**Resource leveling in projects with flexible structures**

Hongbo Li, Linwen Zheng\*, Hanyu Zhu

School of Management, Shanghai University, Shanghai 200444, China

E-mail addresses: ishongboli@gmail.com, hongbo\_li@shu.edu.cn (H. Li); linwenzheng@126.com (L. Zheng); zhuhanyu1101@qq.com (H. Zhu).

\* Corresponding author.

**Abstract**: Project resource leveling, which is usually performed in the project planning phase, aims to minimize fluctuations in resource usage such that the project success probability is enhanced. The existing studies on resource leveling mainly assume a fixed project structure, i.e., the activities and precedence relationships are given in advance. However, in real-world projects, the project structure is usually flexible, i.e., not all activities are implemented, and the precedence relationships exist only between the activities that are implemented. Aiming at providing support for project managers to make effective resource leveling decisions when facing flexible project structures, we study the resource leveling problem with flexible structures (RLP-PS). We present a nonlinear integer programming model for the RLP-PS and a linearization method to transform the nonlinear model into a linear model. To efficiently solve the RLP-PS, we devise two problem-specific algorithms: a two-stage heuristic algorithm and a customized genetic algorithm. Based on the PSPLIB benchmark dataset, extensive computational experiments are performed to analyze the performances of our algorithms. The experimental results reveal the effectiveness and competitiveness of our algorithms.

**Keywords**: Project scheduling; Resource leveling; Flexible projects; Genetic algorithm

# Introduction

In project management, the allocation of renewable resources heavily impacts the execution of the project. Unleveled resource usage tends to lead to "peaks" or "valleys" of resources. Fluctuations in resource usage are very undesirable for project managers because they may cause the hasty deployment of resources and result in financial strains (Son & Skibniewski, 1999; Li et al., 2020). Therefore, effective project resource leveling techniques are indispensable. The resource leveling problem (RLP), which is one of the classic project scheduling problems, aims to minimize the fluctuations in resource utilization by constructing a baseline schedule that is subject to precedence relationship constraints and the project deadline constraint (Li & Demeulemeester, 2016; Li et al., 2018).

In the past few decades, the RLP has been extensively studied, and a large number of exact, heuristic and metaheuristic algorithms have been proposed. Typical exact algorithms include dynamic programming (Bandelloni et al., 1994), branch-and-bound procedures (Neumann & Zimmermann, 2000; Gather et al., 2011; Ponz-Tienda et al., 2017) and integer programming (Rieck et al., 2012). Since the RLP is NP-hard (Neumann et al., 2012), facing large-scale instances, it is usually difficult for exact algorithms to find satisfactory solutions within a reasonable amount of time. In this case, we can rely on heuristic and metaheuristic algorithms. Most of the heuristic algorithms designed for solving the RLP employ the idea of shifting activities within their slacks (Burgess & Killebrew, 1962; Neumann & Zimmermann, 1999; Christodoulou et al., 2010). Compared with heuristic algorithms, metaheuristic algorithms are usually more effective. Many metaheuristic algorithms, such as genetic algorithms (Li et al., 2018), tabu search (Koulinas & Anagnostopoulos, 2013), and iterated greedy algorithms (Ballestín et al., 2007), have been devised to solve the RLP.

The existing studies on project management and scheduling mainly assume that the project structure is fixed (Davari & Demeulemeester, 2019; Caramia, 2020), i.e., all activities must be executed, and all precedence relationships must be satisfied. This is also the case in the RLP area. However, as the project becomes more complex and the technologies or methods used become more diverse, the above assumptions are not consistent with the actual situation. For example, the installation process of the window can be completed by scaffolding or scissor lift with different processes and resources (Tao et al., 2018). In the product manufacturing process, some flexible parts have multiple processing routes to choose (Birjandi et al., 2019). Therefore, a project can be accomplished in a number of ways in a realistic scenario, the project structure tends to be flexible. In a project with flexible structures, not all activities are implemented, and the precedence relationships exist only between the activities that are implemented. Different methods correspond to different activities and precedence relationships, as well as different resource requirements. Taking a bridge construction project as an example, reinforced

In recent years, more and more researches on the combination of flexible structure and resource-constrained project scheduling, which aim to find a project structure and minimize the project makespan (Kellenbrink & Helber, 2015; Servranckx et al., 2021; Tao & Dong, 2017). In practice, most construction projects have a completion date. Since the completion time of the project has been fixed, it is not necessary to consider optimizing the completion time of the project. The project manager pays more attention to how to ensure the smooth completion of the project on the specified date. Within a fixed date, highly fluctuating resource consumption may cause financial difficulties, increase project risks, and lead to low project efficiency and high costs (Kazemi & Davari-Ardakani, 2020; Son & Mattila, 2004).

For flexible construction projects with fixed deadline, reducing fluctuations in resource usage is conductive to the realization of project goals. On the one hand, choosing different methods will affect resource requirements in the process of project execution. Therefore, choosing and scheduling activities in a way that controls fluctuations in resource consumption can help contractors control project costs and increase project profits. On the other hand, in the process of resource leveling, various factors affecting the effect of resource leveling can be considered, such as multi-mode (Li & Dong, 2018), preemption (Alsayegh & Hariga, 2012; Hariga & El-Sayegh, 2011; Karaa & Nasr, 1986), etc. Taking these factors into account can further improve the use of resources and achieve better resource leveling. In addition to the above factors, the flexible structure is also the key to level resources. Resource leveling is usually achieved by shifting noncritical activities within their available floats. When activities are carried out in different periods, the demand for the same resource can be alleviated. The greater the activity’s float, the more conducive it is to adjust the start time of the activity and make the use of resources more stable. In the fixed project structure, the floating time of activities is fixed, and resource leveling optimization can only be carried out based on these floating times. However, in the flexible structure, the implementation activities and precedence relationship in the project are not fixed, which affect the activity’s float. Therefore, the flexible structure can provide new possibilities for the effective use of resources. Based on these, we combine flexible structure with resource leveling to improve the resource utilization of flexible project and improve the decision-making in the scheduling process.

Although flexible or unfixed project structures have been investigated in some project scheduling problems (Beck & Fox, 2000; Barták et al., 2007), to the best of our knowledge, no research has considered flexible structures in project resource leveling. The existing studies on project scheduling with flexible or unfixed project structures can be categorized into two groups. One group is to increase the flexibility of project scheduling by extending the logical relationships between activities. In addition to the traditional AND relationship, some scholars also consider the existence of OR, EXCLUSIVE OR and bidirectional (BI) relationships between activities (Belhe & Kusiak, 1995; Gillies & Liu, 1995; Vanhoucke & Coelho, 2016). Gillies and Liu (1995) further studied the complexity of the resource-constrained project scheduling problem (RCPSP) with generalized AND/OR constraints. Although these studies extend the logical relationships, all activities in the project still need to be implemented. The other group considers flexible project structures. Compared with that of the first group of studies, the project structure in this group is more flexible, i.e., implementing some activities is optional, and the precedence relationships appear with the implemented activities. Čapek et al. (2012) proposed the RCPSP with alternative process plans based on actual wire harness production. Kellenbrink and Helber (2015) extended the RCPSP with a model-endogenous decision on the flexible project structure. Their scheduling problem involves deciding whether to implement specific activities and schedule them. Tao and Dong (2017) combined the alternative activity chain with the RCPSP (RCPSP-AC). They proposed an AND-OR project network to represent the RCPSP-AC and devised a simulated annealing algorithm for the problem. Servranckx and Vanhoucke (2019) extended the RCPSP with optional subgraphs and developed a tabu search algorithm. In addition to the above two groups of studies, Benjaoran et al. (2015) studied the RLP with precedence relationship options. In their research, activities could have one or more optional relationships with other activities. Choosing the type of optional relationship may affect the activity’s float and provide additional flexibility to level the resource requirements. Jaskowski and Biruk (2018) considered the soft precedence relationship in the resource leveling problem, allowing activities to be executed in reverse order or in parallel. They use an example to illustrate that the soft precedence relationship makes the activities have greater float and can obtain schedules with higher resource utilization. Although these studies increase the flexibility of resource leveling by considering changing the type of precedence relationship, the project structure was not flexible, and all activities needed to be implemented.

To fill the research gap in which flexible structures are not considered in the current studies on project resource leveling, we propose the resource leveling problem with flexible structures (RLP-PS). Our main contributions are as follows:

(1) We extend the classic RLP by incorporating the flexible project structure. For the RLP-PS, a nonlinear integer programming model is formulated and linearized into an integer linear programming model.

(2) For the RLP-PS, we design a two-stage heuristic algorithm (TSHA) and a customized genetic algorithm (CGA) from the perspectives of problem decomposition and integration, respectively. Our TSHA is composed of several priority rules and can quickly find feasible solutions. In our CGA, we devise a special schedule encoding/decoding process, a crossover operator and a mutation operator according to the characteristics of the RLP-PS. In addition, a local improvement method is integrated into the CGA to further improve the obtained schedule.

(3) Based on the PSPLIB benchmark dataset, extensive computational experiments are performed to analyze the performances of our algorithms. We use the Taguchi method for the design of the experiment (DOE) to determine the suitable parameter settings for our CGA. In addition, the impacts of various factors on the performance of our CGA are analyzed.

The remainder of this paper is organized as follows: In Section 2, we describe the RLP-PS and present the corresponding nonlinear and linear integer programming models. A TSHA and a CGA are devised in Section 3. Section 4 presents our computational experiments. The last section concludes the paper and discusses future research.

# The RLP-PS

## Problem description

A project is represented by an activity-on-node network. The set of nodes represents the activities, where . Activities 0 and are dummy activities, indicating the start and end of the project, respectively. The set of directed arcs denotes the finish-start precedence relationships with zero-time-lag, which means that each activity cannot be started until all its predecessors have been completed. When , activity is the predecessor of activity , and activity is the successor of activity . Each non-dummy activity has a duration . The duration of the dummy activity is 0. The start time of activity is represented by . The start time of the dummy end activity corresponds to the completion time of the project. The project deadline is . There are types of renewable resources in the project. When non-dummy activity is executed, the requirement for renewable resource type in the unit period is . Dummy activities do not consume any resources. During each time period , the total usage of resource type is denoted by , where is the set of activities that are being executed during time period (). The activities cannot be interrupted during execution, i.e., preemption is not allowed. The above-mentioned parameters are all integers.

