2)$ and has the potential to find optimal solutions. This heuristic can also be adapted for the so-called multi-mode resource leveling schedule generation scheme.
(c) Following the integration approach, we develop a hybrid estimation of distribution algorithm (EDA) that is mixed with genetic algorithm (GA) operators. In our EDA, new individuals are generated jointly using the problem-specific probability model of the EDA and the crossover and mutation operators of the GA. Our EDA is also enhanced by certain new characteristics, such as multi-mode schedule encoding and decoding mechanisms, a problem-specific probability model updating mechanism, and a probability-generating mechanism (PGM).

To evaluate and compare the proposed heuristics, we perform extensive computational experiments on a large number of benchmark instances. The computational results show that our EDA outperforms or is competitive with the baseline heuristics (a random search algorithm and two variants of a GA (Li et al., 2017) that is the best-performing metaheuristic for the RLP-GPR). Our results can serve as a benchmark for future research. In addition, our algorithms can be easily embedded into expert systems or project management software. When facing real-world multi-mode resource leveling problems, the project manager can choose satisfactory schedules with the help of our algorithms. This will make our algorithms more practical.

The remainder of this paper is organized as follows. We first introduce the MRLP-GPR in the next section. In Section 3, we describe the decomposition and integration solution approaches for the MRLP-GPR. Two descent algorithms based on the decomposition approach are presented in Section 4. A hybrid EDA based on the integration approach is proposed in Section 5. Section 6 presents the computational results and comparisons. The last section concludes the paper.

2. The MRLP-GPR

2.1. Problem description

The MRLP-GPR can be stated as follows. We use a weighted cyclic activity-on-node network $G = (N, A)$ to represent a project. N is the set of nodes and indicates the activities, $N = \{0, 1, \ldots, n, n+1\}$. A is the set of directed arcs and denotes the GPRs, and $A \subseteq N \times N$. The activities are numbered from 0 to $n+1$. Dummy activities 0 and $n+1$ indicate the start and the end of the project, respectively.

Allocating different amounts of resources to each activity leads to different activity durations. Thus, each non-dummy activity i can be executed in one of M_i modes. Each mode m_i ($m_i \in \{1, \ldots, M_i\}$) represents an alternative combination of activity duration and resource requirements. There are K resource types. When executed in mode m_i, each non-dummy activity i has an integer duration d_{im_i} and requires r_{im_ik} renewable resources for the resource type k per time period ($k = 1, 2, \ldots, K$). An activity cannot be preempted once it has been started.

Given an activity i, s_i represents its starting time. Thus, the starting time of the dummy end activity s_{n+1} denotes the project's finish time. The project's deadline $\bar{d}$ is predefined, i.e., $s_{n+1} \le \bar{d}$. Given the activity mode assignments and starting times, let u_{kt} denote the resource usage for the resource type k during a time period t, with $t = 1, 2, \ldots, \bar{d}$. u_{kt} can be calculated as $u_{kt} = \sum_{i \in ACT_t} r_{im_ik}$, where ACT_t is the set of activities that are being executed in the time period t, $ACT_t = \{i | s_i < t \le s_i + d_{im_i}\}$.

The activities are subject to GPRs, which refers to the minimum and maximum lag between the starting times of the activities (Neumann et al., 2003). Because start-to-completion, completion-to-start and completion-to-completion time lags can be converted into start-to-start time lags (Bartusch et al. 1988), it is sufficient to consider only the minimum and maximum lag between the starting time of the activities. For activities i and j that are executed in modes $m_i \in \{1, \ldots, M_i\}$ and $m_j \in \{1, \ldots, M_j\}$, respectively, we associate each arc $\langle i, j \rangle \in A$ with a weight $\delta_{ij}^{m_im_j}$ representing the time lag between the starting times of the activities i and j, i.e., the start-to-start time lag. Note that time lags are mode-dependent, which means that the value of the time lag between activities i and j depends on i's and j's modes m_i and m_j. $\delta_{ij}^{m_i m_j} \geq 0$ if there exists a minimum time lag between activities i and j, i.e., activity j can be started only after activity i has been executed for at least $\delta_{ij}^{m_i m_j}$ time units ($s_j - s_i \geq \delta_{ij}^{m_i m_j}$). $\delta_{ij}^{m_i m_j} < 0$ if there exists a maximum time lag between activities j and i, i.e., activity i must be started within $-\delta_{ij}^{m_i m_j}$ time units after the start of activity j ($s_i - s_j \leq -\delta_{ij}^{m_i m_j}$). Both the minimum and maximum time lags can be expressed by the inequalities

$$s_j - s_i \geq \delta_{ij}^{m_i m_j} \quad \forall \langle i, j \rangle \in A \tag{1}$$

In the project network, the length of a path is calculated by summing the weights of all the arcs belonging to this path. For activities i and j that are executed in modes m_i and m_j, respctively, the length of the longest path from i to j is indicated by $l_{ij}^{m_i m_j}$. Once the mode assignments are determined, the longest path length between any two activities can be calculated using the Floyd-Warshall algorithm, which has a time complexity of $O(n^3)$ (Lawler, 1976; Demeulemeester & Herroelen, 2002). For a more detailed discussion on the weighted cyclic activity-on-node project networks, one can refer to Neumann & Schwindt (1997), Neumann et al. (2003) and Li et al. (2017).

2.2. Model formulation

The objective of the MRLP-GPR is to construct a baseline schedule such that the variation of resource use is minimized, subject to the GPRs and the project deadline constraint. A schedule is represented by a vector $MA = (m_0, m_1, \ldots, m_{n+1})$ of mode assignment and a vector $S = (s_0, s_1, \ldots, s_{n+1})$ of starting times, where m_i denotes the execution mode of activity i ($i \in N$). The conceptual model of the MRLP-GPR can be formulated as follows:

$$\textit{minimize} \qquad \sum_{k=1}^{K} c_k \sum_{t=1}^{\bar{d}} u_{kt}^2 \tag{2}$$

subject to:

$$s_j - s_i \geq \delta_{ij}^{m_i m_j} \qquad \forall \langle i, j \rangle \in A \tag{3}$$

$$s_{n+1} \leq \bar{d} \tag{4}$$

$$\sum_{i \in ACT_t} r_{im_i k} \leq u_{kt} \qquad k = 1, \ldots, K; \; t = 1, \ldots, \bar{d} \tag{5}$$

$$m_i \in \{1, \ldots, M_i\} \qquad \forall i \in N \tag{6}$$

$$s_0 = 0 \tag{7}$$

$$s_i \geq 0 \qquad \forall i \in N \tag{8}$$

The objective function in (2) minimizes the total weighted sum of the squared resource usage. The weight c_k indicates the unit cost of resource type k. The constraints in (3) represent the GPRs. The project deadline constraint is specified by (4). The constraints in (5) are used to calculate the resource usage u_{kt}. The constraints in (6) ensure that exactly one mode is selected for each activity. The constraint in (7) forces the project to start at time 0. The constraints in (8) ensure that the starting time for each activity is non-negative.

Given the mode assignment vector MA, if there are no cycles with positive length in the network (Neumann et al., 2003), MA is called a time-feasible mode. Given the mode assignment vector MA and the temporal constraints (3) and (4), the earliest start time for activity i is equal to the longest path length from activity 0 to activity i, i.e., $es_i(MA) = l_{0i}^{m_0 m_i}$. The latest start time for activity i is equal to the difference between the project's deadline and the longest path length from activity i to activity $n+1$, i.e., $ls_i = \bar{d} - l_{i,n+1}^{m_i m_{n+1}}$. If only a part of the activity is assigned a start time, then those activities belong to the set N' of the scheduled activities. The list of start times $s_i \geq 0$ for the scheduled activities $i \in N'$ forms a partial schedule S'. Given a partial schedule $S' = (s_i)_{i \in N'}$ and the mode assignment vector MA, the earliest start time for unscheduled activity $j \in N \backslash N'$ can be calculated as $es_j(S', MA) = \max\{l_{0j}, \max_{i \in N'}(s_i + l_{ij}^{m_i m_j})\}$, where $l_{ij}^{m_i m_j} = -\infty$ if j is not reachable from i. The latest start time for activity j is $ls_j(S', MA) = \min\{\bar{d} - l_{j,n+1}^{m_j m_{n+1}}, \min_{i \in N'}(s_i - l_{ji}^{m_j m_i})\}$.

