Associate Professor Hongbo LI, PhD
School of Management, Shanghai University, China
E-mail: ishongboli@gmail.com, hongbo_li@shu.edu.cn
Ziyi HU, Master's degree
School of Management, Shanghai University, China
E-mail: 512608474@qq.com
Hanyu ZHU, Master student
School of Management, Shanghai University, China
E-mail: zhuhanyu1101@qq.com (Corresponding author)
Associate Professor Yinbin LIU, PhD (Corresponding author)
School of Management, Shanghai University, China
E-mail: yinbinliu@126.com

PREEMPTIVE RESOURCE LEVELING IN PROJECTS

Abstract: *As a well-known NP-hard problem in project scheduling, the resource leveling problem (RLP) has attracted many researchers' attentions. In the RLP, a typical assumption is that activities are non-preemptive during project execution, which means that activities cannot be interrupted once they have been started. However, preemption is not uncommon in project management practice and existing studies already show that it is beneficial to consider preemption when leveling resource usage. Therefore, we investigate the preemptive resource leveling problem and design a genetic estimation of distribution algorithm (GEDA). To analyze the performance of the GEDA, we conduct extensively computational experiments on 2160 randomly generated instances. We also examine the impacts of various factors on the GEDA. Comparative experimental results show that the GEDA outperforms the existing meta-heuristic algorithm.*

Keywords: *Project scheduling; Resource leveling; Preemption; Meta-heuristic algorithms.*

JEL Classification: M11, C44, C61

1 Introduction

It is increasingly popular to manage work in the form of projects. 20% of the global economic activities are organized by projects, which generate an annual economic value of approximately 12 trillion dollars (Li & Hall 2019). In project

DOI: 10.24818/18423264/55.4.21.04

management, allocating various types of resources effectively is a critical success factor for projects. As a well-known NP-hard problem in project resource scheduling, the resource leveling problem (RLP) has attracted many researchers' attentions. In the RLP, a baseline schedule is formed by specifying start time for each activity. This schedule levels the resource utilization while satisfying the precedence relations constraints and the project deadline constraint. Leveled resource usage can reduce unnecessary capital expenditures and avoid hasty deployment of temporary resources (Doulabi et al. 2011; Li et al. 2018). The systematic reviews of the RLP can be further referred to Demeulemeester & Herroelen (2002), Neumann et al. (2003).

In the RLP literature, a typical assumption is that activities are non-preemptive during project execution, which means that activities cannot be interrupted once they have been started. However, preemption is not uncommon in project management practice. Due to management needs or external conditions, the execution of certain activities may be temporarily interrupted, e.g., stopping machines after work, software developers switching between different tasks, etc. (Ballestín et al. 2009). Preemption has been considered in many project scheduling problems, such as multi-project scheduling (Bock & Patterson 1990), multi-objective project scheduling (Nudtasomboon & Randhawa 1997), resource-constrained project scheduling (Ballestín et al. 2008; Demeulemeester & Herroelen 1996), etc.

In the RLP, taking preemption into consideration can lead to more levelled resource utilization (Doulabi et al. 2011; Liu et al. 2019). Therefore, more attentions are being paid to the preemptive resource leveling problem (PRLP) and several exact and meta-heuristic algorithms have been proposed. In terms of exact algorithms, Son & Mattila (2004) develop a linear programming model based on binary variables for the PRLP. Hariga & El-Sayegh (2011) study a mixed integer programming model to minimize the cost caused by resource fluctuations and activity preemption. Nadjafi et al. (2013) proposes a branch-and-bound procedure to solve the PRLP.

The research on the meta-heuristic algorithms for the PRLP is scarce. Razavi & Mozayani (2007) propose a genetic algorithm and they only allow noncritical activities to be interrupted. Alsayegh & Hariga (2012) design a hybrid meta-heuristic for PRLP, in which the cost of splitting activities is taken into consideration. Splitting critical activities is not allowed neither. Doulabi et al. (2011) design a genetic algorithm for the PRLP, in which not all activities are allowed to be preempted. Their objective function consists of two terms: minimizing the cost caused by resource utilization variations and that by activity preemption. In their genetic algorithm, the proposed encoding scheme tends to generate individuals violating the precedence relations constraints. So the authors add a feasibility repair mechanism to the genetic algorithm. They validate the genetic algorithm using 220 randomly generated instances and the data of a tunnel construction project. Different from the above-mentioned studies, we allow any activity to be interrupted at any integer time point. In this case, although the searching space will be enlarged, it increases the possibility of finding a better schedule that results in more leveled resource utilization.

Due to the RLP is NP-hard (Neumann et al. 2003), exact algorithms are only suitable for small instances. They can even hardly obtain feasible solutions for large-scale instances in a reasonable time. In this situation, heuristic algorithms become the only option. In addition, the introduction of preemption further increases the complexity of the RLP, which makes the RLP more difficult to be solved. Specifically, at the end of each unit time period, decisions about which activities should be scheduled need to be made, which significantly increases the number of feasible solutions. While in the non-preemptive RLP, the timing for scheduling decisions only corresponds to the completion of activities. Therefore, efficient meta-heuristic algorithms need to be devised to handle the PRLP.