In a project with flexible structures, some activities are optional (alternative) and some activities are dependent on the implementation of other activities. This means that not all activities need to be implemented and the precedence relationships only exist between activities that are actually implemented. This leads to many different feasible structures for a flexible project. Performing the project with any feasible structure is able to achieve the planned objective of the project, however, different project structures have different effects on the flexibility of the construction schedule, some project structures can provide more floating time and are more likely to reduce fluctuations in resource usage.

To model the flexible project structure, following Kellenbrink and Helber (2015), we divide the activity set into three mutually exclusive subsets: the mandatory activity set , the optional activity set and the dependent activity set (,). The activities in the mandatory activity set are always implemented, but it is not the case for other activities.

The optional activity set may contain multiple mutually exclusive subsets , i.e., , . Each subset corresponds to a choice and consists of several optional activities. Each choice can be triggered by an activity . If activity is implemented, the corresponding choice is triggered, and only one of the optional activities in is implemented. If activity is not implemented, choice is not triggered, and all activities in are not implemented. Note that activity must precede the optional activity , and can be a mandatory activity, an optional activity or a dependent activity. The choice is topologically ordered such that for any pair of choices and (), choice must be triggered before choice .

The implementations of the activities in the dependent activity set are determined by the optional activities. An optional activity may have a set of dependent activities, . If optional activity is implemented, then all of its dependent activities in must be implemented. After we number all activities topologically, the number of optional activities is smaller than the number of activities in its dependent activity set , i.e., . It should be noted that there may not be precedence relationships between the dependent activities and optional activities.

The aim of the RLP-PS is to determine a project structure (i.e., selecting the activities to be implemented and determining the precedence relationships between activities) and construct a baseline schedule while satisfying the project deadline constraint such that the resource utilization is leveled as much as possible.

## Example

We illustrate the concept of the flexible project structure with the example of a bridge construction project. Bridge construction methods include the balanced cantilever method, formwork carriage and in-suit casting on standard falsework (Wu et al., 2010). Bridges are generally divided into superstructure and substructure. Fig. 1 shows the project network of the latter method for constructing the bridge substructure. The white (light gray, dark gray) circles represent mandatory (optional, dependent) activities. The ellipses indicate choices that contain optional activities. The arrow lines represent the precedence relationships. The precedence relationships take effect when the related activities are implemented and the solid arrow lines correspond to this type of precedence relationships. The dashed arrow lines correspond to the precedence relationships not taking effect.

Bridge construction starts from the substructure, i.e., “Construct Abutments” and “Construct Piers”. After the substructure is completed, the construction of the superstructure begins. “Construct Piers” triggers choice 1, i.e., = Construct Piers. Choice 1 is the method used to construct a pier, i.e., = {Reinforced Concrete, Steel Structure}. If the reinforced concrete is chosen, this causes all its dependent activities to be executed, i.e., = {Construct Subbasement and Basement, Construct Pier Shaft}. The execution of the “Construct Pier Shaft” activity further triggers choice 2, i.e., = Construct Pier Shaft; then, only one activity can be implemented in = {Precast, Cast-In-Situ}. If the reinforced concrete is selected in choice 1, the activities related to the steel structure are not carried out, and the precedence relationships corresponding to these activities are invalid. If the steel structure is selected, its dependent activities are performed, i.e., = {Weld}.

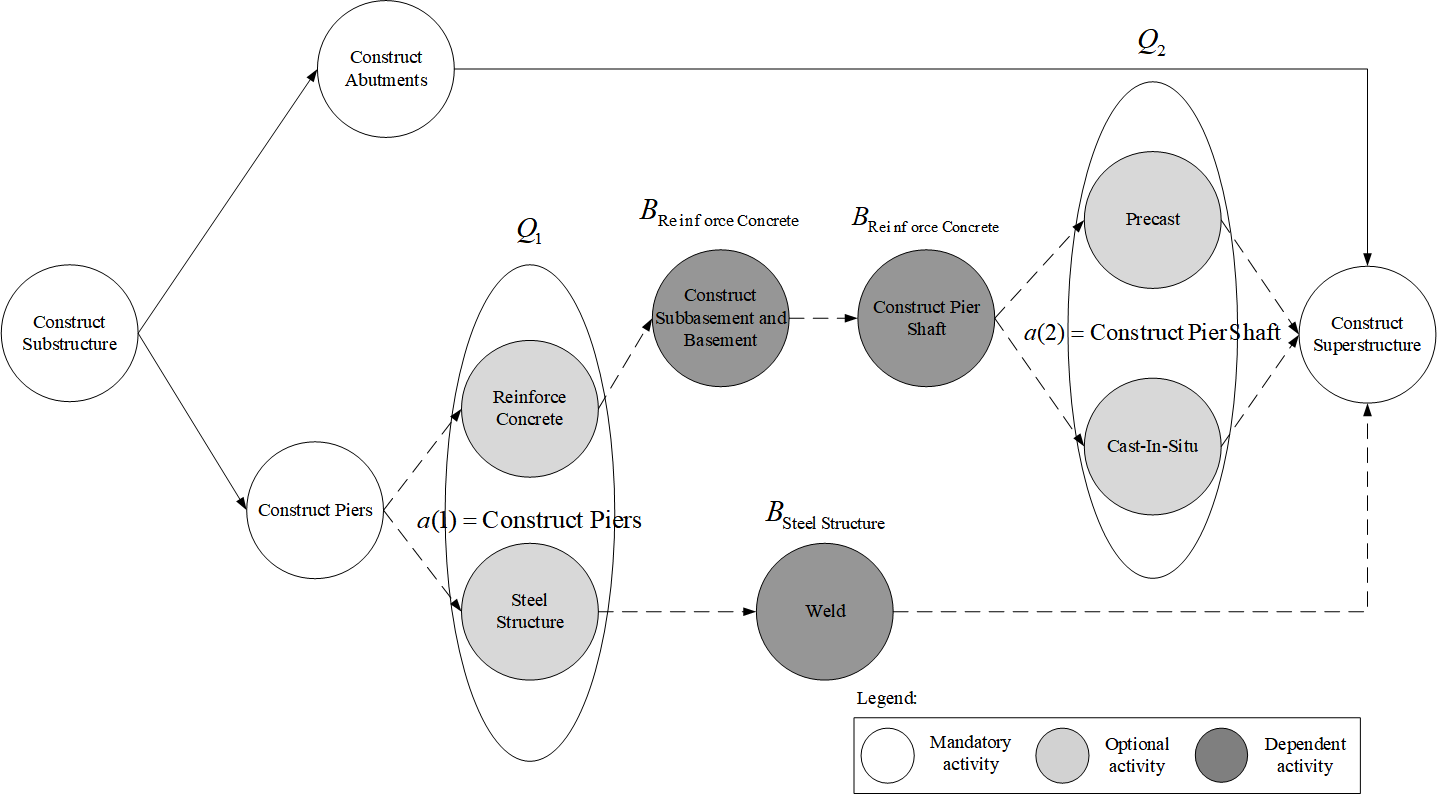


Fig. 1 Project network for bridge substructure construction

## Nonlinear integer programming model

In this section, the optimization model of the RLP-PS is presented. The binary decision variable is introduced to indicate whether activity starts at time . If activity starts at time , then ; otherwise, . At the same time, this variable also implicitly realizes the purpose of whether to execute certain activities, i.e., if activity is not executed at time , is equal to 0. The nonlinear integer programming model M0 of the RLP -PS is as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| (M0) | Minimize |  | (1) |
|  | Subject to: |  |  |
|  |  |  | (2) |
|  |  |  | (3) |
|  |  |  | (4) |
|  |  |  | (5) |
|  |  | ; ; | (6) |
|  |  | ; | (7) |
|  |  | ; | (8) |

In model M0, the objective function (1) is the weighted sum of squares of the resource usage per unit time, which measures the degree of the resource usage fluctuation. is the weight of resource type , representing the unit penalty cost of resource . This objective function is widely used in resource leveling (Demeulemeester & Herroelen, 2002). In addition, this objective function also implicitly minimizes the resource usage. Therefore, we do not explicitly consider resource constraints in our model. Constraint (2) states that all mandatory activities must be implemented, where and represent the earliest start time and latest start time of activity , respectively. Constraint (3) indicates that dummy activity 0 starts at time 1. Constraint (4) describes the precedence relationships between activities, where is a sufficiently large positive number. Only when activities and are executed is the constraint valid. Constraint (5) indicates that if choice is triggered by the implemented activity , then there must be an activity that is implemented. Constraint (6) guarantees that the execution of an optional activity triggers the execution of all its dependent activities. In other words, if optional activity in choice is implemented, then all activities in must also be implemented. Constraint (7) is used to calculate the resource usage for each time period. Constraint (8) provides the range of the decision variable .

## Model linearization

It can be seen that the objective function (1) in M0 is nonlinear. Next, we discuss how to transform the nonlinear integer programming model M0 into an equivalent linear integer programming model, such that it can be solved by commercial solvers.

**Proposition 1**: The nonlinear integer programming model M0 is equivalent to the linear integer programming model M1:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| (M1) | Minimize | | | (9) |
|  | Subject to: | |  |  |
|  | (2) - (6), (8) | |  |  |
|  |  | | ; | (10) |
|  |  | ; ; | | (11) |

**Proof**: Following Rieck et al. (2012), we introduce a binary variable to indicate whether the usage of resource at time reaches :

|  |  |
| --- | --- |
|  | (12) |

The auxiliary parameter ( represents the maximum usage of resource at time . We discretize the resource usage into [0, 1], [1, 2],..., []; then, the variable in the objective function (1) can be expressed by a binary variable as follows:

|  |  |
| --- | --- |
|  | (13) |

For each variable , the parameter is equal to the difference between and such that the objective function (1) can be linearized into equation (9). At the same time, according to equation (13), constraint (7) can be expressed as (10).