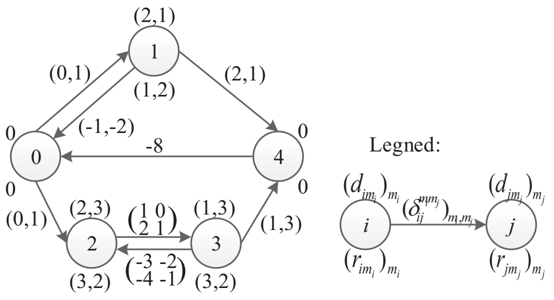

Fig. 1. Project network of the example.

Following the classification scheme of Demeulemeester and Herroelen (2002), the MRLP-GPR can be denoted as $m, 1|\bar{d}, gpr, \delta_n, disc, mu| \sum c \cdot u^2$. In the MRLP-GPR, if there is only one mode for each activity, this will lead to the single-mode RLP-GPR. Whereas the RLP-GPR is NP-hard in the strong sense (Neumann et al., 2003). Therefore, as a generalization of the RLP-GPR, the MRLP-GPR is also NP-hard.

Note that the model presented in this section is conceptual and non-linear. It is also not a standard math programming model, since the variable m_i serves as an index in the equations (3). To the best of our knowledge, there has been no research on the math programming model for the MRLP-GPR. To efficiently solve the NP-hard MRLP-GPR, we will develop several heuristics in the following sections.

In addition, our model and heuristics for multi-mode resource leveling can be potentially applied to various projects. For instance, in many software projects, the use of Minimum Viable Product (MVP) is prevalent (Moogk, 2012). MVP is a product with just sufficient features to satisfy early requirements. It is obvious that implementing a MVP can be managed as a project. When dealing with the MVP project, reducing wasted engineering hours and finishing the project as soon as possible are critical. In our model, the time concerns can be treated by GPRs. The MVP project may also be faced with limited resources availability. In our approach, the resource constraints are implicitly satisfied in constraints (5). Therefore, our heuristics can be used for resource leveling in the MVP project.

2.3. Example

To illustrate the MRLP-GPR, Fig. 1 gives an example project network. The project consists of 5 activities, with 2 dummy activities (activities 0 and 4) and 3 non-dummy activities (activities 1, 2 and 3). The activity indices are placed inside the nodes. Each non-dummy activity has two execution modes. The number(s) appearing above each node indicate(s) the corresponding activity's duration. There is one renewable resource type, and the number(s) appearing below each node indicate(s) the corresponding activ-

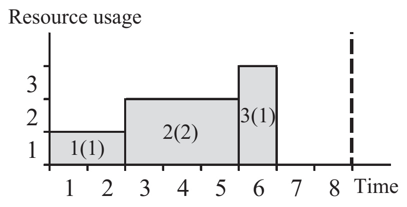

Fig. 2. Optimal schedule of the example project.

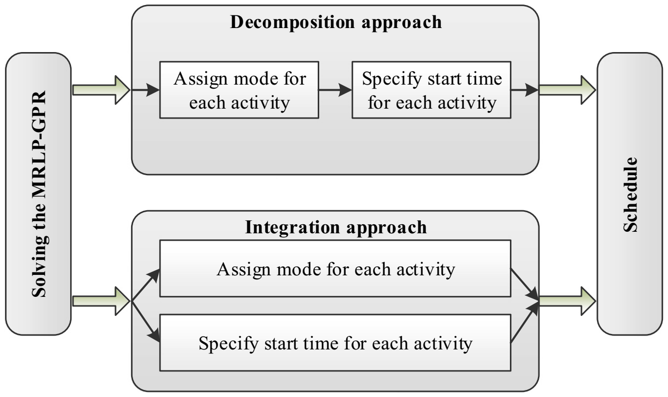
Solving the MRLP-GPR
Decomposition approach
Assign mode for each activity
Specify start time for each activity
Integration approach
Assign mode for each activity
Specify start time for each activity
Schedule

Fig. 3. Solution approach.

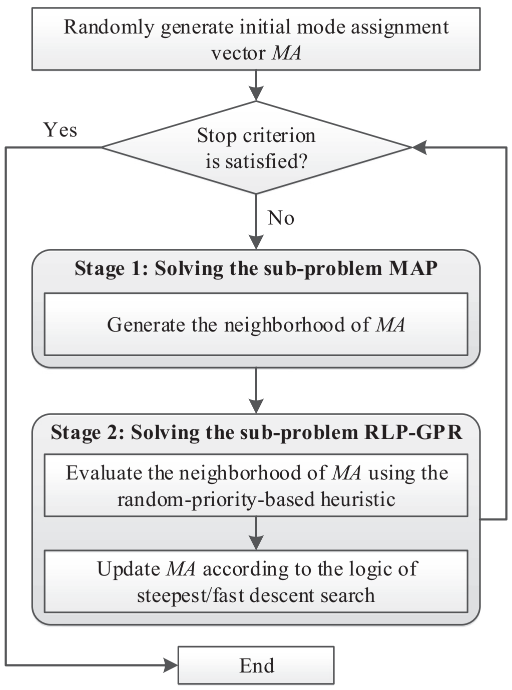
Randomly generate initial mode assignment vector MA
Stop criterion is satisfied?
Yes
No
Stage 1: Solving the sub-problem MAP
Generate the neighborhood of MA
Stage 2: Solving the sub-problem RLP-GPR
Evaluate the neighborhood of MA using the random-priority-based heuristic
Update MA according to the logic of steepest/fast descent search
End

Fig. 4. The framework of the descent algorithms.

ity's resource requirement. For example, if activity 2 is executed in mode 1 (2), its duration will be 2 (3) time units, and it will require 3 (2) resources.

The time-dependent start-to-start time lag, $\delta_{ij}^{m_i m_j}$, is placed besides each arc. Consider that activities 2 and 3 are performed in modes 2 and 1, respectively. The positive and negative time lags between activities 2 and 3 are 2 and −4, respectively, implying that activity 3 cannot be started until 2 time units after the start of activity 2 and that activity 3 must be started within 4 time units after the start of activity 2.

We assume that the project deadline is 8 ($\delta_{4,0} = -8$) and $c_1 = 1$. Fig. 2 shows an optimal schedule for the example project. The numbers in brackets indicate the activity execution modes. In this schedule, activities 1, 2 and 3 are executed in modes 1, 2 and 1, respectively. The corresponding value of the objective function (2) is 23.

3. Solution approach

Solving the MRLP-GPR involves assigning an execution mode and start time to each activity. Accordingly, the MRLP-GPR can be decomposed into two sub-problems: a mode assignment problem (MAP) and the RLP-GPR. The sub-problem MAP aims to obtain a time-feasible mode assignment vector *MA*. Once a feasible mode assignment is determined, a sub-problem RLP-GPR results. The RLP-GPR attempts to find a starting time vector *S* such that the objective function (2) is minimized.

Similar to the MRCPSP-GPR, based on the order that the sub-problems are addressed, there are two approaches for solving the MRLP-GPR (see Fig. 3): decomposition and integration. (a) The decomposition approach solves the sub-problems MAP and RLP-GPR sequentially: a mode assignment vector *MA* is obtained first by solving the MAP. Then, given this mode assignment vector *MA*, the sub-problem RLP-GPR is addressed. Based on this approach, we propose a fast descent algorithm and a steepest descent algorithm (see Section 4). (b) The integration approach treats the sub-problems as a whole, i.e., the mode assignment vector *MA* and the start time vector *S* are determined simultaneously. We develop a hybrid EDA based on this approach (see Section 5).

4. Descent algorithms

Following the decomposition approach, we design a fast descent algorithm and a steepest descent algorithm for solving the MRLP-GPR. Both algorithms address the sub-problems MAP and RLP-GPR sequentially.