In this paper, we investigate the PRLP and develop a genetic estimation of distribution algorithm (GEDA) that combines the genetic algorithm (GA) and the estimation of distribution algorithm (EDA). Our main contributions are as follows:

(1) For the first time, a meta-heuristic, GEDA, is proposed for solving the PRLP, in which each activity is allowed to be interrupted at any integer time point. On the one hand, to the best of our knowledge, there has been no research on meta-heuristics for this kind of PRLP. On the other hand, GAs and EDAs have been successfully applied to various project scheduling problems (Li et al. 2018; Li & Dong, 2018; Forghani & Fatemi-Ghomi, 2019). However, no studies that apply the EDA to solve the PRLP are found.

(2) In the GEDA, a schedule is encoded into an individual consisting of an activity list and a shift key vector. This encoding mechanism ensures that an individual always corresponds to a feasible schedule. The activity list and the shift key vector are updated by specially designed operators based on the EAD and the GA, respectively.

(3) To analyze the effectiveness and efficiency of the GEDA, we conduct extensively computational experiments on 2160 randomly generated instances. The results show that, for small-scale instances with no more than 30 activities whose optimal solutions are known, in most cases, the difference between the solutions obtained by the GEDA and the optimal solutions is within 5%, and the average calculation time of each instance is less than 0.6 seconds. For small-scale instances whose optimal solutions are not known and large-scale instances with 100 activities, the GEDA obtains better solutions than CPLEX on more than half of the instances, while requiring only about 1/1000 of the calculation time of CPLEX.

The rest of the paper is organized as follows. Section 2 states the PRLP, presents the corresponding optimization model, and gives an example for the PRLP. Our GEDA is described in Section 3. We perform computational experiments to analyze the performance of the GEDA in Section 4. The last section summarizes the paper and prospects future research directions.

2. The preemptive resource leveling problem

2.1 Problem statement

The PRLP is described as follows. A project is represented by an activity-on-node network $G = (N, A)$. N is the set of nodes representing activities, $N = \{0,1,2 \ldots, n, n+1\}$. The activities are topologically numbered from 0 to $n+1$. Activities 0 and $n+1$ are dummy ones, indicating the start and the end of the project. A is the set of arcs indicating the precedence relations among activities. If $(i,j) \in A$, activity i is the predecessor of activity j, which means that activity j cannot be started until activity i is finished. This kind of precedence relations is the same as that used in the critical path method (Demeulemeester & Herroelen 2002). The project deadline is $\bar{d}$.

The start time and duration of non-dummy activity i are denoted as s_i and d_i, respectively. The duration of dummy activities is 0. Without loss of generality, we assume that all time-related parameters are integer. There are K types of renewable resources. During the execution of non-dummy activity i, its requirement for resource type k at each time period is r_{ik}, $k = 1,2,\ldots,K$. Dummy activities do not need any resources.

In project management practice, it is not uncommon to interrupt some activities such that resources can be re-allocated to other more critical activities. The interrupted activity is resumed after the resources become sufficient again. Therefore, without loss of generality, we assume that each activity is allowed to be interrupted at any integer time point during execution. For activity i, let p_i denote its number of preemption, $p_i \in [0, P_i]$, where P_i is its maximum number of preemption ($P_i = d_i - 1$). If $p = 0$, then activity i is executed without preemption; If $p = P_i$, then activity i is interrupted after each time unit.

If activity i is interrupted p_i times, then activity i can be viewed as $p_i + 1$ sub-activities and these sub-activities form a set $I_i = \{i_1, i_2, \ldots, i_{p_i+1}\}$. For each sub-activity $i_q \in I_i$ ($q = 1,2,\ldots,p_i + 1$), its start time and duration are denoted as s_{iq} and d_{iq}, respectively.

The PRLP aims at minimizing the variations in resource utilization by constructing a baseline schedule S that decides the start time and duration of each sub-activity under a preemption environment, while meeting the precedence relation constraints and the project deadline constraint.

2.2 Optimization model

Treating each activity i as d_i successive sub-activities, each of which has a duration of 1, we present the optimization model for the PRLP as follows:

Minimize $\sum_{k=1}^{K} \sum_{t=1}^{\bar{d}} (c_k \cdot u_{kt}^2)$ (1)

Subject to: $s_{0,1} = 0$ (2)

$s_{n+1,1} \leq \bar{d}$ (3)

$s_{i,d_i} + 1 \leq s_{j,1} \quad \forall (i,j) \in A$ (4)

$s_{i,q} + 1 \leq s_{i,q+1} \quad \forall i \in N;\ q = 1,2,\ldots,d_i - 1$ (5)

$\sum_{i \in V_t} r_{ik} = u_{kt} \quad k = 1,2,\ldots,K;\ t = 1,2,\ldots,\bar{d};$ (6)

$s_{i,q} \geq 0 \quad \forall i \in N;\ q = 1,2,\ldots,d_i$ (7)

The objective function (1) minimizes the weighted sum of the square of resource usage (Demeulemeester & Herroelen 2002), where u_{kt} is the usage of resource type k at each time period t and c_k is the weight of resource type *k*. The constraint (2) ensures that the project begins at time zero. The constraint (3) promises that the project is completed no later than the deadline. The constraints (4) represent the precedence relations, which means that the successor activities cannot be started until the last sub-activities of its predecessor have been finished. The constraints (5) indicate that the sub-activities of each activity should be executed sequentially. The constraints (6) are used to calculate u_{kt}, in which V_t is the set of sub-activities that are executing during time period t. The constraints (7) let the start time of each sub-activity be non-negative.