Since M1 is equivalent to M0, we will solve M1 in the rest of this paper. If all activities in the RLP-PS are mandatory, then the RLP-PS becomes a classic RLP. Therefore, the RLP is a special case of the RLP-PS. The RLP has been proven to be NP-hard (Neumann et al., 2012), so the RLP-PS is also NP-hard. To efficiently solve the NP-hard RLP-PS, we design a two-stage heuristic algorithm and a customized genetic algorithm in Section 3.

In the rest of this section, we further solve the problem corresponding to the example project in Section 2.2. We number the activities in the example project (Fig. 2). Table 1 shows the activity durations and resource requirements of the example project. We assume that there is only one resource type, i.e., . We let . The fixed per-period resource availability is 4. The project deadline is calculated based on the critical path method (Demeulemeester & Herroelen, 2002) by assuming that all activities are implemented.

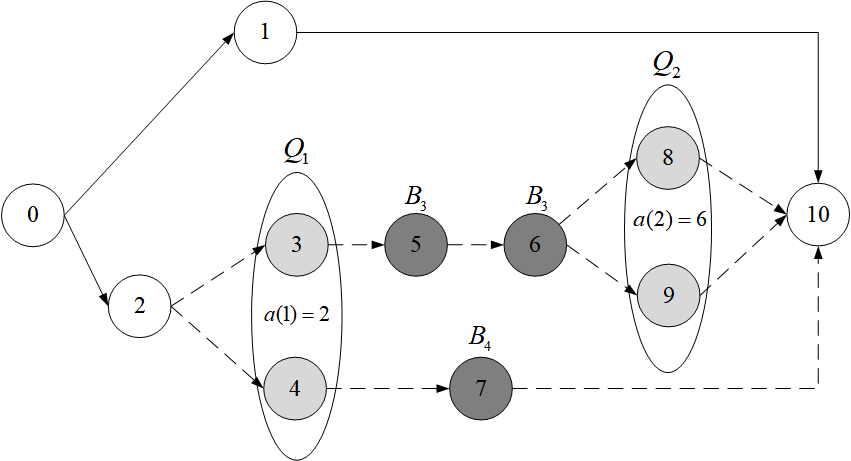


Fig. 2 Project network

Table 1. Activity durations and resource requirements in the example project

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  | 0 | 3 | 2 | 2 | 1 | 2 | 1 | 5 | 2 | 3 | 0 |
|  | 0 | 2 | 3 | 2 | 2 | 3 | 2 | 4 | 3 | 1 | 0 |

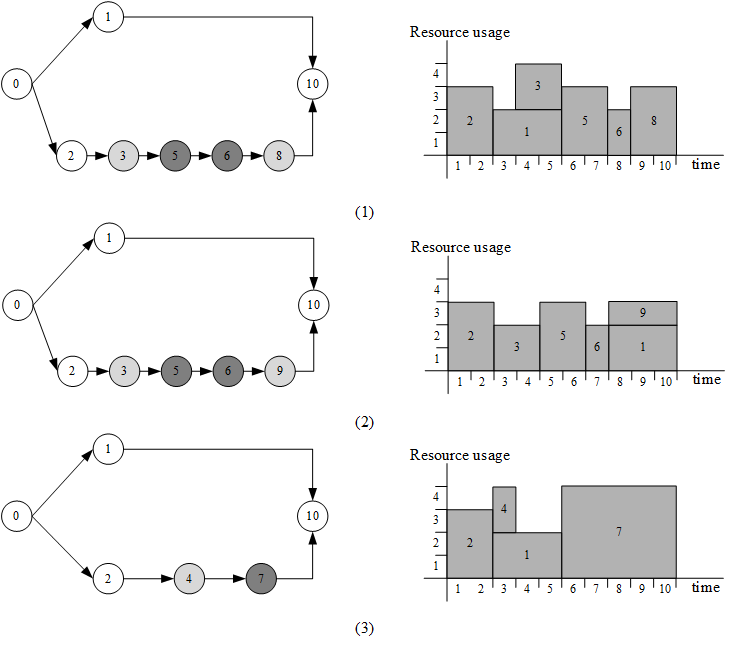


Fig. 3 The project network and corresponding optimal schedule of the example project

The example project has three different project structures and the corresponding set of implemented activities: = {0, 1, 2, 3, 5, 6, 8, 10}, = {0, 1, 2, 3, 5, 6, 9, 10}, = {0, 1, 2, 4, 7, 10}. These structures of the project and the corresponding optimal schedule are shown in Fig.3, the values of the objective function (10) of these optimal schedules are 94, 75 and 122 respectively. It can be seen from Fig.3 that different project structures correspond to different fluctuations of resource usage, which indicates that project structures have an impact on the effect of resource leveling. Some project structures are more conducive to adjusting activities to level resources. We use CPLEX to solve the linear integer programming model M1 of the example project. The project structure network and obtained optimal schedule are shown in Fig. 3(2).







In practice, the RLP is a basic scheduling problem in case that the project structure is given. When the project structure is flexible, in addition to resource leveling, different project managers may have different goals, such as project duration minimization. To meet other goals, our original single objective model M0 can be easily extended to deal with such situations. Specifically, for other objectives, i.e., , , we use -constraint method (Mavrotas, 2009) can convert these objectives functions into constraints, i.e., . Iteratively add these constraints to M0 and use varied well-chosen can lead to the Pareto optimal solution of the multi-objective problem.

# Customized genetic algorithm

The RLP-PS can be decomposed into two subproblems: the project structure selection problem (PSSP) and the RLP. The PSSP aims to determine the project structure by selecting a subset of activities. The objective of the RLP is to minimize the fluctuations in resource usage by assigning a start time for each activity. There are two basic strategies when solving the RLP-PS: integration and decomposition. The integration strategy solves the two subproblems simultaneously, while the decomposition strategy solves the two subproblems sequentially. Following the integration strategy, we design a customized GA. In addition, a two-stage heuristic algorithm (TSHA) is embedded into the CGA to produce initial populations. The TSHA is based on the decomposition strategy and it can also be used as an independent heuristic solution algorithm for the RLP-PS.

The classic GA is a population-based metaheuristic algorithm designed for solving hard optimization problems. It draws on genetic theory, simulates the evolution of a given population, and continuously optimizes the individuals in the population (Goldberg et al., 1989). GA has been successfully applied in solving RLPs (Chan et al., 1996; El-Rayes & Jun, 2009).

Our CGA encodes a schedule into an individual (Section 3.1). The initial population that consists of individuals is generated based on the TSHA (Section 3.2). During each iteration of the CGA, parent individuals are randomly selected from the population to perform the crossover operation for generating child individuals (Section 3.3). Then, the mutation operator is applied to the child individuals (Section 3.4). Subsequently, the decoding mechanism is used to convert the parent and child individuals into schedules (Section 3.1), and the objective function (1) corresponding to the schedule is used to evaluate the individuals. The best individuals among the parents and children serve as the new population in the next iteration. After the termination condition is satisfied, the CGA returns an optimized schedule. Finally, a local improvement method is used to further improve the obtained schedule (Section 3.5).

## Schedule encoding and decoding

The solution of the RLP-PS is a schedule. When metaheuristic algorithms are used to solve project scheduling problems, encoding the schedule usually makes the algorithms more efficient. Therefore, our CGA encodes the schedule as an individual. Each individual is a triple composed of a random key vector , a shift key vector and an implementation list vector : (1), where each of the elements denotes the priority value of activity . The higher the priority value of activity is, the earlier the activity is chosen for scheduling. (2) , where each element measures the deviation of activity from its earliest possible start time and is used to determine the start time of the activity during decoding (which is explained in more detail in the following decoding procedure). (3) . If activity is implemented, then ; otherwise, .

To convert the above individuals into a schedule, we design a decoding procedure, and its pseudocode is shown in Algorithm 1. In Algorithm 1, the scheduling order for the activities is determined by their s. The activity with the highest random key value that has not been scheduled is selected (Line 9). If activity needs to be implemented, we assign it a start time (Lines 10-12), where is the partial schedule formed by the scheduled activities, i.e., , and indicates the set of scheduled activities. denotes a possible project structure, which is determined by the (Line 3). If an activity is not implemented, no start time is assigned to it. When an activity has been assigned a start time, the earliest and latest start times of the unscheduled activities are updated (Lines 14-19). In Lines 16 and 17, represents the longest path length from activity to , provided that is reachable from . The path is a sequence of different activities. The length of path is . If activity is not reachable from activity , we set . After activity is assigned a start time , the possible earliest start time of an unscheduled activity is equal to the sum of and the longest path from activity to activity . The possible latest start time of activity equals the difference between and . The above steps are repeated until all activities to be implemented have been assigned start times.

|  |  |
| --- | --- |
| **Algorithm 1. Decoding procedure** | |
| 1 | **Input**: An individual () |
| 2 | **Output**: Schedule |
| 3 | Determine project structure (the implemented activities and the precedence relationships between implemented activities) according to ; |
| 4 | ; |
| 5 | ; |
| 6 | ; |
| 7 | For all ; |
| 8 | While , do |
| 9 | Select an activity with the highest random key form; |
| 10 | If |
| 11 | ; |
| 12 | ; |
| 13 | // Update the earliest start time and the latest start time of unscheduled |
|  | activities to be implemented |
| 14 | For all |
| 15 | If |
| 16 | ; |
| 17 | ; |
| 18 | End if |
| 19 | End for |
| 20 | End if |
| 21 | End while |

## Initial population generation based on the TSHA

The initial population of our CGA is generated based on the TSHA. In the following, we first describe the TSHA. The first stage of the TSHA solves the PSSP. In a project with flexible structures, choosing different activities for implementation results in different resource usages. If there are many implemented activities with relatively small resource requirements, then there may be much adjustment space when constructing the schedule, such that the total resource usage is highly leveled. Therefore, in this stage, for each triggered choice , we select activity to be implemented from the optional activity set according to the minimum resource requirement rule, i.e., . It can be seen that the resource requirement of each activity and its dependent activities in is taken as the priority value. When a choice is triggered, the activity with the smallest priority value in is selected for implementation. If multiple activities have the same priority value, the activity with the smallest duration is selected. If there is still a tie, the activity with the smallest activity number is selected.

In the flexible project structure, the implementations of optional activities and dependent activities are related. As long as the optional activities to be performed are determined according to the priority rule, the dependent activities to be performed are also executed. Then the precedence relationships between these activities are determined accordingly. This leads to a fixed project structure.