4.1. Framework

Our descent algorithms focus primarily on obtaining the best mode assignment, since with the best (optimal) mode assignment, it is more likely to find an optimal or near optimal solution for the RLP-GPR and eventually for the MRLP-GPR. Therefore, our descent algorithms prioritize mode assignment searches over starting time searches. Specifically, we use a relatively fast and simple priority-based heuristic (see Section 4.2) to obtain an upper bound for the objective function value such that more time can be saved for evaluating more mode assignments. This idea is similar to solving the "best mode assignment problem," which was successfully applied to the MRCPSP-GPR (Ballestín et al., 2013; Barrios et al., 2011).

The framework of the descent algorithms is shown in Fig. 4. Both algorithms start with a randomly generated initial mode assignment vector $MA = (m_0, m_1, \ldots, m_{n+1})$, i.e., a starting solution. The *MA* specifies a specific mode for each activity. Then, our descent algorithms can be divided into two stages. The first stage is neighborhood generation, which corresponds to solving the sub-problem MAP. We define the neighborhood of *MA* as a collection of mode assignment vectors. Each of these vectors is obtained by randomly changing the mode of exactly one non-dummy activity to another one. Therefore, $n-1$ mode assignment vectors exist in the neighborhood of *MA*.

In the second stage, all the mode assignment vectors in the neighborhood of *MA* are evaluated heuristically. This approach essentially solves the sub-problem RLP-GPR heuristically, with a given mode assignment vector. The heuristic we used is described in Section 4.2. Heuristically solving the RLP-GPR provides an upper bound in a relatively short time such that we can transfer resources to searching for the best mode assignment. Each mode assignment vector is affiliated with an upper bound value, and this value plays the role of evaluating the mode assignment vector. The steepest descent algorithm chooses the mode assignment vector with the smallest upper bound value as the updated starting solution. The fastest descent algorithm, in contrast, selects the first encountered mode assignment vector that has a smaller upper bound value than *M* as the updated starting solution. If no better mode assignments exist in the neighborhood, then both algorithms will generate new neighborhoods for *MA* randomly.

By repeating the above two stages, both algorithms continue to update the starting solution and generate new mode assignment vectors until a prescribed stop criterion is met.

Our descent algorithms make the multi-mode resource leveling automatic by mimicking the project manager's decision-making process in a rigorous fashion. The MRLP-GPR arise often in practice. If there is no automated algorithm, a common-sense method is often adopted. Project managers assign resources (e.g., manpower, equipment, machinery, etc.) to activities. Then project managers develop time tables for the activities based on the allocated resources to optimize certain project objectives, such as resource leveling. Because the resulting time table may not be satisfactory, the project managers have to re-obtain a different allocation of resources. The "mode assignment and resource leveling" process continues until a satisfactory schedule is generated. However, this process is very time-consuming, and even if repeated many times cannot ensure a good schedule. Therefore, it is very important to use automatic algorithms for making resource-leveling decisions.

4.2. Random-priority-based heuristic

We propose a random-priority-based heuristic to evaluate the mode assignments appearing in the second stage of the descent algorithms. In addition, this heuristic can also be used as a decoding procedure that transforms a schedule representation into a schedule in the hybrid EDA, as will be discussed in Section 5.

Algorithm 1 gives the pseudo-code of the random-priority-based heuristic. For the schedule returned by Algorithm 1, we can calculate the corresponding value of the objective function (2). This value is used to evaluate the mode assignment vector in the descent algorithms.

There are three important input parameters for the random-priority-based heuristic: (1) A mode assignment vector *MA* is provided by the first stage of the descent algorithms. Once the *MA* is determined, a single-mode RLP-GPR results. (2) An activity list $AL=(\pi_0,\pi_1,\ldots,\ \pi_{n+1})$, which is a permutation of all activities in terms of π_i, indicates the activity number in the *AL* (the dummy start/end activity is always kept in the first/last position). In the random-priority-based heuristic, an *AL* is generated randomly, i.e., activities are ordered randomly in the *AL*. Note that the *AL* is not restricted to feasible precedence since the precedence constraints are handled by the random-priority-based heuristic. (3) A shift key vector is $SK=(sk_0,sk_1,\ldots sk_{n+1})$, each of whose elements is a random number belonging to the interval [0, 1] that determines which time point should be selected as each activity's starting time given the possible early and late starting times. Specifically, sk_i indicates the proportion of the difference between activity *i*'s starting time and the possible earliest starting time in the time window formed by *i*'s possible earliest and latest starting times, meaning that sk_i determines for how many periods activity *i* is scheduled beyond its possible earliest starting time. In the random-priority-based heuristic, *SK* is also generated randomly.

The random-priority-based heuristic sequentially chooses activities according to the *AL*. For each selected activity *i*, the starting time is set to $s_i = es_i(S', MA) + sk_i \times [ls_i(S', MA) - es_i(S', MA)]$. In this way, the resulting starting time s_i for activity *i* lies in the time window formed by *i*'s feasible early and late starting times, thus satisfying the precedence relations. Then, one must update the possible early and late starting times for the rest of the activities. We repeat the steps provided above until every activity has been assigned a starting time. In Algorithm 1, if activity *i* cannot reach *j*, then the longest path length $l_{ij} = -\infty$.

Note that in Algorithm 1, the activity list and the shift key vector can also be generated by a priority rule. However, to accelerate the heuristic, we simply use random numbers, meaning that we place more effort into the mode assignment searching phase. In contrast, as discussed in the next section, our hybrid EDA concentrates not only on the mode assignment but also on the random and shift key.

5. A hybrid EDA

Built upon statistical learning theory, the EDA is a population-based evolutionary algorithm for solving challenging optimization problems (Larrañaga & Lozano, 2002; Wang, Fang, Suganthan, & Liu, 2014). Probability models, which are the core of the EDA, are used to describe the distribution information of the candidate solutions in the search space. New populations are generated by sampling according to the probability models.

Following the integration approach, we propose a hybrid EDA that combines probability models and GA operators to generate new individuals. In Section 5.1, the general framework for our hybrid EDA is introduced. Schedule encoding and decoding mechanisms are given in Section 5.2. Section 5.3 describes how to generate new individuals using the operators for EDA and GA. Section 5.4 presents the probability model updating mechanism.

5.1. Framework

The framework of our hybrid EDA is shown in Fig. 5. First, the EDA population consists of *POP* individuals (Section 5.2). The probability matrices PM_{MA} and PM_{AL} that are used to generate mode assignment and activity lists are initialized. *POP* initial shift key vectors are also generated. Then, the following procedures are repeated until the stop criterion is met: *POP* mode assignment vectors and *POP* activity lists are generated by sampling the probability matrices PM_{MA} and PM_{AL}, respectively (Section 5.3). The *POP* new individuals are evaluated using the random-priority-based heuristic (Section 5.2). The top *P* ($P < POP$) individuals are selected from the population to update the probability matrices PM_{MA} and PM_{AL} (Section 5.4). At the same time, crossover and mutation operators are applied to shift key vectors (Section 5.3). Finally, the EDA returns an optimized schedule.

5.2. Schedule encoding and decoding

In our hybrid EDA, an individual corresponds to a schedule and is represented by a triple $I=(MA,\ AL,\ SK)$ of a mode assignment vector *MA*, an activity list *AL*, and a shift key vector *SK*. The definitions of *MA*, *AL* and *SK* were discussed in Section 4.

Each individual $I=(MA,\ AL,\ SK)$ can be decoded into a schedule by applying the so-called multi-mode resource leveling schedule generation scheme (we directly use the random-priority-based heuristic of Section 4.2 as the schedule generation scheme). Based on the mode indicated by the mode assignment *MA*, we sequentially choose the activity from the activity list *AL* and determine its starting time indicated by the shift key vector *SK*. The fitness of an individual $I=(MA,\ AL,\ SK)$ is equal to the value of the objective function (2).