Obviously, the object function (1) is non-linear due to the existence of u_{kt}^2. In addition, the set V_t in constraints (6) also make the constraints non-linear. Therefore, the above model is non-linear. However, based on the linearization method proposed in Rieck et al. (2012) and Liu et al. (2019), this non-linear model can be transformed into a linear integer programming model.

The PRLP can be reduced to the RLP by allowing each activity to be interrupted zero times. This means that the PRLP is a generalization of the NP-hard RLP (Neumann et al., 2003). Therefore, the PRLP is also NP-hard. In this case, exact algorithms can hardly find a satisfactory solution in a reasonable time for the large-scale PRLPs. We will design a meta-heuristic algorithm in Section 3 to efficiently obtain satisfactory solutions for the PRLP.

2.3 Example

We use an example to illustrate the PRLP and show that it is possible to obtain a more leveled schedule after considering preemption. Figure 1 displays a project network with 4 non-dummy activities. The rectangles and the arrows represent activities and precedence relations, respectively. There is one resource type. The number above (below) each rectangle is the duration (resource requirement) of the corresponding activity. The project deadline is 4. The weight of the resource $c_1 = 1$.

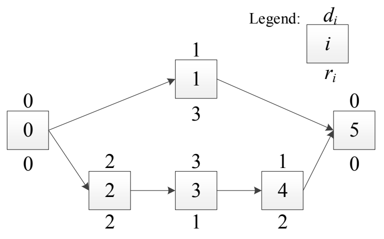
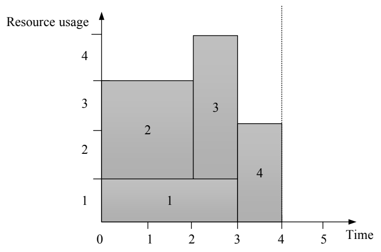


Figure 1. Example project network

When the preemption is not allowed, Figure 2 shows the resulting optimal schedule $S = (0,0,0,2,3,4)$ with the value of the objective function being $\sum_{k=1}^{1}\sum_{t=1}^{4}\left(c_k \cdot u_{kt}^2\right) = 3^2 + 3^2 + 4^2 + 2^2 = 38$.

If we allow the preemption, we can obtain the optimal schedule $S' = [0,(0,3),0,2,3,4]$ as shown in Figure 3. In S', activity 1 is interrupted once. The first sub-activity (1a) of activity 1 starts at time 0 with a duration of 2 and the second one (1b) starts at time 3 with a duration of 1. The corresponding objective value of S' is $\sum_{k=1}^{1}\sum_{t=1}^{4}\left(c_k \cdot u_{kt}^2\right) = 3^2 + 3^2 + 3^2 + 3^2 = 36$. It can be seen that a more leveled schedule is constructed after allowing the preemption.



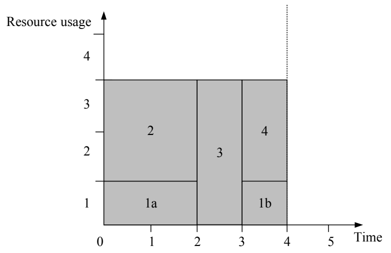


Figure 3. The optimal schedule with preemption

Figure 2. The optimal schedule without preemption

3. Genetic estimation of distribution algorithm

The EDA is a prospering meta-heuristic for solving optimization problems. There are some similarities between the EDA and the GA, e.g., they are population-based algorithms and the population is updated at each iteration. The EDA is based on statistical learning theory. The core of the EDA is probability models that describe the distribution of potential solutions in the searching space. The EDA updates its populations by sampling according to the probability models.

In our GEDA, there are POP individuals in the population. Each individual consists of two elements: an activity list AL and a shift key vector SK (Section 3.2). New individuals are generated by applying the EDA and the GA operators to the activity list and the shift key vector, respectively (Sections 3.3 and 3.4).

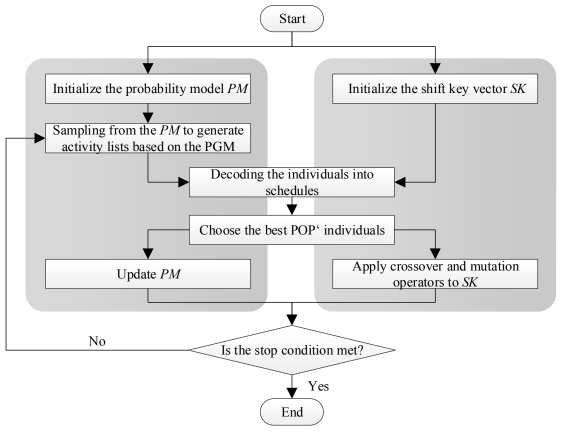

Start
Initialize the probability model PM
Initialize the shift key vector SK
Sampling from the PM to generate activity lists based on the PGM
Decoding the individuals into schedules
Choose the best POP' individuals
Update PM
Apply crossover and mutation operators to SK
No
Is the stop condition met?
Yes
End

Figure 4. The framework of the GEDA

The framework of the GEDA is described as follows (Figure 4). First, the probability model PM and the shift key vector SK are initialized. PM is used to generate activity lists. Then, based on the probability generating mechanism, POP activity lists are obtained by sampling the probability model PM. Next, the individuals are decoded into schedules (Section 3.2) and the corresponding objective function values are calculated. After that, PM is updated based on the top POP' individuals in terms of the objective function value. In the meanwhile, crossover and mutation operators are adopted to update the shift key vectors. The above process keeps iterating and the schedules are improved until a pre-determined termination condition is satisfied.