After the project structure is obtained, the RLP-PS becomes the classical RLP. Solving the RLP corresponds to the second stage of the TSHA. In this stage, a forward-backward heuristic algorithm based on a dynamic priority rule (He & Zhang, 2013) is employed to generate the baseline schedule. The algorithm consists of three modules: two backward iteration modules and one forward iteration module. These modules shift activities according to the forward total float, the forward total float and the backward free float of noncritical activities to level resource usages, respectively.

Based on the above-mentioned TSHA, the initial population of our CGA is generated as follows:

(1) Generation of POP implement list vectors. First, we use the first stage of the TSHA to determine the activities that need to be implemented and the corresponding implement list vector . Then, the mutation operator is applied to (Section 3.5) POP-1 times to generate the rest implement list vectors. Note that when this mutation operator is used here, we assume a mutation probability of 1.

(2) Generation of POP random key vectors. We first use the second stage of the TSHA to obtain start times for each implemented activity in the previously obtained . Then we sort these activities in ascending order of their start times. The random key values for these sorted activities are set to , where is the number of implemented activities in . The above operations are also performed on the rest POP-1 implement list vectors to obtain the rest POP-1 random key vectors.

(3) Generation of POP shift key vectors. For each implemented activity in , its shift key value , where the start time of activity is obtained by the second stage of the TSHA. For the non-implemented activity, its shift key value is set to 0.

## Crossover

In the flexible project structure, the trigger of a choice affects the implementation of both optional and dependent activities. If standard crossover operators are applied to the implementation list, they tend to generate infeasible individuals. Taking the example project in Section 2.4 as an example, given two parent individuals and a crossover point corresponding to activity 3 (Fig. 4), we perform a one-point crossover and this results in two infeasible child individuals. In the two child individuals, the implemented activity 2 triggers choice 1, so only one optional activity can be implemented in . However, both activities 3 and 4 are implemented in the resulting son individual, while neither activity 3 nor activity 4 is implemented in the daughter individual. In addition, if activity 3 is implemented, all its dependent activities in must be implemented. However, although activity 3 is implemented in the son individual, its dependent activities 5 and 6 are not implemented. In the daughter individual, activity 3 is not implemented, but both activity 5 and activity 6 are implemented. Therefore, neither of the two child individuals is feasible.

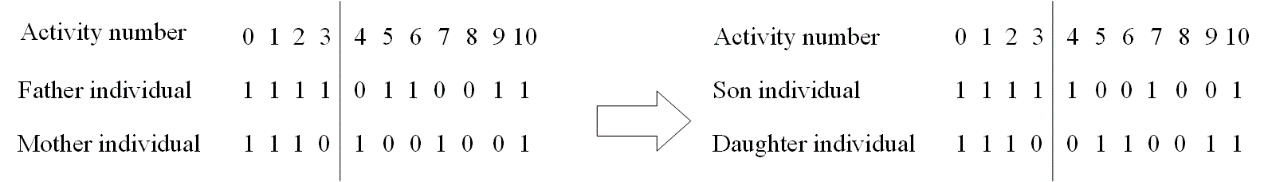


Fig. 4 An example of generating infeasible implementation lists after using the standard one-point crossover

According to the above analysis, the crossover operator for the needs to consider whether each choice is triggered and the impact of the optional activity’s status on dependent activities. Therefore, we specially design a one-point crossover operator for the implementation list. In our one-point crossover operator, the choices are chosen as crossover points.

Consider two individuals, i.e., a mother and a father . We now show how to generate the daughter and the son . We first draw a random number . If ( is a predetermined crossover probability), then the child individuals are obtained by directly copying the parent individuals; otherwise, the crossover operation is performed. In the crossover operation, we first initialize () by letting the status of all mandatory activities be executed, i.e., . The rest of the activities are set to be non-implemented, i.e., . Then, we randomly choose a crossover position from the choice set, and the implementation status of the optional activities in choice inherits from (), i.e., . After that, the implementation status of the dependent activities in () is updated according to the implemented activity in , i.e., . For the remaining choices , the implementation status of the activities in of () are inherited from (). Similarly, after this, the implementation status of the dependent activities is updated accordingly. In addition, choice () may be triggered by the following activities: (a) the implemented activities in choice of () that inherit from (), or (b) the implemented dependent activities in (). This means that there exists an optional activity in of () that needs to be implemented. If choice in () has been triggered (i.e., an activity in has been implemented), then in () inherits from (). Otherwise, in () inherits from () to ensure the feasibility of the generated individuals.

We give an example in Fig. 5 to show how to generate the son individual using our one-point crossover operator. First, we initialize . Suppose that choice 1 is the crossover point. Then the activities’ implementation status in choice 1 of inherits from , i.e., . Since activity 3 is implemented, all of its dependent activities in are also implemented, i.e., . For choice 2, the activities’ implementation status in inherits from . It should be noted that dependent activity 6 of is implemented, so choice 2 in is triggered. Since choice 2 in is not triggered, to ensure that is feasible, choice 2 of inherits from , i.e., . The implemented activity 9 in has no dependent activities.

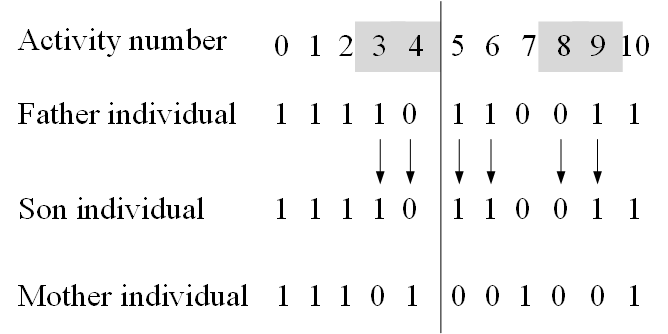


Fig. 5 Crossover operator for the implementation list

For and , the standard two-point crossover is used. The crossover operator is applied to the pair of each individual simultaneously. For each pair of parent individuals, a random number is generated. If , then the crossover operator is applied to the parent individuals. Specifically, we randomly select two individuals as the father and mother individuals. Two crossover points and ( are generated randomly. The elements of the father (mother) and between and are passed to the corresponding positions of the daughter (son). The remaining positions of the daughter (son) are filled by the corresponding elements of the mother (father) and . If , the crossover operator is applied to the parent individuals, and the child individuals are directly copied from the parent individuals.

## Mutation

Mutation operators are applied to the child individuals that are obtained by the crossover operator described in Section 3.3. Similar to the case for the crossover operation, the cannot directly employ standard mutation operators. Therefore, we specially design a mutation operator for s. Specifically, to avoid generating infeasible individuals, we use the choice as the mutation point. We sample a random number for each child. If ( is a predetermined mutation probability), then the mutation operator is applied to corresponding to the child individual (Algorithm 2).

|  |  |
| --- | --- |
| **Algorithm 2. The mutation operator for the implementation list** | |
| 1 | **Input**: Implementation list |
| 2 | **Output**: Mutated implementation list |
| 3 | ; |
| 4 | Generate a random number between 0 and 1; |
| 5 | If |
| 6 | Randomly generate a mutation position ); |
| 7 | For do |
| 8 | If |
| 9 | If |
| 10 | Randomly select an activity ; |
| 11 | While |
| 12 | Randomly select an activity; |
| 13 | End |
| 14 | ,0,; |
| 15 | Else |
| 16 | Randomly select an activity ; |
| 17 | ; |
| 18 | End if |
| 19 | // Update the implementation states of dependent activities based on the |
|  | implementation of optional activities |
| 20 | For in |
| 21 | If |
| 22 | ; |
| 23 | else |
| 24 | ; |
| 25 | End if |
| 26 | End for |
| 27 | Else |
| 28 | If |
| 29 | *,*; |
| 30 | For in |
| 31 | ; |
| 32 | End for |
| 33 | End if |
| 34 | End if |
| 35 | End for |
| 36 | End if |

In Algorithm 2, we randomly generate a mutation position ). If a previously triggered () choice is triggered () again, we randomly select an activity from the optional activity set to implement. The premise of selecting this activity is that the activity has not been implemented before being selected. Except for the selected activities, other activities are not implemented (Lines 8-14). If a choice is first triggered (, ), then an activity is randomly selected from to implement (Lines 15~18). After selecting the optional activity to be implemented, the implementation status of each dependent activity needs to be updated according to the implementation of the optional activity in (Lines 20-25). If a previously triggered () choice is not triggered () again, then the optional activities in are not implemented, and the corresponding dependent activities are not implemented (Lines 28-33). After one choice is considered, we determine in turn whether the subsequent choices are triggered.

For and , a uniform mutation is applied. Specifically, for each pair of , a random number is generated. If , then we use two random numbers to replace the corresponding .

## Local improvement

A local improvement method is used to further improve the final solution output by the CGA. The pseudocode of the local improvement method is shown in Algorithm 3. In Algorithm 3, indicates the current best objective function value. The binary variable indicates whether the resource usage is more leveled after an activity is assigned a new start time. Based on the schedule produced by the CGA, we can calculate the total float for each activity. Then, we form an activity list by arranging the activities in ascending order of their total floats (with the smallest activity number first being the tie-breaker). We adjust the start time of each activity in turn according to the sequence indicated in . Specifically, for each activity, a new start time within its total float is assigned, and the start times of the other activities remain unchanged. Every time the start time of an activity is changed, the performance indicator needs to be re-calculated, where is equal to the value of the objective function (1) and is used to evaluate the extent of resource leveling. If the objective function value is improved, the current best start time of activity is set to (Lines 11-18). After an activity is adjusted, if is equal to 1, then the start time of activity is changed to (Lines 19-22).

|  |  |
| --- | --- |
| **Algorithm 3. Local improvement method** | |
| 1 | **Input**: Schedule |
| 2 | **Output**: Improved schedule |
| 3 | denotes the objective function value corresponding to schedule ; |
| 4 | ; |
| 5 | For to |
| 6 | ; |
| 7 | For each ,; |
| 8 | For each ,; |
| 9 | ; |
| 10 | ; |
| 11 | For each in |
| 12 | Calculate performance indicator ; |
| 13 | If then |
| 14 | ; |
| 15 | ; |
| 16 | ; |
| 17 | End if |
| 18 | End for |
| 19 | If |
| 20 | ; |
| 21 | ; |
| 22 | End if |
| 23 | End for |

In our CGA, we use the maximum number of generated schedules in the worst case as the termination condition. The calculation of this value consists of two parts: (1) Every time our decoding procedure (Algorithm 1) is run, it is considered as one schedule is generated (Kolisch & Hartmann, 2006). (2) For the local improvement method (Algorithm 3), given an individual, in the worst-case scenario (i.e., only mandatory activities are executed and every possible start time is checked within their total float), the maximum number of evaluated schedules is , where is the number of mandatory activities.