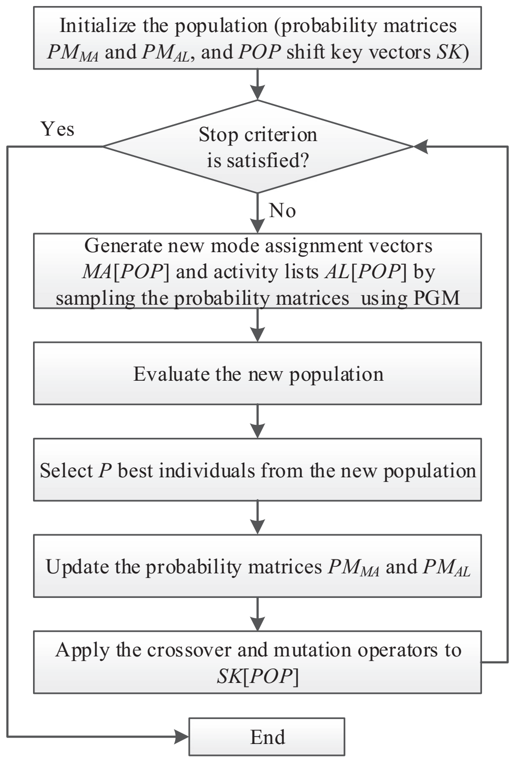
Initialize the population (probability matrices PM_{MA} and PM_{AL}, and POP shift key vectors SK)
Stop criterion is satisfied?
Yes
No
Generate new mode assignment vectors $MA[POP]$ and activity lists $AL[POP]$ by sampling the probability matrices using PGM
Evaluate the new population
Select P best individuals from the new population
Update the probability matrices PM_{MA} and PM_{AL}
Apply the crossover and mutation operators to $SK[POP]$
End

Fig. 5. The framework of the hybrid EDA.

The following proposition and corollary guarantee that our schedule encoding and decoding mechanisms have the potential to find optimal solutions.

Proposition 1. *For a MRLP-GPR instance with any time-feasible mode assignment, there exists at least one feasible schedule representation $I=(MA, AL, SK)$.*

Proof. For a MRLP-GPR instance, if the mode assignment *MA* is fixed, a RLP-GPR instance results. As discussed in Section 2.2, if the mode assignment is further time-feasible, there exists at least one schedule that satisfies the temporal constraints (3) and (4) of our model. Since our model does not explicitly consider resource constraints, the schedule (solution) is also feasible for the resulting RLP-GPR instance. Li et al. (2017) have proven that for any feasible solution of a RLP-GPR instance, there exists at least one random key *RK* and shift key *SK* representation corresponding to the feasible solution. An activity list *AL* can be obtained by sorting activities according to the random key values. Hence, the shift key *SK*, the activity list *AL* and the time-feasible mode assignment *MA* form a feasible schedule representation $I=(MA, AL, SK)$ for the MRLP-GPR instance. □

Corollary 1. *If a feasible solution of a MRLP-GPR instance is optimal, the corresponding schedule representation $I=(MA, AL, SK)$ is also optimal.*

5.3. Generation of new individuals

New individuals are generated jointly using the operators for EDA and GA. Specifically, a new mode assignment vector *MA* and activity list *AL* are generated by sampling according to the probability model of the EDA. A new shift key vector is obtained by applying the crossover and mutation operators of the GA.

5.3.1. Definition of the probability model

Our probability model consists of a probability matrix PM_{MA} and a probability vector PM_{AL}. PM_{MA} predicts the mode assigned to each non-dummy activity as follows:

$$PM_{MA}=\begin{pmatrix}\lambda_{11} & \cdots & \lambda_{1M}\\ \vdots & \ddots & \vdots\\ \lambda_{n1} & \cdots & \lambda_{nM}\end{pmatrix},$$

where the element λ_{im} represents the probability that mode m is assigned to activity i. Thus, λ_{im} reflects the suitability of activity i to be executed in mode m. In PM_{MA}, $M=\max\{M_1,M_2,\dots, M_n\}$. For λ_{im}, if $m > M_i$, we let $\lambda_{im}=0$.

In the initialization phase, to uniformly sample the entire solution space, we set the elements of PM_{MA}

$$\lambda_{im}=\begin{cases}\frac{1}{M_i} & \text{if } m\le M_i\\ 0 & \text{else}\end{cases},\quad \text{for } i=1,\dots,n;\ \ m=1,\dots,M$$

The probability vector PM_{AL} predicts the probability that a non-dummy activity is chosen when constructing the activity list *AL*, and $PM_{AL}=(\alpha_1,\alpha_2,\dots, \alpha_n)$. PM_{AL} can be explained as follows: given the set N' of activities that have not been added to the *AL*, the probability that activity i ($i\in N'$) is selected to *AL* is $\alpha_i/\sum_{j\in N'}\alpha_j$.

Similar to PM_{MA}, to uniformly sample the entire solution space, we initialize PM_{AL} as follows: $PM_{AL}=(\frac{1}{n},\dots,\frac{1}{n})$.

5.3.2. Probability-generating mechanism

Based on the probability model defined in the previous section, we devise a PGM to produce a new mode assignment vector *MA* and activity list *AL* (see Algorithms 2 and 3 for pseudo-code).

In Algorithm 2, the mode for each activity is determined sequentially from activity 1 to activity n. The probability $\Pr_{MA}(i,m)$ that mode m is assigned to activity i is calculated as $\Pr_{MA}(i,m)=\frac{\lambda_{im}}{\sum_{j=1}^{M_i}\lambda_{ij}}$. Pr serves as the cumulative probability and is compared with the random number $r\in(0, 1)$.

In Algorithm 3, the initial activity list *AL* is empty; the activity number π is inserted into the *AL* one by one. The order that activities are selected is determined by PM_{AL}. π_{pos} denoting the activity number appearing in the *AL*, where *pos* indicates the *pos*th position of the *AL* ($pos=1, 2, \dots, n$). After an activity i is inserted into the *AL*, the corresponding element in PM_{AL} is set to zero, i.e., $\alpha_i=0$. In this way, each activity is selected exactly once. As mentioned in Section 5.3.1, the probability that activity i ($i\in N'$) is chosen as the next activity for *AL* is $\alpha_i/\sum_{j\in N'}\alpha_j$.

5.3.3. Crossover and mutation operators

The two-point crossover and one-point mutation operators that arise from the GA are used to generate new shift key vectors. For the two-point crossover, we first determine *POP*/2 pairs of parent individuals, one half of which are called father individuals and the other half mother individuals. The father individual is chosen from the population with a predefined probability P_c. The mother individuals are formed by the best *POP*/2 individuals. A father individual and a mother individual generate son and daughter individuals. Specifically, we first generate two crossover points t_1 and t_2 randomly such that $0 < t_1 < t_2 < n+1$. Then, the elements of the father's (mother's) shift key vector elements between t_1 and t_2 are passed to the corresponding positions of the son's (daughter's) shift key vector. The rest of the positions of the son's (daughter's) vector are filled by the corresponding elements of the mother's (father's) vector.

In the one-point mutation, for each child individual, every element in the shift key vector is updated with a probability of P_m.

Specifically, for each sk_i ($i = 1, \ldots, n$) in the shift key vector, we generate a random number rn ($0 < rn < 1$). If $rn < P_m$, we replace the corresponding sk_i with rn.

5.4. Probability model updating mechanism

At the end of each iteration, the probability model is updated such that it can give a better prediction of the solution distribution. Specifically, given POP individuals generated according to the probability matrices, the best P ($P < POP$) individuals are chosen to update the probability matrices PM_{MA} and PM_{AL} according to the following equations:

$$\lambda'_{im} = (1-\beta)\cdot\lambda_{im} + \frac{\beta}{P}\sum_{j=1}^{P}\eta_{im}^{j}, \quad (1 \le i \le n, 1 \le m \le M_i) \tag{9}$$

$$\alpha'_i = (1-\beta)\cdot\alpha_i + \frac{\beta}{P}\sum_{j=1}^{P}\omega_i^{j}, \quad (1 \le i \le n) \tag{10}$$

where λ'_{im} (α'_i) is the updated value of λ_{im} (α_i), and β is the learning speed.

Given an individual j, η_{im}^{j} indicates whether a mode is assigned to an activity:

$$\eta_{im}^{j} = \begin{cases} 1 & \text{if mode } m \text{ is assigned to activity } i \\ 0 & \text{else} \end{cases}.$$

ω_i^j can be interpreted as a weight indicating the importance of putting an activity on a certain position of the activity list AL since the activity list AL is created by placing the first selected activity in the first position, second selected activity in the second position, and so on. Therefore, we let the activity appearing in the first position of the AL have the largest weight n, the activity appearing in the second position have the second largest weight $n-1$, ..., and the activity appearing in the last position have the smallest weight of 1. Accordingly, $\omega_i^j = \frac{n-pos+1}{n+(n-1)+\ldots+1} = \frac{n-pos+1}{(n+1)n/2}$, where pos is the position at which activity i appeared in AL.