3.1 Unit time project network

To simplify the schedule encoding mechanism, we introduce the unit time project network.

Definition: Unit time project network $G' = (N', A')$. For each activity $i \in N$, splitting it into d_i sub-activities, each of which has a duration of 1. Let N' denote the set of the resulting sub-activities. For any two adjacent sub-activities, adding a precedence relation between them. The newly added and original precedence relations form the set A'. In this way, N' and A' form the unit time project network G'.

The set N' contains $subn = \sum_{i=1}^{n} d_i$ non-dummy sub-activities. We re-number these sub-activities according to the following rule: For sub-activity i_q ($q = 1,2,\dots,d_i$), its number in N' is $m = \sum_{j=1}^{i-1} d_j + q$. The dummy end activity in N' is numbered $subn + 1$. Figure 5 illustrate this rule using the example project shown in Figure 1.

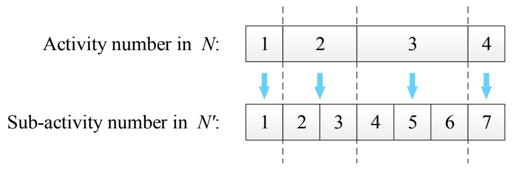


Figure 5. Numbering the sub-activities

After introducing the unit time project network, we do not need to specifically consider the interruption in the PRLP. Because for any two adjacent sub-activities i_{q1} and i_{q2} belonging to the same activity, if the finish time of i_{q1} equals the start time of i_{q2}, then it means that there is no interruption between i_{q1} and i_{q2}; otherwise, there exists interruption. In the following, our GEDA acts on the unit time project network G' to obtain the schedule for the RPLP.

3.2 Encoding and decoding

Properly encoding and effectively evaluating a schedule is the key to design effective meta-heuristic algorithms for solving the PRLP. In project scheduling research, the meta-heuristic algorithms usually operate on some kind of encoding of the schedule. The schedule generation mechanism is used to decode the encoded schedule.

In the GEDA, an individual corresponds to a schedule. The individual is represented by a tuple (AL, SK) of the activity list AL and the shift key vector SK. The activity list $AL = (\pi_1, \pi_2, \dots, \pi_i, \dots, \pi_{subn})$ is a permutation of the sub-activities, where π_i is the sub-activity number that appears on the *i*th position. The number of elements in AL equals that of sub-activities $subn$. It should be noted that the order of the sub-activities in AL does not need to obey the precedence relations constraints, because we will deal with these constraints in the decoding procedure. There are also $subn$ elements in the shift key vector $SK = (sk_1, \dots sk_i, \dots, sk_{subn})$, $sk_i \in [0,1]$. sk_i indicates the degree to which the start time of sub-activity i deviates from its earliest start time. In other words, the earliest and latest start times of sub-activity i form a time window, and sk_i is the ratio of the difference between the start time of the sub-activity i and its earliest start time to the time window. In the initial population, AL is sampled from the initial probability model (Section 3.3.1), and SK is generated randomly.

In the GEDA, the resource levelling schedule generation scheme (RLSGS) (Li et al. 2018; Li & Dong 2018) is used to decode individuals into schedules. The main process of the RLSGS is as follows. For an individual (AL, SK), the first unscheduled sub-activity i indicated by AL is selected and its start time is set as $s_i = es_i + \lfloor sk_i \times (ls_i - es_i) \rfloor$, where es_i (ls_i) is the earliest (latest) start time of sub-activity i after considering the scheduled activities. Then the earliest and latest start times of unscheduled sub-activities are updated. It can be seen that the start time assigned to each sub-activity in the above manner is always within a feasible time window, so that the precedence relationships between the sub-activities are satisfied. The above process is repeated until all sub-activities have been assigned a start time. In this way, the schedule is obtained. Given a schedule, we calculate its objective function value according to Equation (1), which is then used to evaluate each individual.

3.3 Operators on the activity list

In each iteration of the GEDA, the activity list AL is the updated using the probability models.

3.3.1 Probability-generating mechanism

In the probability-generating mechanism (PGM), the probability model

$PM = (\alpha_1, \alpha_2, \dots, \alpha_i, \dots, \alpha_{subn})$ predicts the probability α_i that sub-activity i is chosen when constructing the sub-activity list AL. The basic idea of the PGM is as follows. The numbers in the interval $(0, \sum_{i=1}^{subn} \alpha_i)$ correspond to different sub-activities. The sub-activity list can be obtained by sampling from this interval according to the probability $(\alpha_1, \alpha_2, \dots, \alpha_i, \dots, \alpha_{subn})$ of each sub-activity in PM. Details on the PGM is shown in Algorithm 1.