# Computational experiments

Our algorithms have been coded in MATLAB R2014a. The computational experiments are performed on a PC with an Intel Core i5 3.20 GHz CPU using the 64-bit version of Windows 7. In the following subsections, we first introduce the benchmark dataset and algorithmic performance measures. Next, the parameters of the CGA are determined based on the Taguchi method for the DOE. Then, we report the computational results and perform sensitivity analysis. Finally, we compare our CGA with the existing metaheuristic algorithm.

## Experimental setup

The benchmark dataset used in our experiments is obtained by extending the PSPLIB (Kolisch & Sprecher, 1996) such that the instances in the PSPLIB have flexible structures. Specifically, we select the J30, J60, and J120 datasets, which contain a total of instances, from the PSPLIB. The original instances in the PSPLIB are generated by systematically varying some control parameters. The representative parameters include network complexity (NC), resource factor (RF) and resource strength (RS). The NC reflects the average number of immediate successors of an activity (Davis, 1975). The RF measures the average number of resources required by each activity. The RS describes the scarceness of resource capabilities (Cooper, 1976).

Each instance in the PSPLIB is extended according to the parameter settings in Table 2, where represents the number of choices, indicates the number of optional activities in the choices, is the number of optional activities that trigger dependent activities, and denotes the number of dependent activities of an optional activity. For , and , their values are generated by following Kellenbrink and Helber (2015). For each instance in the PSPLIB, we randomly choose an activity and let of its successor activities act as optional activities. These optional activities form an optional activity set that corresponds to a choice. This process is repeated until we get disjoint optional activity sets.

We also need to determine which optional activities trigger dependent activities and which activities are dependent activities of these optional activities. Specifically, for each instance, we randomly select an optional activity from . We then identify dependent activities that are triggered by the selected optional activity and ensure that the dependent activities are not in the set . The above process is repeated until optional activities and their dependent activities are obtained. The remaining activities except for the optional and dependent activities are mandatory activities.

Table 2. Parameter settings for each instance

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dataset |  |  |  |  |  |
| J30 | 32 | 2 | 2 | 1 | 1 |
| J60 | 62 | 3 | 2 | 2 | 1 |
| J120 | 122 | 5 | 3 | 4 | 3 |

In the flexible project, once the project structure is fixed, some activities and related precedence relationships become redundant and are deleted from the project network. In addition, to ensure the feasibility of the generated project structure, we also need to add additional precedence relationships such that each implemented activity is directly or indirectly connected to the dummy start and end activities. Specifically, when all successor (processor) activities of an implemented activity are not implemented (i.e., its successor (processor) activities are not mandatory), the dummy activity (0) is regarded as the successor (processor) of the activity.

In the experiment, for each instance, two different deadlines are specified: a tight deadline () and a loose deadline (), where is calculated by assuming that all activities are implemented. Therefore, we finally obtain instances.

We use CPLEX 12.9 to solve the M0 and M1 corresponding to each instance in the benchmark dataset respectively. The time limit for solving each instance is 600 seconds. As shown in Table 3, the benchmark dataset is divided into two subsets according to the solution results: the instance set , in which the optimal solutions are obtained, and the instance set , in which the optimal solutions are not obtained. It should be noted that contains instances where feasible solutions can be obtained and instances where feasible solutions cannot be obtained. CPU represents the average computation time of . To measure the performance of both models, we use ARI to evaluate their computations on instances where feasible solutions can be found. The ARI is calculated as follows:

|  |  |
| --- | --- |
|  | (14) |

Where and respectively represents the objective function values obtained by solving the mod M0 and M1of instance with CPLEX. indicates the number of instances where both the M0 and M1 of the instance can be solved by CPLEX. The larger the ARI value, the better the solution result of the linearized model.

Table 3. Number of instances in and

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | M0 | | |  | M1 | | |  |  |
| Dataset |  | # of solutions in | # of solutions in | CPU(s) |  | # of solutions in | # of solutions in | CPU(s) |  | ARD |
| J30 |  | 113 | 367 | 154.94 |  | 271 | 209 | 83.00 |  | -0.31% |
|  | 3 | 477(453) | 391.12 |  | 122 | 358 | 121.29 |  | -0.36% |
| J60 |  | 1 | 479(463) | 191.42 |  | 40 | 440 | 180.11 |  | 0.57% |
|  | 0 | 480(378) | - |  | 2 | 478 | 382.30 |  | 4.81% |
| J120 |  | 0 | 600(236) | - |  | 0 | 600(576) | - |  | 5.83% |
|  | 0 | 600(183) | - |  | 0 | 600(519) | - |  | -15.48% |

Based on Table 3, we can see that the linearized model M1 is better, therefore, in the subsequent experiments, we use the results obtained by M1 as the benchmark for comparison.

## Performance measures

The following four performance measures are used to evaluate the performance of the TSHA and the CGA:

(1) Average relative deviation (). For an algorithm that is applied to , which contains instances with known optimal solutions, we use the to evaluate the algorithm.

|  |  |
| --- | --- |
|  | (15) |

where represents the value of objective function (1) obtained by the for instance and is the optimal value of objective function (1) for instance calculated by CPLEX. A smaller value means that the quality of the solution produced by the is better.

(2) Average relative improvement (). For the TSHA and the CGA that are applied to , which contains instances with unknown optimal solutions, we use the to compare the relative performances of the two algorithms. The measures the improvement in the objective function value of the schedule obtained by the CGA compared to that obtained by the TSHA. The ARI is calculated as follows:

|  |  |
| --- | --- |
|  | (16) |

where () denotes the value of objective function (1) for instance solved by the CGA (TSHA). A larger value means that the CGA performs better than the TSHA.

(3) GAP. For an algorithm that is applied to , we also compare the gap between the value of objective function (1) obtained by and a lower bound.

|  |  |
| --- | --- |
|  | (17) |

We design two lower bound to measure the performance of the proposed algorithm:

1) Consider only the lower bound of mandatory activities, denoted with LB-M. Specifically, for each instance , we consider only the mandatory activities, then the average usage of resource at each time period is . This corresponds to the most ideal situation in resource leveling. In this case, the objective function value can be used as the lower bound .

2) Lagrangian relaxation, denoted with Lagr. Specifically, for each instance , constraint (10) is relaxed by introducing multipliers , . The objective function of the Lagrangian relaxation model LR(λ) is as follows:

|  |  |
| --- | --- |
| Minimize | (18) |

The above objective function can be rewritten as:

|  |  |
| --- | --- |
| Minimize | (19) |

The Lagrangian problem LR() consists of (2)-(6), (8), (11), (18).

In order to obtain the best lower bound of Lagrangian relaxation of the original model, the following Lagrangian dual problem is approximately solved by using the subgradient method (Bianco & Caramia, 2012):

|  |  |
| --- | --- |
| LD: Maximize LR() | (20) |

In each iteration of the algorithm, the Lagrangian coefficients need to be updated as follows:

* The subgradient values ​​for each relaxed constraint related to resource and time period is determined as.
* The scalar step is determined as , is the upper bound, and in order to obtain the best result, we take the solution obtained by CGA as the upper bound. is the current best lower bound, which is obtained by solving LR(λ) by CPLEX. is a scaling factor with an initial value of 2, and when is not improved in consecutive iterations, is reduced to half of it.
* The Lagrange coefficient is determined as .

The termination condition of the algorithm is 300 iterations or .

(4) Computational time (CPU). Average computational time in seconds for each instance.

## Parameter settings for the CGA

Selecting appropriate parameter values helps to improve the quality of the solution yielded by the CGA. We use the Taguchi method for the DOE to determine appropriate parameter values ​​for the CGA. Since most of the instances in J30 have been solved optimally, we use J30 to conduct the DOE. Our CGA has three key parameters: the population size (), crossover probability () and mutation probability (). We choose three levels for each parameter (Table 4). We use as the orthogonal array, which means that we have 9 treatments in total. The termination condition of the CGA is to generate 5000 schedules at most. The deadline for each instance is . We use the as the performance measure. Table 5 shows the orthogonal array and values.

Table 4. Combinations of parameter values

| Factor level | Parameters | | |
| --- | --- | --- | --- |
|  |  |  |
| 1 | 50 | 0.8 | 0.01 |
| 2 | 100 | 0.9 | 0.05 |
| 3 | 150 | 0.95 | 0.1 |

Table 5. Orthogonal array and values

| Experiment number | Parameters | | |  |
| --- | --- | --- | --- | --- |
|  |  |  |
| 1 | 1 | 1 | 1 | 0.025502 |
| 2 | 1 | 2 | 2 | 0.015874 |
| 3 | 1 | 3 | 3 | 0.014228 |
| 4 | 2 | 1 | 2 | 0.017165 |
| 5 | 2 | 2 | 3 | 0.017223 |
| 6 | 2 | 3 | 1 | 0.026353 |
| 7 | 3 | 1 | 3 | 0.019974 |
| 8 | 3 | 2 | 2 | 0.027010 |
| 9 | 3 | 3 | 1 | 0.018780 |

Based on Table 5, the ARD values of each parameter at different levels are listed in Table 6. In Table 6, the row “Delta” represents the range of the average objective function value for each parameter, and the row “Rank” indicates the importance ranking of each parameter.