5.5. Computational complexity analysis

We briefly analyze the computational complexity of the proposed EDA. In what follows, we focus on the upper bound of the worst-case running time. In the initialization phase, we need to compute the earliest/latest start time for each activity, which mainly relies on the Floyd-Warshall algorithm (Lawler, 1976; Demeulemeester & Herroelen, 2002). As mentioned in Section 2.1, this algorithm has a time complexity of $O(n^3)$, where n is the number of activities.

In the proposed EDA, Algorithm 1 is used to decode an individual into a schedule and it is with the complexity of $O(n^2)$. Algorithm 2 serves for generating a mode assignment vector and it is with the complexity of $O(n \times M)$. Algorithm 3 is used to generate an activity list and it is with the complexity of $O(n^2)$. Each iteration of our EDA involves POP individuls. Therefore, in each iteration, the complexities related to Algorithms 1, 2 and 3 are $O(POP \times n^2)$, $O(POP \times n \times M)$ and $O(POP \times n^2)$, respectively. When updating the probability matrices, we need to select P best individuals from the population. In the crossover operator, we need to select $POP/2$ best individuals as mother individuals. Both of the selection procedures use the quick sorting algorithm and hence both of them have worst-case time complexities of $O(P^2)$ and $O((POP/2)^2)$, respectively.

As we can see, the computational complexity of our EDA is acceptable. In the next section, our EDA will be further evaluated by computational experiments.

Table 1
Primary parameter settings for the test set.

Parameter	Value
n	10, 20, 30
RT	0.25, 0.5, 0.75
RF	1.0
RS	0.0, 0.25, 0.5
M_i	2, 3, 5
K	5
r_{im_ik}	Uniformly drawn from [1,5]
d_{im_i}	Uniformly drawn from [1,10]

6. Computational results

We programmed the proposed algorithms in MATLAB R2014a and performed the computational experiments on an Intel Core i5 3.20 GHz PC with 4 GB RAM under the 64-bit version of Windows 7. The benchmark instances and performance measures employed in the experiments are introduced in Sections 6.1 and 6.2, respectively. Section 6.3 discusses the parameter settings of the proposed algorithms using the Taguchi method for design of experiments (DOE). In Section 6.4, the solution quality of our algorithms is investigated, and the computational results obtained by comparing our algorithms with three baseline heuristics are reported.

6.1. Benchmark instances

The test set from Kolisch, Schwindt, and Sprecher (1999) is used to evaluate the performance of our algorithms. The test set can be downloaded from http://www.wiwi.tu-clausthal.de/en/abteilungen/produktion/forschung/schwerpunkte/project-generator/mrcpspmax/. The test set is produced by the project scheduling problem instance generator ProGen/max (Schwindt, 1998). By specifying different control parameters, ProGen/max can generate problem instances with different characteristics. Table 1 gives the main parameter settings used by the test set. In particular, the restrictiveness of Thesen (RT ∈ [0, 1]) indicates how tightly the precedence constraints restrict the number of feasible activity sequences (Schwindt, 1998). For a parallel (series) network, RT = 0 (RT = 1). The resource factor (RF ∈ [0, 1]) represents how many resource types are required by an activity on average. The resource strength (RS ∈ [0, 1]) reflects the scarceness of the resource capacity. A smaller RS value is correlated with scarcer resources. Since we do not consider non-renewable resources in this paper, RF and RS in Table 1 are related only to renewable resources.

In the test set, for every combination of n, RT, RS and M_i, 10 problem instances are generated, resulting in $3 \times 3 \times 3 \times 3 \times 10 = 810$ instances in total.

For each instance, the project deadline is set to $\bar{d} = \alpha \times \bar{l}_{0,n+1}$, where α is a coefficient that is no less than 1 and $\bar{l}_{0,n+1}$ is an upper bound of the critical path length. Note that we do not directly use the critical path length. The reason is that under the GPR constraints, the critical path length cannot be computed by simply choosing the shortest mode for each activity. Even without considering resource constraints, obtaining the optimal mode assignment that leads to the shortest makespan is still challenging (De Reyck & Herroelen, 1999).

$\bar{l}_{0,n+1}$ is calculated as follows. For any two activities i and j, the time lag $\delta_{ij}^{m_i m_j}$ between them is replaced by the mode-independent upper bound $\bar{\delta}_{ij} = \max_{m_i}\max_{m_j}\{\delta_{ij}^{m_i m_j}\}$. If $\bar{\delta}_{ij}$ is less than 0, this time lag will be deleted. Then, based on the new time lags $\bar{\delta}_{ij}$ that have only non-negative values, the earliest start time for each activity can be computed using CPM, and we obtain $\bar{l}_{0,n+1} = es_{n+1}$.

6.2. Performance measures

The performance of our algorithms (SD, FD and EDA) is compared to baseline algorithms. Specifically, we implemented three baseline algorithms: a random search algorithm (RSA) and two GAs.

The purpose that we use the RSA is to obtain the upper bounds of our problem instances. The process of the RSA is as follows. The RSA iterates until the termination criterion is met. In each iteration, a schedule is obtained by decoding a randomly generated individual $I=(MA, AL, SK)$. After the RSA terminates, the schedule with the smallest objective function value is returned.

The GAs used in our experiment are two variants of the GA proposed by Li et al. (2017), which is the best-performing metaheuristic for the RLP-GPR. In our implementation of the GAs, a schedule corresponds to a chromosome that is characterized by a mode assignment vector (see Section 5.2) and a random-shift key vector (Li et al., 2017). The random-priority-based heuristic is used as the schedule generation scheme. For the GA1, the crossover and mutation operators proposed by Li et al. (2017) are applied to the mode assignment vector. For the GA2, we use the probability model proposed in Sections 5.3 and 5.4 to operate the mode assignment vector. By doing so, the GA variants are able to handle multiple modes and solve the MRLP-GPR. The rest of the GAs are the same as Li et al. (2017).

The following two performance measures are used to evaluate our algorithms:

- Average relative deviations (ARDs): The average percentage deviations from the upper bound values. This means that the performance of each algorithm is compared with the RSA. Specifically, for a given algorithm ALG ($ALG \in$ {SD, FD, EDA, GA1, GA2}), the ARD is calculated as follows:

$$\mathrm{ARD}(ALG, \mathrm{RSA}) = \frac{\sum_{i=1}^{NrINS}[(rsa_i - alg_i)/rsa_i]}{NrINS} \tag{11}$$

where alg_i (rsa_i) is the objective function value of the ith instance obtained by the ALG (RSA) and $NrINS$ denotes the number of instances. A larger ARD value means that the algorithm ALG can obtain better solutions. From Eq. (11), we can see that the value of rsa serves as the upper bound. Generally, for any feasible solution of a minimization problem, the corresponding objective function value can be used as an upper bound. In this paper, we use the RSA-based upper bound for the MRLP-GPR. Specifically, given a MRLP-GPR instance ins, we solve it by using the RSA and obtain a feasible solution (schedule) fs_{ins}. Then the upper bound value rsa_{ins} equals the value of objective function (2) corresponding to fs_{ins}.

- Computation time (CPU): Average computation times in seconds for each instance.

6.3. Parameter settings

Before we conduct the experiment, the appropriate parameters of the proposed algorithms need to be determined. For the descent algorithms, in addition to the stop criteria, no other parameters require specification. Therefore, in the following, we discuss how to determine suitable parameter values for the EDA using the Taguchi method of DOE (Montgomery, 2008). The instances with 10 activities are used to perform the DOE test. We set the project deadline for each instance $\bar{d} = 1.5 \times \bar{l}_{0,n+1}$ and the unit cost for each resource type $c_k = 1$.

There are five key parameters for the EDA: the population size (POP), the number of selected individuals (P) to update the probability matrix, the learning speed (β), the crossover probability (P_c) and the mutation probability (P_m). For each of these parameters, we select 5 different values. Table 2 gives the combinations of these parameter values. Next, we will show the process of determining suitable values for the parameters. In the remainder of this subsection, these parameters are sometimes called factors.