Algorithm 1. Using the PGS to generate the sub-activity list

```
Input: the probability model $PM$
Output: the sub-activity list $AL$
$AL = \emptyset$;
$U = \sum_{i=1}^{subn} \alpha_i$;
For $pos = 1$ to $subn$
    $\pi = 1$;
    $Pr = \alpha_\pi$;
    Sample a random number $r$ from the interval $(0, U)$
    While $r > Pr$: $\pi = \pi + 1$, $Pr = Pr + \alpha_\pi$;
    $\pi_{pos} = \pi$;
    $U = U - \alpha_\pi$;
    $\alpha_\pi = 0$;
End for
Return $AL = (\pi_1, \pi_2, \dots, \pi_{subn})$
```

In Algorithm 1, the initial sub-activity list AL is set to empty. The sub-activity number π needs to be inserted into AL, and the order that the sub-activities are selected is determined by the probability model PM. π_{pos} represents the sub-activity number in AL and pos is the pos-th position in AL ($pos = 1,2, \dots, subn$). Whenever a sub-activity is put into AL, the corresponding element in PM is set to 0 (i.e. $\alpha_i = 0$). In doing so, we ensure that each sub-activity is only selected once, which means that after a sub-activity is selected, it will not be selected again. Given a set $\overline{N'}$ of sub-activities that have not been added to the sub-activity list AL, the probability that the sub-activity $i \in \overline{N'}$ is selected into AL is $\alpha_i / \sum_{j \in \overline{N'}} \alpha_j$.

At the beginning of the GEDA, we initiate $PM = (1/subn, 1/subn, \dots, 1/subn)$ such that each sub-activity is chosen with an equal probability. This means that we sample uniformly from the solution space to form the initial sub-activity list. In this way, the diversity of the initial solutions in the solution space is guaranteed.

3.3.2 Probability model updating mechanism

At the end of each iteration, the probability model PM needs to be updated such that PM can better estimate the distribution of the solutions. Specifically,

POP' best individuals are selected from the POP individuals generated using the PGM in terms of the objective function value. Then the elements in PM are updated based on the following formula:

$$\alpha_i' = (1-\beta)\cdot\alpha_i + \frac{\beta}{POP'}\sum_{j=1}^{POP'} \omega_i^j\,,\ 1 \le i \le subn \tag{8}$$

where α_i' is the updated value of α_i, and β is the learning speed. ω_i^j reflects the importance of sub-activity i appearing in different positions of the sub-activity list. When constructing a sub-activity list using the PGM, it is implied that the more front the activity in the sub-activity list, the more important it is. Therefore, to calculate ω_i^j, we let the sub-activity that appears first in the sub-activity list has the largest weight $subn$, the weight of the subsequent sub-activity decreases by 1, and so on. So the weight of the last sub-activity in the sub-activity list is 1. Consequently, $\omega_i^j = \frac{subn-pos+1}{subn+(subn-1)+\cdots+1} = \frac{subn-pos+1}{(subn+1)subn/2}$, where pos is the position of sub-activity i in AL.

3.4 Operators on the shift key vector

The crossover and mutation operators from the GA are applied to the shift key vector to update it.

3.4.1 Crossover

Two-point crossover is adopted in the GEDA. First, $POP/2$ individuals are chosen as father individuals with probability P_c; and top $POP/2$ individuals in terms of the objective function value are selected as mother individuals. Then, the father and mother individuals are randomly matched into $POP/2$ pairs of individuals. Next, two crossover points t_1 and t_2 $(1 \le t_1 < t_2 \le subn)$ are selected randomly. The father's (mother's) gene between t_1 and t_2 are copied to son (daughter) individual. The remaining gene positions in the son (daughter) individual are filled with the genes on the corresponding positions of the mother (father) individual.

3.4.2 Mutation

The mutation operator used in the GEDA is one-point mutation. For each child individual, each element in the shift key vector has a mutation probability of P_m. Specifically, for each element sk_i $(i = 1, \dots, subn)$ in the shift key vector, we generate a random number $rand \in (0,1)$. If $rand$ is smaller than the mutation probability P_m, then we replace sk_i with a new random number between 0 and 1.

4. Computational experiments

Our GEDA is implemented in Matlab R2014a. Our computational experiments are performed on a computer equipped with an Intel Core i5 2.5 GHz

CPU and Windows 10.

4.1 Experimental setup

Our computational experiments adopt two benchmark data sets: T_{A} and T_{B}. T_{A} is from Liu et al. (2019). Both sets are generated using the project scheduling problem instance generator RanGen (Demeulemeester et al. 2003). Specifying various control parameters in RanGen, project networks with different number of activities, topological structures and resource types can be produced. The main parameters of the data sets T_{A} and T_{B} are shown in Table 1. The order strength (OS) is the ratio of the number of precedence relations in the project network to the theoretical maximum number of precedence relations. The OS represents the complexity of the project network structure. ES_{n+1} is the critical path length computed by the critical path method.

Table 1. The main parameters of data sets T_{A} and T_{B}

Parameter	Dataset T_{A}	Dataset T_{B}
$\lvert N\rvert$	10, 20, 30	100
OS	0.3, 0.5, 0.7	0.3, 0.5, 0.7
K	4	4
$\bar{d}$	$1.0 \cdot ES_{n+1}$; $1.2 \cdot ES_{n+1}$	$1.0 \cdot ES_{n+1}$; $1.2 \cdot ES_{n+1}$

Given the parameter values of T_{A} in Table 1, 90 instances are generated for each value combination and this results in $3 \times 3 \times 1 \times 2 \times 90 = 1620$ instances in T_{A}. In terms of the number of activities contained in each instance, T_{A} is a small scale data set.

The main difference between T_{A} and T_{B} is the number of activities. T_{B} is a larger data set with 100 activities in each instance. Given the parameter values of T_{B} in Table 1, 90 instances are also generated for each value combination and this results in $1 \times 3 \times 1 \times 2 \times 90 = 540$ instances in T_{B}.