Table 6. ARD and rank for each factor

| Level |  |  |  |
| --- | --- | --- | --- |
| 1 | **0.018535** | 0.020880 | 0.023545 |
| 2 | 0.020247 | 0.020036 | 0.020016 |
| 3 | 0.021921 | **0.019787** | **0.017142** |
| Delta | 0.003386 | 0.001093 | 0.006403 |
| Rank | 2 | 3 | 1 |

According to Table 6, we can see that the mutation probability has the greatest impact on the solution results of the CGA, the population size is the second-most important factor, and the crossover probability has the least impact. For each parameter, we choose the level with the smallest average ARD, i.e., , , and . These parameter settings will be used in subsequent experiments.

## Experimental results

We use the TSHA and the CGA to solve the benchmark dataset. For the CGA, three termination conditions are adopted, i.e., generating 1000, 3000, and 5000 schedules at most. This leads to three variants of the CGA: CGA-1K, CGA-3K and CGA-5K.

Table 7 shows the computational results on . From Table 7, we can find that the solution quality of the CGA is acceptable for small and medium-sized instances. In terms of the objective function values, the deviation between the objective values obtained by the CGA and the optimal values is between 1.19% and 5.83%. For the TSHA, this value is between 7.06% and 16.36%, indicating that the CGA outperforms the TSHA. For the running speed, the CPU time of the TSHA is significantly shorter than those of the CGA and CPLEX. The CPU time of the CGA increases with the increase of the number of schedules and the number of activities. In addition, the CGA can obtain higher quality solutions for instances with tight project deadlines.

Table 7. The computational results on

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | ARD | | | |  | CPU (s) | | | | |
|  |  | CGA-1K | CGA-3K | CGA-5K | TSHA |  | CGA-1K | CGA-3K | CGA-5K | TSHA | CPLEX |
| 32 |  | 1.86% | 1.35% | 1.19% | 8.85% |  | 1.02 | 3.16 | 5.31 | 0.01 | 83.00 |
|  | 2.68% | 1.48% | 1.25% | 7.06% |  | 1.01 | 3.17 | 5.31 | 0.02 | 121.29 |
| 62 |  | 5.70% | 3.96% | 3.51% | 16.36% |  | 3.52 | 11.00 | 18.48 | 0.06 | 180.11 |
|  | 5.83% | 4.53% | 3.98% | 10.22% |  | 2.64 | 8.26 | 13.87 | 0.15 | 382.30 |

The computational results on are shown in Table 8. It can be seen that the CGA outperforms the TSHA. The advantage of the CGA is more obvious under a tight deadline. However, the CPU time of the CGA is much longer than that of the TSHA.

Table 8. The computational results on

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | ARI | | |  | GAP(LB-M) | | | |  | CPU (s) | | | |
|  |  | CGA-1K | CGA-3K | CGA-5K |  | CGA-1K | CGA-3K | CGA-5K | TSHA |  | CGA-1K | CGA-3K | CGA-5K | TSHA |
| 32 |  | 8.41% | 9.21% | 9.42% |  | 26.92% | 25.77% | 25.49% | 39.14% |  | 0.99 | 3.08 | 5.16 | 0.02 |
|  | 10.14% | 11.12% | 11.44% |  | 33.95% | 32.49% | 32.00% | 49.61% |  | 0.99 | 3.12 | 5.25 | 0.02 |
| 62 |  | 7.47% | 8.32% | 8.69% |  | 24.36% | 23.15% | 22.62% | 34.85% |  | 3.32 | 10.43 | 17.52 | 0.06 |
|  | 9.92% | 10.96% | 11.39% |  | 30.79% | 29.21% | 28.56% | 45.24% |  | 3.31 | 10.48 | 17.64 | 0.09 |
| 122 |  | 10.96% | 11.73% | 12.01% |  | 13.28% | 12.22% | 11.85% | 27.88% |  | 8.76 | 27.86 | 46.93 | 0.32 |
|  | 13.05% | 13.60% | 13.94% |  | 16.00% | 15.23% | 14.73% | 34.17% |  | 8.62 | 27.72 | 46.82 | 0.42 |

要不要考虑下界的计算时间

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | ARI | | |  | GAP(LB-M) | | | |  | GAP(Lagr) | | | |  | CPU (s) | | | |
|  |  | CGA-1K | CGA-3K | CGA-5K |  | CGA-1K | CGA-3K | CGA-5K | TSHA |  | CGA-1K | CGA-3K | CGA-5K | TSHA |  | CGA-1K | CGA-3K | CGA-5K | TSHA |
| 32 |  | 8.41% | 9.21% | 9.42% |  | 26.92% | 25.77% | 25.49% | 39.14% |  | 27.92% | 26.77% | 26.50% | 40.35% |  | 0.99 | 3.08 | 5.16 | 0.02 |
|  | 10.14% | 11.12% | 11.44% |  | 33.95% | 32.49% | 32.00% | 49.61% |  | 33.72% | 32.25% | 31.76% | 49.41% |  | 0.99 | 3.12 | 5.25 | 0.02 |
| 62 |  | 7.47% | 8.32% | 8.69% |  | 24.36% | 23.15% | 22.62% | 34.85% |  | 19.35% | 18.18% | 17.69% | 29.41% |  | 3.32 | 10.43 | 17.52 | 0.06 |
|  | 9.92% | 10.96% | 11.39% |  | 30.79% | 29.21% | 28.56% | 45.24% |  | 27.05% | 25.54% | 24.91% | 41.21% |  | 3.31 | 10.48 | 17.64 | 0.09 |
| 122 |  | 10.96% | 11.73% | 12.01% |  | 13.28% | 12.22% | 11.85% | 27.88% |  |  |  |  |  |  | 8.76 | 27.86 | 46.93 | 0.32 |
|  | 13.05% | 13.60% | 13.94% |  | 16.00% | 15.23% | 14.73% | 34.17% |  |  |  |  |  |  | 8.62 | 27.72 | 46.82 | 0.42 |

We further show the comparison results between the best feasible solution obtained by CPLEX with the solution obtained by GA on (Table 9), using ARD as the performance measure, and replace in formula (15) with , represents the feasible solution obtained by CPLEX solving instance .

Table 9. The computational results on

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | ARD | | | |
|  |  | CGA-1K | CGA-3K | CGA-5K | TSHA |
| 32 |  | 2.66% | 1.75% | 1.52% | 12.58% |
|  | 2.99% | 1.87% | 1.49% | 15.21% |
| 62 |  | 3.25% | 2.28% | 1.86% | 11.96% |
|  | 3.43% | 2.23% | 1.73% | 15.44% |
| 122 |  | -6.09% | -6.95% | -7.28% | 5.74% |
|  | -41.84% | -42.25% | -42.49% | -33.85% |

It can be seen from Table 9 that in the small and medium-scale instances, the solution results of GA are not far from the solution results of model M1, while in large-scale instances, there is a big gap between the solution results of GA and model M1, especially in a loose deadline. This shows that when the scale of the instance becomes larger, the solution of the model M1 is more difficult and the solution result is worse.

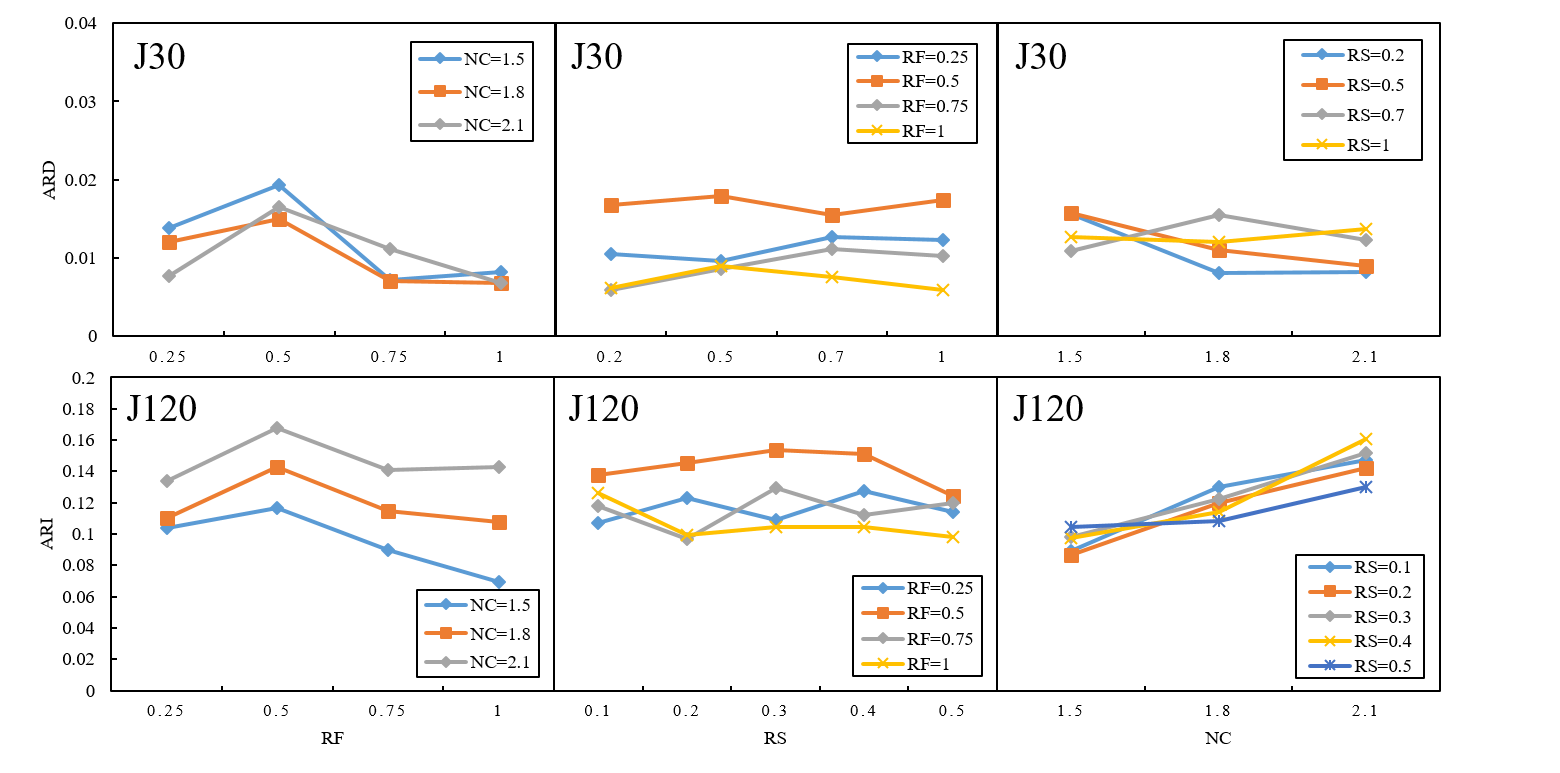
Overall, the experimental results show that our CGA can effectively solve the RLP-PS. The solution quality of the CGA is higher than that of the TSHA. The solutions obtained by the CGA are very close to the optimal solutions, especially for the small-scale instances.