Table 2
Combinations of parameter values.

Factor level	Parameters (Factors)				
	POP	P	β	P_c	P_m
1	50	0.1POP	0.005	0.7	0.01
2	100	0.2POP	0.01	0.8	0.05
3	150	0.3POP	0.05	0.9	0.1
4	200	0.4POP	0.1	0.95	0.2
5	250	0.5POP	0.5	0.99	0.3

Table 3
Orthogonal array and ARV values.

Experiment number	Factors					ARV
	POP	P	β	P_c	P_m	
1	1	1	1	1	1	0.575425
2	1	2	2	2	2	0.583909
3	1	3	3	3	3	0.617740
4	1	4	4	4	4	0.619290
5	1	5	5	5	5	0.618464
6	2	1	2	3	4	0.565653
7	2	2	3	4	5	0.606022
8	2	3	4	5	1	0.619268
9	2	4	5	1	2	0.621580
10	2	5	1	2	3	0.549897
11	3	1	3	5	2	0.606263
12	3	2	4	1	3	0.614908
13	3	3	5	2	4	0.620366
14	3	4	1	3	5	0.544482
15	3	5	2	4	1	0.554100
16	4	1	4	1	4	0.612796
17	4	2	5	2	5	0.621893
18	4	3	1	3	1	0.550931
19	4	4	2	4	2	0.549210
20	4	5	3	5	3	0.561287
21	5	1	5	1	5	0.621828
22	5	2	1	2	1	0.545014
23	5	3	2	3	2	0.547067
24	5	4	3	4	3	0.565860
25	5	5	4	5	4	0.576927

Table 4
Response table for ARV and rank for each factor.

Level	POP	P	β	P_c	P_m
1	**0.602966**	**0.596393**	0.553150	0.584053	0.587309
2	0.592484	0.594349	0.559988	0.586566	0.587922
3	0.588024	0.591075	0.591434	0.585339	**0.590717**
4	0.579223	0.580084	0.608638	**0.590434**	0.582322
5	0.571339	0.572135	**0.620826**	0.587644	0.585766
Delta	0.031626	0.024258	0.067677	0.006381	0.008395
Rank	2	3	1	5	4

First, we use the ARD as the average response variable (ARV), i.e., ARV(EDA, RSA) = ARD(EDA, RSA). The ARV measures the average percentage deviation of the EDA's objective function value from an upper bound value obtained by the RSA.

Then, we choose $L_{25}(5^5)$ as the orthogonal array, which means that we have 25 treatments in total and five levels for each of the five factors (parameters). Table 3 lists the orthogonal array and the ARV values. The ARV values are obtained by running the EDA with various combinations of the parameter settings given in Table 3. The termination criterion of the EDA and the RSA is the same, i.e., generating 5000 schedules at most.

According to Table 3, we can obtain the response table as shown in Table 4. The second to the sixth row of Table 4 shows the average value of the ARV for a factor with different levels. The

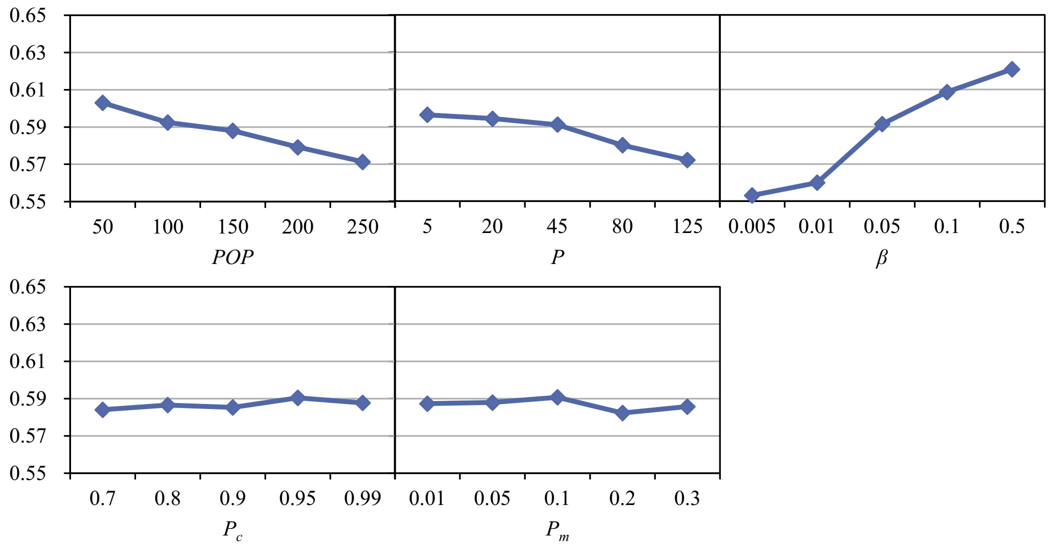

Fig. 6. EDA factor level trend.

Table 5
Computational results for the 10-activity instances.

#sched.	$\bar{d}$	M_i	SD ARD	SD CPU	FD ARD	FD CPU	EDA ARD	EDA CPU	GA1 ARD	GA1 CPU	GA2 ARD	GA2 CPU
1000		2	50.14	0.18	50.17	0.20	**52.27**	0.21	47.66	0.18	52.16	0.18
	$1.25\ \times \bar{l}_{0,n+1}$	3	62.03	0.18	62.18	0.20	**63.94**	0.21	55.74	0.17	63.55	0.18
		5	71.20	0.18	71.56	0.20	**72.39**	0.21	59.68	0.17	72.21	0.18
		2	48.10	0.18	48.30	0.19	**50.15**	0.21	46.57	0.18	50.01	0.18
	$1.5\ \times \bar{l}_{0,n+1}$	3	58.79	0.18	58.75	0.20	60.43	0.21	52.25	0.17	**60.52**	0.18
		5	71.79	0.18	71.74	0.20	**72.12**	0.22	61.22	0.18	71.88	0.19
5000		2	51.95	0.87	51.86	0.95	**53.52**	1.07	52.18	0.85	53.50	0.87
	$1.25\ \times \bar{l}_{0,n+1}$	3	62.68	0.89	62.42	0.97	**63.37**	1.09	59.78	0.85	63.06	0.89
		5	**73.58**	0.90	73.57	0.98	73.55	1.07	67.28	0.84	73.34	0.90
		2	49.88	0.87	50.16	0.96	**51.07**	1.02	50.07	0.85	50.90	0.88
	$1.5\ \times \bar{l}_{0,n+1}$	3	62.43	0.89	62.59	0.98	63.11	1.03	60.78	0.83	**63.13**	0.90
		5	**72.19**	0.90	72.08	0.99	71.63	1.04	66.42	0.82	71.41	0.90

row “Delta” indicates the range of the average ARV values for each factor. The row “Rank” denotes the significance rank for each factor. Fig 6 depicts the factor level trend.

Based on Table 4 and Fig. 6, we find that the learning speed has the largest impact on the ARV and that the crossover probability has the least significant impact. For each parameter, the level that results in the largest ARV value is chosen. Therefore, the parameter settings for the EDA are: $POP = 50$ (level 1), $P = 0.1\ \times POP = 5$ (level 1), $\beta = 0.5$ (level 5), $P_c = 0.95$ (level 4) and $P_m = 0.1$ (level 3). These parameter settings will be used in the succeeding experiments.

6.4. Comparison results

In our experiment, two stop criteria are used, i.e., the algorithms are terminated after either 1000 or 5000 schedules are generated. For each instance, we set the unit cost for each resource type $c_k = 1$.

Tables 5, 6 and 7 show the results obtained on the instances with 10, 20 and 30 activities, respectively. Bold numbers in these tables indicate superior performance for the corresponding combination of $\bar{d}$ and M_i. All of our proposed algorithms (i.e., SD, FD and EDA) can find feasible solutions for all instances within 10 seconds and outperform the RSA. The objective function values obtained by the SD and the FD are similar. Overall, the EDA performs best among the proposed algorithms. Only for the instances with 10 activities and 5 resource types, when the maximum number of schedules is 5000, the EDA is slightly worse than the SD. This finding indicates that unlike SD and FD, which center only on a mode search, a focus on both mode and starting time searches makes the EDA more effective.