For the GEDA parameters, after fine-tuning the parameters, we set the population size $\mathrm{POP} = 200$, the number of individuals used to update the probability model $\mathrm{POP}' = 50$, the learning speed $\beta = 0.5$, the crossover probability $P_c = 0.8$ and the mutation probability $P_m = 0.3$. The termination condition of the GEDA is to generate up to 1000 schedules.

4.2. Performance measures

To evaluate the performance of our GEDA, we take the mixed-integer linear programming algorithm in CPLEX as the baseline algorithm and compare it with the GEDA. Specifically, we transform the model (1) - (7) into a mixed-integer linear programming model according to Liu et al. (2019). Then the resulting model is solved using CPLEX. For CPLEX, the time limit for each instance is set to 600 seconds. This means that an optimal solution will be output if it can be found within 600 seconds. Otherwise, the best feasible solution is output.

In our experiments, the following performance measures are employed:

(1) Average relative deviation (ARD): The average percentage deviations from the objective values obtained in CPLEX. The ARD is calculated as follows:

$$\text{ARD} = \frac{\sum_{i=1}^{n}\left[\left(O_i^{GEDA} - O_i^{CPLEX}\right)/O_i^{CPLEX}\right]}{n} \times 100\% \qquad (9)$$

where O_i^{GEDA} (O_i^{CPLEX}) is the objective function value of the *i*th instance obtained by the GEDA (CPLEX) and *n* is the number of instances in the test set. The value of the ARD reflects the performance difference between the GEDA and CPLEX. A smaller ARD value means that the GEDA can obtain better solutions.

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

4.3. Computational results

We examine the performance of the GEDA by comparing it with CPLEX. Table 2 shows the computational results on the small-scale test set T_A. It should be noted that CPLEX only obtains the optimal solutions of 358 instances in T_A. The remaining instances have only feasible solutions so far (Liu et al., 2019). Accordingly, the results in Table 2 are divided into two groups corresponding to the columns labeled with "Instances with known optimal solutions" and "Instances with feasible solutions only", respectively. The cells filled with "-" mean that the optimal solutions of the corresponding instances have not been found. A negative value of the ARD indicates that our GEDA outperforms CPLEX in the corresponding condition.

From Table 2, it can be seen that for instances with known optimal solutions, the solutions obtained by the GEDA are close to the optimal ones. For instances with feasible solutions, the GEDA obtains better solutions in more than half of the cases (at this time, the ARD is negative); in the other cases, the results of the GEDA are close to CPLEX (the ARD is less than 2%). In terms of the computation time, the average CPU time of the GEDA is less than 0.6 seconds, which is much faster than CPLEX. In summary, the results in Table 2 show that for the small-scale PRLP instances, the GEDA is able to obtain satisfactory solutions in a short time.

Because the scale of the data set T_B is larger, CPLEX has not found the optimal solutions. The ARDs in Table 3 indicate the comparison results between the solutions obtained by the GEDA and the best solutions of CPLEX. We can see from Table 3 that the GEDA obtains better solutions when the project deadline is loose. For the tight project deadline, the solutions of the GEDA are only slightly worse than CPLEX, and the difference between objective values is within 2%. In addition, the average CPU time of the GEDA is within 6 seconds, whose efficiency is much higher than CPLEX. In short, when solving large-scale instances, the effectiveness and efficiency of our GEDA are also satisfactory.

Table 2. Computational results on the data set T_A

$\bar{d}$	$\|N\|$	OS	Instances with known optimal solutions		Instances with feasible solutions only	
			ARD	CPU (s)	ARD	CPU (s)
$1.0 \cdot ES_{n+1}$	10	0.3	2.39%	0.128	-0.71%	0.132
		0.5	4.47%	0.117	-2.37%	0.123
		0.7	2.93%	0.122	-2.22%	0.123
	20	0.3	2.08%	0.304	0.41%	0.317
		0.5	3.04%	0.312	-0.29%	0.310
		0.7	1.60%	0.298	-0.29%	0.313
	30	0.3	0.55%	0.562	0.31%	0.592
		0.5	1.01%	0.550	0.87%	0.590
		0.7	1.01%	0.542	0.89%	0.587
$1.2 \cdot ES_{n+1}$	10	0.3	3.37%	0.125	0.23%	0.133
		0.5	5.05%	0.119	-1.68%	0.124
		0.7	5.97%	0.122	-1.98%	0.125
	20	0.3	2.79%	0.303	0.08%	0.315
		0.5	10.54%	0.293	-0.12%	0.312
		0.7	4.63%	0.292	-0.03%	0.320
	30	0.3	1.64%	0.584	-0.29%	0.597
		0.5	3.81%	0.537	1.05%	0.593
		0.7	-	-	1.44%	0.593

Table 3. Computational results on the data set T_B

$\bar{d}$	OS	ARD	CPU (s)
$1.0 \cdot ES_{n+1}$	0.3	0.64%	5.754
	0.5	0.58%	5.711
	0.7	1.94%	4.242
$1.2 \cdot ES_{n+1}$	0.3	-3.11%	5.783
	0.5	-8.54%	5.750
	0.7	-11.48%	4.237

4.4. Sensitivity analysis

In this subsection, we examine the impact of the number of activities, the project deadline and the OS on the performance of the GEDA (Figures 6-9). In Figures 6-9, the lower the line, the better the results.