## Sensitivity analysis

In this section, we further investigate the impact of different factors on the CGA. According to the experimental results in Section 4.4, the CGA has the best performance when the termination condition is 5000 schedules. For two different deadlines, the impacts of different factors are similar. Therefore, in our sensitivity analysis, we focus on CGA-5K and the tight project deadline.

We first investigate the impacts of the NC, RF and RS on the CGA when solving small-scale and easy instances (i.e., the J30 dataset in ). The results are shown in the top row of Fig. 6. It can be seen that the NC and RS have no obvious impact on the performance of the CGA in small-scale instances, thereby showing that our CGA is robust in this situation. For the impact of the RF, there are no obvious patterns. Although the performance of the CGA is poor when the RF is equal to 0.5, the CGA improves as the number of resource types increases. This means that our CGA is more suitable for projects with more resource types. Overall, our CGA is stable when solving small-scale instances.

Next, the impacts of the above-mentioned three factors on the CGA are analyzed for large-scale and hard instances (i.e., the J120 dataset in ). The results are shown in the bottom row of Fig. 6, from which we can see that as the NC increases, the improvement degree of the CGA gradually increases. This indicates that the more complex the project network is, the better the performance of the CGA relative to that of the TSHA. The impact of the RS is not obvious, indicating that the CGA is robust in situations with resource shortages. Similar to the small-scale instances, the impact of the resource types required by various activities exhibits no obvious patterns.



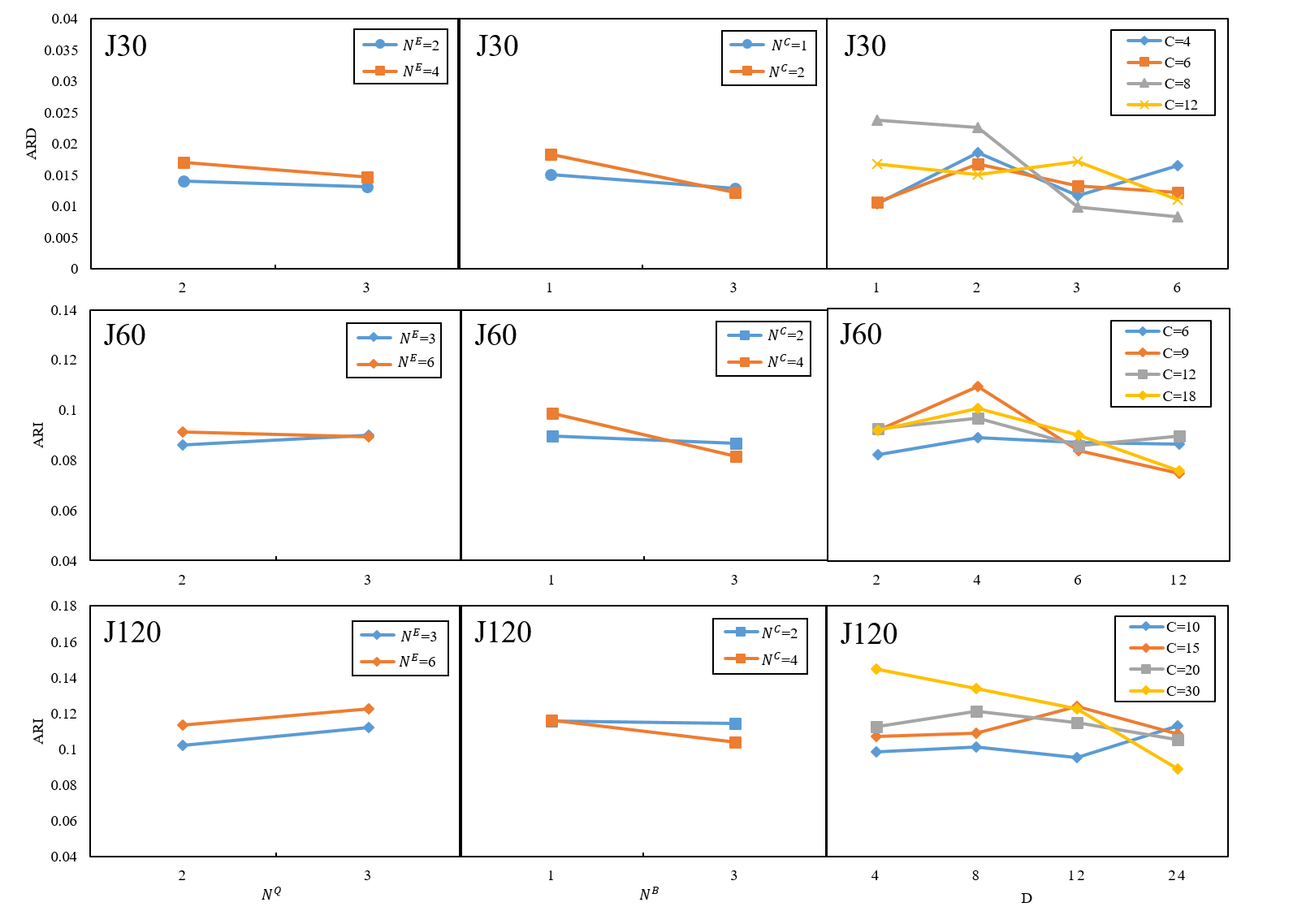
(a) The impact of NC and RF (b) The impact of RF and RS (c) The impact of RS and NC

Fig. 6 The impacts of PSPLIB parameters on the CGA-5K for J30

Finally, we analyze the impacts of the flexible structure related parameters (i.e., , , and ) on the CGA. These parameters used in the previous experiments are all given a single value and this is not suitable in the sensitivity analysis. Therefore, we extend the J30, J60 and J120 datasets in the PSPLIB based on the parameter settings listed in Table 10. This results in the dataset that is used in the sensitivity analysis. For J30, we consider only the instances that are solved optimally by CPLEX. For instances in J60 and J120, they are solved using CGA and TSHA. The sensitivity analysis results are shown in Fig. 7. In Fig. 7, indicates the number of optional activities () and indicates the number of dependent activities (). It can be seen that for J30, both the number of optional activities and the number of dependent activities have positive impacts on the CGA. For J60 and J120, with the increase of the number of optional activities, the CGA performs better than the TSHA. The number of dependent activities has a negative impact on the CGA. On the other hand, when the flexibility of the project increases, the advantage of the CGA over the TSHA is shrinking.

Table 10. Parameter settings for each instance in

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Dataset |  |  |  |  |
| J30 | 2, 4 | 2, 3 | 1, 2 | 1, 3 |
| J60 | 3, 6 | 2, 3 | 2, 4 | 1, 3 |
| J120 | 5, 10 | 2, 3 | 4, 8 | 1, 3 |



(a) The impact of and (b) The impact of and (c) The impact of and

Fig. 7 The impacts of flexible structure related parameters on the CGA

## Comparison with the existing metaheuristic algorithm

To further investigate the performance of our CGA, we compare the CGA-5K with an existing metaheuristic algorithm. There are no algorithms for the RLP-PS prior to us. Therefore, to obtain a benchmark algorithm, we implement a variant of the adaptive harmony search algorithm (AHSA) (Ponz-Tienda et al., 2017) that is originally designed for the RLP with minimal lags and is competitive with existing metaheuristics. In our implementation of the AHSA variant, we combine the minimum resource requirement priority rule that is used in the TSHA with the AHSA. Specifically, when solving an instance, its project structure is obtained based on the priority rule and then an instance of the RLP results. The RLP instance is finally solved by using the AHSA.

In the experiment, the same termination condition (i.e., generating a maximum of 5000 schedules) is used by the CGA-5K and the AHSA. The experimental results are shown in Table 11. In Table 11, the values of ARD and CGAP are obtained on and , respectively. It can be seen that our CGA outperforms the AHSA in all cases. Note that since the optimal solutions for J120 are not available, the ARD cannot be calculated and the corresponding cells in Table 11 are marked with “-”.

Table 11. Comparison results

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | ARD | |  | GAP(LB-M) | |  | GAP(Lagr) | |
|  |  | CGA-5K | AHSA-5K |  | CGA-5K | AHSA-5K |  | CGA-5K | AHSA-5K |
| 32 |  | 1.19% | 5.44% |  | 25.49% | 32.47% |  |  |  |
|  | 1.25% | 7.80% |  | 32.00% | 44.21% |  |  |  |
| 62 |  | 3.51% | 12.93% |  | 22.62% | 30.90% |  |  |  |
|  | 3.98% | 20.38% |  | 28.56% | 45.50% |  |  |  |
| 122 |  | - | - |  | 11.85% | 21.16% |  |  |  |
|  | - | - |  | 14.73% | 31.05% |  |  |  |

# Conclusions and future research

In this paper, considering flexible project structures, we study the models and algorithms for the RLP-PS. We present a nonlinear integer programming model for the RLP-PS and transform it into an equivalent linear integer programming model. To efficiently solve the RLP-PS, we design a TSHA and a customized CGA from the perspectives of problem decomposition and integration, respectively. In our CGA, by analyzing the characteristics of the RLP-PS, we devise a problem-specific schedule encoding/decoding mechanism, a crossover operator, a mutation operator and a local improvement method.

Based on the extended PSPLIB benchmark dataset, we conduct extensive computational experiments to validate our algorithms and compare them with the exact solver CPLEX and the existing metaheuristic algorithm. The experimental results show that the gap between the solutions obtained by our CGA and the optimal solutions is less than 1.5% when solving small-scale instances. Our CGA is also competitive when solving large-scale instances. Sensitivity analysis is also performed to further analyze the performance of our CGA. Our method provides support for project managers to determine appropriate project structures and level resource utilizations in flexible project structure environments.