As for comparison with the GA1 and the GA2, from Tables 5, 6 and 7, it is clear that the EDA outperforms the GA1. It is surprising that the SD and the FD perform better than the GA1 in most cases. In certain cases, the SD even obtains the best results. Since the SD and the FD focus primarily on a mode search, we conclude that such a priority is beneficial in some cases. It is also surprising that the GA2 outperforms the SD, the FD and the GA1 in most cases. The overall performance of the GA2 is weaker than the EDA. As mentioned in Section 6.2, the only difference between the GA1 and the GA2 is that the GA2 uses the probability model to generate the mode list. This also shows that our probability model is very effective.

For all algorithms, a larger number of modes leads to improved performance. In most cases, when the project deadline is tight, the algorithms can obtain solutions of higher quality. For SD and FD, their performance usually drops when the number of activities increases. However, this pattern does not apply to the EDA, indicating that the EDA is robust with respect to the number of activities.

Note that for a given algorithm under different stop criteria, the results in Tables 5, 6 and 7 cannot be directly compared. This limitation arises because for the stop criterion of 1000 (5000) schedules, the upper bounds are obtained by running the RSA with a

Table 6
Computational results for the 20-activity instances.

#sched.	$\bar{d}$	M_i	SD		FD		EDA		GA1		GA2	
			ARD	CPU	ARD	CPU	ARD	CPU	ARD	CPU	ARD	CPU
1000	$1.25 \times \bar{l}_{0,n+1}$	2	47.47	0.65	48.43	0.68	**53.42**	0.73	41.34	0.64	53.31	0.63
		3	58.04	0.65	59.05	0.68	**66.02**	0.76	47.15	0.64	65.92	0.63
		5	61.43	0.65	65.00	0.69	**73.76**	0.75	47.63	0.64	73.16	0.64
	$1.5 \times \bar{l}_{0,n+1}$	2	46.08	0.63	47.69	0.68	**52.55**	0.74	40.51	0.64	52.23	0.63
		3	54.77	0.64	57.04	0.69	**63.74**	0.77	45.21	0.64	63.52	0.63
		5	61.87	0.64	64.34	0.69	**73.44**	0.76	48.47	0.64	73.07	0.64
5000	$1.25 \times \bar{l}_{0,n+1}$	2	51.57	3.15	51.50	3.38	**55.27**	3.55	45.27	3.06	55.07	3.13
		3	62.92	3.18	63.13	3.41	67.01	3.58	51.17	3.04	**67.10**	3.16
		5	72.14	3.20	72.41	3.43	**74.87**	3.60	50.81	3.02	74.61	3.23
	$1.5 \times \bar{l}_{0,n+1}$	2	51.22	3.15	52.03	3.37	**55.12**	3.56	45.92	3.06	55.04	3.15
		3	62.87	3.20	63.45	3.43	**66.69**	3.58	51.75	3.04	66.35	3.17
		5	71.58	3.21	71.60	3.44	**73.69**	3.61	50.80	3.02	73.52	3.19

Table 7
Computational results for the 30-activity instances.

#sched.	$\bar{d}$	M_i	SD		FD		EDA		GA1		GA2	
			ARD	CPU	ARD	CPU	ARD	CPU	ARD	CPU	ARD	CPU
1000	$1.25 \times \bar{l}_{0,n+1}$	2	41.69	1.70	44.03	1.79	**52.00**	1.90	38.24	1.72	51.53	1.65
		3	46.92	1.71	49.42	1.80	**63.28**	1.94	41.94	1.73	63.20	1.67
		5	49.73	1.72	51.28	1.81	**71.70**	1.98	43.53	1.73	71.69	1.68
	$1.5 \times \bar{l}_{0,n+1}$	2	40.00	1.70	41.18	1.79	**50.77**	1.96	38.05	1.72	51.50	1.66
		3	44.09	1.72	46.99	1.86	**63.08**	2.00	42.87	1.73	62.88	1.67
		5	46.29	1.72	48.60	1.83	**70.91**	1.99	43.82	1.73	70.75	1.68
5000	$1.25 \times \bar{l}_{0,n+1}$	2	49.38	8.31	49.74	8.90	**54.78**	9.10	43.29	8.31	54.60	8.29
		3	56.78	8.41	58.14	8.97	**66.04**	9.24	44.28	8.30	65.30	8.42
		5	65.79	8.43	65.92	9.02	**74.86**	9.21	48.18	8.38	74.59	8.39
	$1.5 \times \bar{l}_{0,n+1}$	2	47.06	8.31	47.61	8.92	**53.88**	9.10	42.39	8.38	53.32	8.31
		3	57.55	8.41	57.94	9.00	**66.17**	9.19	47.16	8.31	66.16	8.37
		5	64.89	8.44	64.11	9.02	**74.73**	9.22	50.21	8.30	74.26	8.54

Algorithm 1
Pseudo-code of the random-priority-based heuristic.

```
Input: a mode assignment vector $MA$, an activity list $AL$ and a shift key vector $SK$
Output: schedule $S$
Determine the value of each activity's time lag according to $MA$;
$s_0 = 0$; $s_{n+1} = \bar{d}$; $N' = \{0, n+1\}$;
For all $i \in N \backslash N'$, $es_i(S', MA) = es_i$, $ls_i(S', MA) = ls_i$;
While $N \backslash N' \neq \emptyset$, do
    Select activity $i$ from $N \backslash N'$ according to the activity order indicated by $AL$;
    $N' = N' \cup \{i\}$;
    $s_i = es_i(S', MA) + \lfloor sk_i \times [ls_i(S', MA) - es_i(S', MA)] \rfloor$;
    For all $j \in N \backslash N'$,
        $es_j(S', MA) = \max\{es_j(S', MA),\ s_i + l_{ij}\}$;
        $ls_j(S', MA) = \min\{ls_j(S', MA), s_i - l_{ji}\}$;
    End for
End while
```

Algorithm 2
Generating the mode assignment vector using the PGM.

```
Input: probability matrix $PM_{MA}(\lambda_{im})$
Output: mode assignment vector $MA$
For $i = 1$ to $n$
    $m = 1$;
    $\Pr = \lambda_{im}$;
    Draw a random number $r$ from interval (0, 1);
    While $r > \Pr$
        $m = m + 1$;
        $\Pr = \Pr + \lambda_{im}$;
    End while
    $m_i = m$;
End for
Return $MA = (m_1, \ldots, m_n)$;
```

stop criterion of 1000 (5000) schedules. In other words, when the stop criterion is different, the upper bounds used in the ARD calculation are also different. Clearly, the upper bounds with 5000 schedules are tighter than (or at least the same as) the ones with 1000 schedules. Thus, although the results cannot be directly and fairly compared, to a certain extent, the following result still shows that the algorithms presented here can obtain improved schedules with more iterations: for a stop criterion of 1000 schedules, the ARDs for SD, FD, EDA, GA1 and GA2 on all instances are 53.36%, 54.76%, 62.55%, 46.77% and 62.39%, respectively. These values increase to 60.36%, 60.57%, 64.41%, 51.54% and 64.18% when the maximum number of schedules increases to 5000.

In summary, according to the comparison results, our hybrid EDA outperforms or is competitive with all of the baseline algo-

Algorithm 3
Generating the activity list using the PGM.

```
Input: probability vector $PM_{AL}(\alpha_i)$
Output: activity list $AL$
$U = \sum_{i=1}^{n} \alpha_i$;
For $pos = 1$ to $n$
    $\pi = 1$;
    $\Pr = \alpha_\pi$;
    Draw a random number $r$ from interval $(0, U)$;
    While $r > \Pr$
        $\pi = \pi + 1$;
        $\Pr = \Pr + \alpha_\pi$;
    End while
    $U = U - \alpha_\pi$;
    $\alpha_\pi = 0$;
    $\pi_{pos} = \pi$;
End for
Return $AL = (\pi_1, \ldots, \pi_n)$;
```

rithms (the RSA, the GA1 and the GA2). Our SD and FD perform better than the baseline algorithms (the RSA and the GA1) in most cases, and the SD obtains the best results in certain cases. Our proposed algorithms are effective for solving the MRLP-GPR.