Figures 6-8 display the results on the small-scale test set T_A. It can be seen that there are no obvious patterns for the impact of the number of activities. When other factors are given, the GEDA can get better results for instances with 10 and 30 activities. For the project deadline, a looser deadline leads to better solutions. When the number of activities is small ($|N| = 10$), the impact of the OS is not obvious; as this factor becomes larger ($|N| = 20{,}30$), the GEDA achieves the best performance when the OS is the smallest.

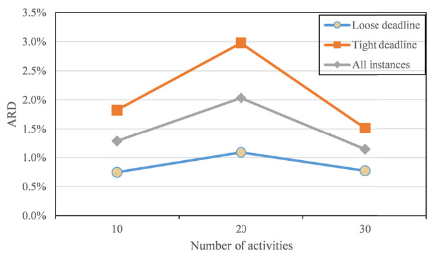


Figure 6. Impact of the number of activities and the project deadline (set T_A)

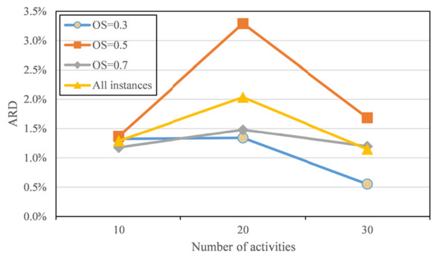


Figure 7. Impact of the number of activities and the OS (set T_A)

Figure 9 shows the results on the large-scale test set T_B. Different from the results on the small-scale test set, given the OS, the tighter the project deadline, the better the performance of the GEDA. The impact of the OS on the GEDA is dependent on the project deadline: When the deadline is loose, the impact of the OS is not obvious; when the deadline is tight, the performance of the GEDA improves as the OS increases.

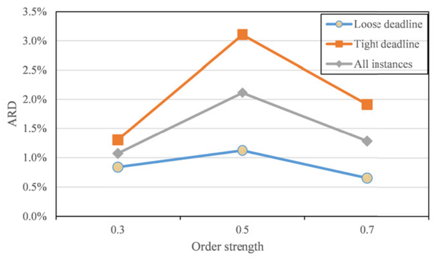


Figure 8. Impact of the OS and the project deadline (set T_A)

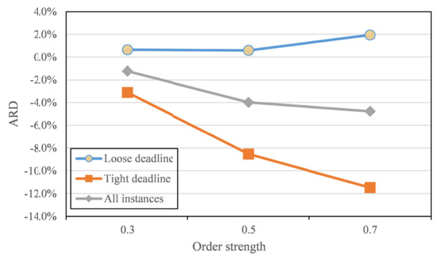


Figure 9. Impact of the OS and the project deadline (set T_B)

4.5. Comparison with existing meta-heuristic algorithm

Prior to our GEDA, no meta-heuristic algorithm has been found for the specific PRLP studied in this paper. Therefore, it is difficult to directly compare the GEDA with the existing meta-heuristic algorithm. However, in the existing literature, there are meta-heuristic algorithms for solving other types of PRLP. Among them, the HGA of Doulabi et al. (2011) is currently the best performing meta-heuristic algorithm. There are some differences between our PRLP and the problem of Doulabi et al. (2011), such as they consider preemption costs, allow only a part of activities to be interrupted, and the activities require different amounts of resources at different times. Therefore, our GEDA can only be indirectly compared with the HGA in this paper. The results are shown in Table 4.

Table 4. Comparison results between our GEDA and the HGA of Doulabi et al. 2011

$\|N\|$	GEDA		HGA	
	ARD	CPU (2.5GHz)	ARD	CPU (3.0GHz)
10	4%	0.122	0%	18
20	2%	0.305	6%	41
30	1%	0.551	4%	118

Since the results in Table 4 are indirect comparison results, some explanations need to be made before explaining these results: (1) The instances solved by the GEDA are those with known optimal solutions in T_{A}. The data set used by the HGA is generated by PROGEN/MAX (Schwindt 1995). The project deadlines in both data sets are the same and both of them equal $1.0 \cdot ES_{n+1}$. (2) The number of schedules produced by the GEDA is 1000 while that for the HGA is 2500. (3) As mentioned earlier, there are some differences between the problems solved by the GEDA and the HGA.

Although there are some differences in the experimental environments of the GEDA and the HGA, both algorithms are evaluated in terms of the average deviations from the optimal objective function value (i.e., the ARDs). Therefore, according to the ARDs, the GEDA and the HGA can still be compared to a certain extent.

We observe from Table 4 that when the number of activities exceeds 10, the computational results of the GEDA are better than the HGA; in other words, the solutions obtained by the GEDA is closer to the optimal ones. A possible explanation would be that the encoding method used in the HGA has a probability of producing infeasible solutions during the iteration process, which may reduce the optimization efficiency of the HGA. While in the GEDA, our encoding method can ensure that the solutions generated in each iteration are always feasible. In addition, the CPU frequency used by the GEDA is lower than the HGA, but the running time of the GEDA is far less than the HGA. In summary, the results in Table 4 indirectly indicate our GEDA outperforms the HGA in terms of the solution effectiveness and efficiency.

5 Conclusions and future research

We have proposed an effective and efficient meta-heuristic algorithm, GEDA, for the PRLP. In the PRLP, each activity is allowed to be interrupted at any integer time point. Prior to this paper, there has been no meta-heuristics for this type of resource balancing problems. In the proposed GEDA, a schedule is encoded as an individual consisting of an activity list and a shift key vector. Our encoding and decoding methods ensure that the generated schedule is always feasible. Considering the characteristics of the RPLP, several specially designed operators are also integrated into the GEDA, e.g., the probability models, the probability-generating mechanism, the probability updating mechanism, the crossover and mutation operators.

