A general variable neighborhood search based on path relinking algorithm for multi-skill resource-constrained project scheduling problem with multiple restrictions

H Dai \(^{1,2,3}\) and W Cheng\(^{1,2}\)

\(^{1}\) School of Mechanical Engineering, Southwest Jiaotong University, 610031 Chengdu, PR China
\(^{2}\) Technology and Equipment of Rail Transit Operation and Maintenance Key Laboratory of Sichuan Province, 610031 Chengdu, PR China
No. 111, North 1st Section of Second Ring Road, Jinniu District, Chengdu City, Sichuan Province, PRC
Email: ann@my.swjtu.edu.cn

Abstract. This paper proposes a general variable neighborhood search based path relinking algorithm (GVNS-PR) to solving the multi-skill resources-constrained project scheduling problem with multiple restrictions. Integrating a local refinement procedure and a path relinking framework, the proposed GVNS-PR is assessed on two datasets and achieves highly competitive results. One set of benchmark instances which are known in the literature can be used to demonstrate the capability of the proposed algorithm to seek for the high quality schedules, in comparison with the existed state-of-the-art algorithms. The other one revises the previous through adding the researched restrictions into instances information. Two critical components of the proposed algorithm are shed light on to confirm their key role to the successful method.

1. Introduction

A set of precedence-constrained tasks, which should compete for limited resources, will be scheduled in resource constrained project scheduling problem (RCPSP). As for the multi-skill RCPSP, which can be abbreviated as MS-RCPSP, it was derived from integrating RCPSP [1] with the multi-skill resources. The introduced skill domain [2] breaks the situation where restricts a one-to-one mapping between resources and certain kinds of tasks. The RCPSP is NP-hard [1] and therefore it is unlikely a polynomial time optimal algorithm for MS-RCPSP, a more general problem, to be found. In the literature, more and more attentions [3] have been paid to RCPSP and metaheuristics, including Tabu search [4, 5], Simulated Annealing [6, 7] or Genetic Algorithm [8], are the most usually methods. Swarm Intelligence also is an effective way for solving RCPSP, such as Bee Colony Optimization [9] or Ant Colony Optimization [10, 11].

Different from RCPSP, one of the hotspots of scheduling problem [12], MS-RCPSP has not attained the attention it deserves although several solving methodologies have been proposed. [13] introduced MS-RCPSP and [14] gave a benchmark. After that, [15] used a branch-and-bound method to search for an exact solution to minimize makespan and [2] presented a lower bound with a linear programming scheme for the RCPSP. A hybrid MILP/CP benders decomposition algorithm is developed in [16] to minimize the total staffing costs of RCPSP with multi-skilled human beings. [17] provided a MILP formulation and several additional sets of inequalities to RCPSP with flexible resources.

Besides, to research more thoroughly, a hybrid ant colony optimization approach (HAC) [18], a greedy randomized adaptive search procedure (GRASP) [19], a hybrid differential evolution and greedy algorithm (DEGR) [20] and a Co-Evolutionary algorithm [21] were proposed. Additionally, [22] gave a teaching-learning-based optimization algorithm. Lately, [23] researched on the integration of MS-RCPSP and step deterioration, proposing an improved tabu search (ITS) with two mutation operators.

MS-RCPSP takes account of the multi-skill of resources. However, some restrictions were avoided consciously when establishing the mathematical model, such as the budget of project, differentiated cost...
and the capacities of resources. This paper will study the MS-RCPSP under multiple restrictions with the goal of makespan optimization. Based on the preceding discussions, given the NP-hard nature of MS-RCPSP, it is not hard to admit the intractability of researched problem, and no run result can be found during polynomial time. For the convenience of expression, the derived problem is dubbed MS-RCPSPMR. To the best of our knowledge, there is no research concentrates on MS-RCPSPMR.

In this paper, we investigate a kind of heuristic, namely a path relinking algorithm and devise an algorithm called the General Variable Neighborhood Search-based Path Relinking (GVNS-PR). The general PR framework [24] shows outstanding performance in difficult combinatorial optimization problems, such as traveling salesman problem [25], vehicle routing problem [26] and unconstrained binary quadratic optimization [27]. Moreover, PR is proposed for multi-mode RCPSP where the project activities have a set of execution modes in [28] and a scatter search procedure based PR is used to consider RCPSP in [29]. In this paper, an effective path relinking algorithm for MS-RCPSPMR is introduced which relies on both a solution relinking procedure and a local search procedure. As for GVNS, it is simple and effective, proposed by [30] and yielded through systematic neighborhoods change. There are various optimization problems regarding GVNS as a way to pursue better solutions [31-33].

The remaining part of this paper is organized as follows. Section 2 gives the description of the MS-RCPSPMR and Section 3 elaborates the GVNS-PR. In Section 4, the computational experiments and results are shown before concluding in Section 5.