Our future work will involve designing more efficient algorithms while considering uncertainties in projects. It will also be interesting to design more effective linearization method. In addition, our future work will consider the flexible project structures with more complex factors, such as multiple objectives and preemptions.

# Acknowledgments

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**

Alsayegh, H., & Hariga, M. (2012). Hybrid meta-heuristic methods for the multi-resource leveling problem with activity splitting. *Automation in Construction, 27*, 89-98.

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.

Bandelloni, M., Tucci, M., & Rinaldi, R. (1994). Optimal resource leveling using non-serial dyanamic programming. *European Journal of Operational Research, 78*(2), 162-177.

Barták, R., Čepek, O., & Hejna, M. (2007). Temporal reasoning in nested temporal networks with alternatives. In International Workshop on Constraint Solving and Constraint Logic Programming (pp. 17-31). Springer, Berlin, Heidelberg.

Beck, J. C., & Fox, M. S. (2000). Constraint-directed techniques for scheduling alternative activities. *Artificial Intelligence, 121*(1-2), 211-250.

Benjaoran, V., Tabyang, W., & Sooksil, N. (2015). Precedence relationship options for the resource levelling problem using a genetic algorithm. *Construction Management and Economics, 33*(9), 711-723.

Bianco, L., & Caramia, M. (2012). An exact algorithm to minimize the makespan in project scheduling with scarce resources and generalized precedence relations. *European journal of operational research, 219*(1), 73-85.

Birjandi, A., Mousavi, S. M., Hajirezaie, M., & Vahdani, B. (2019). Optimizing and solving project scheduling problem for flexible networks with multiple routes in production environments. *Journal of Quality Engineering and Production Optimization, 4*(1), 175-196.

Burgess, A., & Killebrew, J. B. (1962). Variation in activity level on a cyclical arrow diagram. *Journal of Industrial Engineering, 13*(2), 76-83.

Čapek, R., Šůcha, P., & Hanzálek, Z. (2012). Production scheduling with alternative process plans. *European Journal of Operational Research, 217*(2), 300-311.

Caramia, M. (2020). Project management and scheduling. *Annals of Operations Research*, *285(1)*, 1-8.

Chan, W.-T., Chua, D. K., & Kannan, G. (1996). Construction resource scheduling with genetic algorithms. *Journal of Construction Engineering and Management, 122*(2), 125-132.

Christodoulou, S. E., Ellinas, G., & Michaelidou-Kamenou, A. (2010). Minimum moment method for resource leveling using entropy maximization. *Journal of Construction Engineering and Management, 136*(5), 518-527.

Cooper, D. F. (1976). Heuristics for scheduling resource-constrained projects: An experimental investigation. *Management Science, 22*(11), 1186-1194.

Davari, M., & Demeulemeester, E. (2019). Important classes of reactions for the proactive and reactive resource-constrained project scheduling problem. *Annals of Operations Research, 274(1)*, 187-210

Davis, E. W. (1975). Project network summary measures constrained-resource scheduling. *AIIE Transactions, 7*(2), 132-142.

Demeulemeester, E. L., & Herroelen, W. S. (2002). Project Scheduling: A Research Handbook.

El-Rayes, K., & Jun, D. H. (2009). Optimizing resource leveling in construction projects. *Journal of Construction Engineering and Management, 135*(11), 1172-1180.

Gather, T., Zimmermann, J., & Bartels, J.-H. (2011). Exact methods for the resource levelling problem. *Journal of Scheduling, 14*(6), 557-569.

Gillies, D. W., & Liu, J. W.-S. (1995). Scheduling tasks with AND/OR precedence constraints. *SIAM Journal on Computing, 24*(4), 797-810.

Goldberg, D. E., Korb, B., & Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems, 3*(5), 493-530.

He, L., & Zhang, L. (2013). Dynamic priority rule-based forward-backward heuristic algorithm for resource levelling problem in construction project. *Journal of the Operational Research Society, 64*(8), 1106-1117.

Jaskowski, P., & Biruk, S. (2018). Reducing renewable resource demand fluctuation using soft precedence relations in project scheduling. *Journal of Civil Engineering and Management*, 24(4), 355-363.

Karaa, F. A., & Nasr, A. Y. (1986). Resource management in construction. *Journal of construction engineering and management, 112*(3), 346-357.

Kazemi, S., & Davari-Ardakani, H. (2020). Integrated resource leveling and material procurement with variable execution intensities. *Computers & Industrial Engineering, 148*, 106673.

Kellenbrink, C., & Helber, S. (2015). Scheduling resource-constrained projects with a flexible project structure. *European Journal of Operational Research, 246*(2), 379-391.

Kolisch, R., & Hartmann, S. (2006). Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*.

Kolisch, R., & Sprecher, A. (1996). PSPLIB—a project scheduling problem library. *European Journal of Operational Research, 96*, 205-216.

Koulinas, G., & Anagnostopoulos, K. (2013). A new tabu search-based hyper-heuristic algorithm for solving construction leveling problems with limited resource availabilities. *Automation in Construction, 31*, 169-175.

Li, H., & Demeulemeester, E. (2016). A genetic algorithm for the robust resource leveling problem. *Journal of Scheduling, 19*(1), 43-60.

Li, H., & Dong, X. (2018). Multi-mode resource leveling in projects with mode-dependent generalized precedence relations. *Expert Systems with Applications, 97*, 193-204.

Li, H., Xiong, L., Liu, Y., & Li, H. (2018). An effective genetic algorithm for the resource levelling problem with generalised precedence relations. *International Journal of Production Research, 56*(5), 2054-2075.

Li, H., Zhang, X., Sun, J., & Dong, X. (2020). Dynamic resource levelling in projects under uncertainty. *International Journal of Production Research*, 1-21.

Mavrotas, G. (2009). Effective implementation of the ε-constraint method in multi-objective mathematical programming problems. *Applied mathematics and computation*, 213(2), 455-465.

Neumann, K., Schwindt, C., & Zimmermann, J. (2012). *Project scheduling with time windows and scarce resources: temporal and resource-constrained project scheduling with regular and nonregular objective functions*: Springer Science & Business Media.

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.

Ponz-Tienda, J. L., Salcedo-Bernal, A., & Pellicer, E. (2017). A Parallel Branch and Bound Algorithm for the Resource Leveling Problem with Minimal Lags. *Computer-Aided Civil and Infrastructure Engineering, 32*(6), 474-498.

Ponz-Tienda, J. L., Salcedo-Bernal, A., Pellicer, E., & Beniloch-Marco, J. (2017). Improved Adaptive Harmony Search algorithm for the Resource Leveling Problem with minimal lags. *Automation in Construction, 77*, 82-92.

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

Servranckx, T., Coelho, J., & Vanhoucke, M. (2021). Various extensions in resource-constrained project scheduling with alternative subgraphs. *International Journal of Production Research*, 1-20.

Servranckx, T., & Vanhoucke, M. (2019). A tabu search procedure for the resource-constrained project scheduling problem with alternative subgraphs. *European Journal of Operational Research, 273*(3), 841-860.

Son, J., & Mattila, K. G. (2004). Binary resource leveling model: Activity splitting allowed. *Journal of construction engineering and management, 130*(6), 887-894.

Son, J., & Skibniewski, M. J. (1999). Multiheuristic approach for resource leveling problem in construction engineering: Hybrid approach. *Journal of Construction Engineering and Management, 125*(1), 23-31.

Tao, S., & Dong, Z. S. (2017). Scheduling resource-constrained project problem with alternative activity chains. *Computers & Industrial Engineering, 114*, 288-296.

Tao, S., Wu, C., Sheng, Z., & Wang, X. (2018). Stochastic project scheduling with hierarchical alternatives. *Applied Mathematical Modelling, 58*, 181-202.

Vanhoucke, M., & Coelho, J. (2016). An approach using SAT solvers for the RCPSP with logical constraints. *European Journal of Operational Research, 249*(2), 577-591.

Wu, I.-C., Borrmann, A., Beißert, U., König, M., & Rank, E. (2010). Bridge construction schedule generation with pattern-based construction methods and constraint-based simulation. *Advanced Engineering Informatics, 24*(4), 379-388.

Alsayegh, H., & Hariga, M. (2012). Hybrid meta-heuristic methods for the multi-resource leveling problem with activity splitting. *Automation in Construction, 27*, 89-98.

Bianco, L., & Caramia, M. (2012). An exact algorithm to minimize the makespan in project scheduling with scarce resources and generalized precedence relations. *European journal of operational research, 219*(1), 73-85.

Birjandi, A., Mousavi, S. M., Hajirezaie, M., & Vahdani, B. (2019). Optimizing and solving project scheduling problem for flexible networks with multiple routes in production environments. *Journal of Quality Engineering and Production Optimization, 4*(1), 175-196.

Hariga, M., & El-Sayegh, S. M. (2011). Cost optimization model for the multiresource leveling problem with allowed activity splitting. *Journal of construction engineering and management, 137*(1), 56-64.

Karaa, F. A., & Nasr, A. Y. (1986). Resource management in construction. *Journal of construction engineering and management, 112*(3), 346-357.

Kazemi, S., & Davari-Ardakani, H. (2020). Integrated resource leveling and material procurement with variable execution intensities. *Computers & Industrial Engineering, 148*, 106673.

Kellenbrink, C., & Helber, S. (2015). Scheduling resource-constrained projects with a flexible project structure. *European journal of operational research, 246*(2), 379-391.

Li, H., & Dong, X. (2018). Multi-mode resource leveling in projects with mode-dependent generalized precedence relations. *Expert Systems with Applications, 97*, 193-204.

Servranckx, T., Coelho, J., & Vanhoucke, M. (2021). Various extensions in resource-constrained project scheduling with alternative subgraphs. *International Journal of Production Research*, 1-20.

Son, J., & Mattila, K. G. (2004). Binary resource leveling model: Activity splitting allowed. *Journal of construction engineering and management, 130*(6), 887-894.

Tao, S., & Dong, Z. S. (2017). Scheduling resource-constrained project problem with alternative activity chains. *Computers & Industrial Engineering, 114*, 288-296.