Based on a large number of benchmark instances, the performance of the GEDA is analyzed through extensive computational experiments. The experimental results show that the GEDA is able to find satisfactory solutions within a reasonable time. For the instances with known optimal solutions, the gap between the solutions obtained by the GEDA and the optimal solution is within 5% in most cases. For the remaining instances, the solutions obtained by the GEDA are better than or close to CPLEX, while the calculation time of the GEDA is only about 1/1000 of CPLEX. In addition, the comparative experimental results reveal that the proposed GEDA outperforms the existing meta-heuristic algorithm.

It will be an important research direction to design more effective meta-heuristics for the PRLP. Considering uncertainties in the PRLP will also be an interesting topic.

ACKNOWLEDGEMENTS

This work was supported by the National Natural Science Foundation of China (Grant Number 71602106) and the Key Soft Science Project of Shanghai Science and Technology Innovation Action Plan (Grant Number 20692192400).

REFERENCES

[1] **Alsayegh, H. & Hariga, M. (2012),** ***Hybrid Meta-heuristic Methods for the Multi-resource Leveling Problem with Activity Splittin***g. *Automation in Construction,* 27(1), 89-98;

[2] **Ballestín, F., Valls, V. & Quintanilla, S. (2008),** ***Pre-Emption in Resource-constrained Project Scheduling.*** *European Journal of Operational Research,* 189(3), 1136-1152;

[3] **Ballestín, F. Valls, V. & Quintanilla, S. (2009),** ***Scheduling Projects with Limited Number of Preemptions.*** *Computers & Operations Research*, 36(11), 2913-2925;

[4] **Bock, D. B. & Patterson, J. H. (1990),** ***A Comparison of Due Date Setting, Resource Assignment, and Job Preemption Heuristics for the Multiproject Scheduling Problem.*** *Decision Sciences,* 21(2), 387-402;

[5] **Demeulemeester, E. L. & Herroelen, W. S. (1996),** ***An Efficient Optimal Solution Procedure for the Preemptive Resource-constrained Project Scheduling Problem.*** *European Journal of Operational Research,* 90(2), 334-348;

[6] **Doulabi, S., Seifi, A. & Shariat, S. Y. (2011),** ***Efficient Hybrid Genetic Algorithm for Resource Leveling via Activity Splitting.*** *Journal of Construction Engineering and Management,* 137(2), 137-146;

[7] **Forghani, K. & Fatemi-Ghomi, S. T. (2019),** ***Cellular Manufacturing Scheduling in the Presence of Multiple Process Routings and Considering Job Splitting***. *Economic Computation and Economic Cybernetics Studies and Research,* 53(2), 271-288; *ASE Publishing;*

[8] **Hariga, M. & El-Sayegh, S. M. (2011),** ***Cost Optimization Model for the Multi-resource Leveling Problem with Allowed Activity Splitting***. *Journal of Construction Engineering and Management,* 137(1), 56-64;

[9] **Li, H. & Dong, X. (2018),** ***Multi-mode Resource Leveling in Projects with Mode-dependent Generalized Precedence Relations***. *Expert Systems with Applications,* 97, 193-204;

[10] **Li, H., Xiong, L., Liu, Y., & Li, H. (2018),** ***An Effective Genetic Algorithm for the Resource Leveling Problem with Generalised Precedence Relations.*** *International Journal of Production Research,* 56(5), 2054-2075;

[11] **Liu, Y., Hu, Z., Li, H. & Zhu, H. (2019),** ***Does Preemption Lead to More Leveled Resource Usage in Projects? A Computational Study Based on Mixed-Integer Linear Programming.*** *Economic Computation and Economic Cybernetics Studies and Research,* 53(4), 243-258; *ASE Publishing;*

[12] **Nadjafi, B., Khalaj, Z. & Mehdizadeh, E. (2013),** ***A Branch and Bound Approach to Solve the Preemptive Resource Leveling Problem***. *International Journal of Manufacturing Engineering,* (1), 24-26;

[13] **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;*

[14] **Nudtasomboon, N. & Randhawa, S. U. (1997),** ***Resource-constrained Project Scheduling with Renewable and Non-renewable Resources and Time-resource Tradeoffs.*** *Computers & Industrial Engineering,* 32(1), 227-242;

[15] **Razavi, N. & Mozayani, N. (2007),** ***A Resource Leveling Model Based On Genetic Algorithms: Activity Splitting Allowed.*** Proceedings of the International MultiConference of Engineers and Computer Scientists 2007, IMECS 2007, March 21-23, 2007, Hong Kong, China. DBLP;

[16] **Rieck, J. & Gather, T. (2012),** ***Mixed-integer Linear Programming for Resource Leveling Problems.*** *European Journal of Operational Research,* 221(1), 27-37;

[17] **Schwindt, C. (1995), ProGen/max:** ***A New Problem Generator for Different Resource-constrained Project Scheduling Problems with Minimal and Maximal Time Lags.*** Institute for Economic Theory and Operations Research 449, Universitat Karlsruhe, Karlsruhe, Germany;

[18] **Son, J. & Mattila, K. G. (2004),** ***Binary Resource Leveling Model: Activity Splitting Allowed.*** *Journal of Construction Engineering and Management,* 130(6), 887-894.