7. Conclusions and future research

By considering multiple execution modes and generalized precedence relations, the MRLP-GPR has been studied for the first time. Built upon decomposition and integration solution approaches, we developed several heuristics for the NP-hard MRLP-GPR: (a) Based on decomposition approach, we proposed a steepest descent algorithm and a fast descent algorithm. Both algorithms solve the MRLP-GPR by obtaining activity mode assignments and activity starting times sequentially. A fast and simple priority-based heuristic is integrated into both algorithms. (b) Based on the integration approach, we devised a hybrid estimation of distribution algorithm that determines mode assignments and starting times in parallel. A series of novel components are designed to enhance the EDA, including multi-mode schedule encoding and decoding mechanisms, a problem-specific probability model and its updating mechanism, probability-generating mechanism, and combining probability models and GA operators to generate new individuals.

Computational results obtained on a large number of benchmark instances reveal the effectiveness and efficiency of the proposed heuristics. Our EDA outperforms or is competitive with three baseline heuristics (a random search algorithm and two variants of a GA (Li et al., 2017) that is the best-performing metaheuristic for the RLP-GPR). Our results can serve as a benchmark for future research.

For future work, it will be interesting to integrate our algorithms into project management software. This will make our algorithms more practical and lead to automated project resource leveling decision-making. It may be fruitful to develop efficient mixed-integer linear programming models and constraint programming models for the MRLP-GPR and to design more efficient meta-heuristics for solving larger instances with more than 30 activities. It may also be advantageous to improve the multi-mode resource leveling schedule generation scheme by introducing more efficient priority rules. Furthermore, the scope of practical applications of our study is limited by without considering non-renewable resources. In the future, we will extend our approach to take the non-renewable resources into account. Additionally, extending and applying our approach to Minimum Viable Product and other emerging practices in software project management could contribute greatly to the research subject, especially in a more agile business environment.

Acknowledgments

The authors thank the Editor-in-Chief, Prof. Binshan Lin, and the reviewers for providing constructive suggestions that have improved the quality of this paper significantly. This research was supported by the National Science Foundation of China [Grant numbers 71602106, 71702097]; the Humanities and Social Sciences Foundation of the Ministry of Education of China [Grant number 15YJCZH077]; and the College Young Teachers Program of Shanghai Municipal Education Commission [Grant number ZZSD16025].

References

Ballestín, F., Barrios, A., & Valls, V. (2013). Looking for the best modes helps solving the MRCPSP/max. *International Journal of Production Research, 51*(3), 813–827.
Ballestín, F., Schwindt, C., & Zimmermann, J. (2007). Resource leveling in make-to-order production: Modeling and heuristic solution method. *International Journal of Operations Research, 4*(1), 50–62.
Barrios, A., Ballestín, F., & Valls, V. (2011). A double genetic algorithm for the MRCPSP/max. *Computers & Operations Research, 38*(1), 33–43.
Bartusch, M., Möhring, R. H., & Radermacher, F. J. (1988). Scheduling project networks with resource constraints and time windows. *Annals of operations Research, 16*(1), 199–240.
Behrouz, A. N., Hojjat, N., & Esmaeil, M. (2012). A branch-and-bound procedure for resource leveling in multi-mode resource constraint project scheduling problem. *Research Journal of Recent Sciences, 1*(7), 33–38.
Coughlan, E. T., Lübbecke, M. E., & Schulz, J. (2015). A branch-price-and-cut algorithm for multi-mode resource leveling. *European Journal of Operational Research, 245*(1), 70–80.
Demeulemeester, E., & Herroelen, W. (2002). *Project scheduling: A research handbook*. Kluwer Academic Pub.
De Reyck, B., & Herroelen, W. (1999). The multi-mode resource-constrained project scheduling problem with generalized precedence relations. *European Journal of Operational Research, 119*(2), 538–556.
Dorndorf, U., Pesch, E., & Phan-Huy, T. (2000). A time-oriented branch-and-bound algorithm for resource-constrained project scheduling with generalised precedence constraints. *Management Science, 46*(10), 1365–1384.
Gather, T., Zimmermann, J., & Bartels, J. H. (2011). Exact methods for the resource levelling problem. *Journal of Scheduling, 14*(6), 557–569.
Guo, Y., Li, N., Zhang, H., & Ye, T. (2012). Elitist vector evaluated particle swarm optimization for multi-mode resource leveling problems. *Journal of Computational Information Systems, 8*(9), 3697–3705.
Heilmann, R. (2001). Resource–constrained project scheduling: A heuristic for the multi–mode case. *OR Spectrum, 23*(3), 335–357.
Heilmann, R. (2003). A branch-and-bound procedure for the multi-mode resource–constrained project scheduling problem with minimum and maximum time lags. *European Journal of Operational Research, 144*(2), 348–365.
Jędrzejowicz, P., & Ratajczak-Ropel, E. (2011). A-Team for solving MRCPSP/max problem. *Agent and Multi-Agent Systems: Technologies and Applications*, 466–475.
Kolisch, R., Schwindt, C., Sprecher, A., & Weglarz, J. (1999). Benchmark instances for project scheduling problems. In *Project scheduling - Recent models, algorithms and applications* (pp. 197–212). Boston: Kluwer.
Kreter, S., Rieck, J., & Zimmermann, J. (2014). The total adjustment cost problem: Applications, models, and solution algorithms. *Journal of Scheduling, 17*(2), 145.
Larrañaga, P., & Lozano, J. A. (2002). *Estimation of distribution algorithms: A new tool for evolutionary computation*. Netherlands: Springer.
Lawler, E. L. (1976). *Combinatorial optimization: Networks and matroids*. Courier Dover Publications.
Li, H., Xiong, L., Liu, Y., & Li, H. (2017). An effective genetic algorithm for the resource leveling problem with generalized precedence relations. *International Journal of Production Research*. doi:10.1080/00207543.2017.1355120.
Li, H., Xu, Z., & Demeulemeester, E. (2015). Scheduling policies for the stochastic resource leveling problem. *Journal of Construction Engineering and Management, 141*(2), 04014072.
Li, H., & Demeulemeester, E. (2016). A genetic algorithm for the robust resource leveling problem. *Journal of Scheduling, 19*(1), 43–60.
Markou, C., Koulinas, G. K., & Vavatsikos, A. P. (2017). Project resources scheduling and leveling using multi-attribute decision models: Models implementation and case study. *Expert Systems with Applications, 77*, 160–169.
Menesi, W., & Hegazy, T. (2015). Multimode resource-constrained scheduling and leveling for practical-size projects. *Journal of Management in Engineering, 31*(6), 04014092.
Moogk, D. R. (2012). Minimum viable product and the importance of experimentation in technology startups. *Technology Innovation Management Review, 2*(3), 23–26.
Neumann, K., & Schwindt, C. (1997). Activity-on-node networks with minimal and maximal time lags and their application to make-to-order production. *Operations-Research-Spektrum, 19*(3), 205–217.
Neumann, K., Schwindt, C., & Zimmermann, J. (2003). *Project scheduling with time windows and scarce resources: Temporal and resource-constrained project scheduling with regular and nonregular objective functions*. Springer.

Neumann, K., & Zimmermann, J. (1999). Resource levelling for projects with schedule-dependent time windows. *European Journal of Operational Research, 117*(3), 591–605.

Neumann, K., & Zimmermann, J. (2000). Procedures for resource leveling and net present value problems in project scheduling with general temporal and resource constraints. *European Journal of Operational Research, 127*(2), 425–443.

Rieck, J., Zimmermann, J., & Gather, T. (2012). Mixed-integer linear programming for resource leveling problems. *European Journal of Operational Research, 221*(1), 27–37.

Rieck, J., & Zimmermann, J. (2015). Exact methods for resource leveling problems. In *In handbook on project management and scheduling: 1* (pp. 361–387). Springer International Publishing.

Wang, L., Fang, C., Suganthan, P. N., & Liu, M. (2014). Solving system-level synthesis problem by a multi-objective estimation of distribution algorithm. *Expert Systems with Applications, 41*(5), 2496–2513.