2. Problem statement

To define MS-RCPSPMR clearly, a 0-1 mixed integer programming model can be expressed as follows to illustrate the problem. 

$$G(V, E)$$ consists of a set $$V$$ of nodes denoting task $$(i, i \in V)$$ and a set $$E$$ of edges meaning the precedence relationships between a pair of tasks $$(i, j), (i, j) \in E$$. In other words, each pair $$(i, j) \in E$$ represents task $$i$$ precedes $$j$$. Specifically, task $$j$$ cannot start until task $$i$$ finished. To perform task $$i$$, the skill $$q_i$$ also should be provided. To complete $$n$$ tasks, a set $$K$$ of $$m$$ renewable resources will be provided. Resources vary in mastered skills $$Q_k$$, denoting the skills covered by resource $$k$$, and total available time $$Cap_k$$ which can be occupied by tasks. Naturally, according to the correspondences, a subset $$V_k$$ of $$V$$ which incorporates all tasks that can be processed by resource $$k$$ ($$q_i \in Q_k, i \in V_k$$) is available. In same way, a subset resources $$K_k$$, including all resources which can be utilized to handle task $$i$$ is available. Besides, the duration time $$d_{kj}$$ cost $$c_{kj}$$ which a specific resource $$k$$ spend on selected task $$j$$ also is a prior known. The total Budget for the whole project is restrict to $$Bud$$. Besides, a resource can perform at most one task simultaneously and one task can be executed by at most a resource at a time. The goal is minimizing the makespan.

Here we formulate a 0-1 integer programming model to decide the resource assignments and the beginning time $$b_i$$ of task $$i, (i \in V)$$. Firstly, the binary variables $$x_{kj}$$ receive value 1 if task $$j$$ is assigned to resource $$k$$, and 0 otherwise. The binary variables $$y_{ij}$$ will set to 1 if task $$i$$ is precede $$j$$, 0 otherwise.

**Objective function:**

$$\text{Minimize} \quad C_{\text{max}}$$

**Subject to:**

$$\sum_{j=1}^{n} x_{kj} = 1 \quad \forall j \in V \quad (1)$$

$$\sum_{k=1}^{m} x_{kj} = 1 \quad \forall j \in V \quad (2)$$

$$\sum_{j=1}^{n} y_{ij} - d_{ij} \leq Cap_k \quad \forall k \in K \quad (3)$$

$$\sum_{j=1}^{n} \sum_{i=1}^{n} c_{kj} * x_{kj} \leq Bud \quad (4)$$

$$\sum_{i=1}^{n} x_{ij} * d_{ij} \leq b_{f} - b_{i} \quad \forall (i, j) \in E \quad (5)$$

$$b_{i} + \sum_{j=1}^{n} x_{ij} * d_{ij} \leq C_{\text{max}} \quad \forall j \in V \quad (6)$$

$$x_{k} + x_{kj} - 1 \leq y_{ij} + y_{ji} \quad \forall i, j \in V, k \in K \quad (7)$$

$$b_{j} + \sum_{i=1}^{n} x_{ij} * d_{ij} - M * (1 - y_{ij}) \leq b_{i} \quad \forall i, j \in V, i \neq j \quad (8)$$

$$b_{i} + \sum_{j=1}^{n} x_{ij} * d_{ij} - M * (1 - y_{ij}) \leq b_{j} \quad \forall i, j \in V, i \neq j \quad (9)$$
Objective function (1) denotes the optimal direction. Constraints (2) restrict one resource to a task. Constraints (3) respect the skill constraints between resources and tasks. Constraints (4) show the effects of the time limits of resources while constraint (5) expresses restriction imposed by budget. Constraints (6) highlight the precedence relationship. Constraints (7) calculate the completion time of whole project. Constraints (8) illustrate logical relationships between the assignment variables and sequencing variables. That means, when resource \( k \) will handle with task \( i \) and \( j \) simultaneously, the sequence between two tasks must be determined: either \( i \) precedes \( j \) or \( j \) precedes \( i \). Constraints (9) are the big-M formulation to enforce the relationship between the sequencing variable and the continuous starting time variable, while Constraints (10) regular the domains of the variables.

3. GVNS-based path relinking algorithm

3.1. Evaluation function

Based on above notations and depictions, to evaluate the quality of a candidate solution \( s \in \Omega \), an evaluation function is adapted which is defined as:

\[
f(s) = \max \{ F_i : i \in V \} + M \cdot \delta
\]

In equation (11), \( M \) is a large positive constant such that \( M \to +\infty \) as \( n \to +\infty \). The finishing time of task \( i \) can be calculated by \( F_i = b_i + d_i \), where \( d_i \) relies on the assigned resource, and \( b_i \) are the earliest available time of assigned resource \( k, i \in V_k \). \( \delta \) will accumulate when any violation happen.

The first part of Eq. (11) represents the completion time of the final task, and the last part of equation (11) is an augmented penalty value. If \( \delta = 0 \) holds, it tells the schedule is feasible, and its evaluation function will be calculated by the first part, equal to makespan. Given two solutions \( s' \) and \( s'' \), \( s' \) is better than \( s'' \) if \( f(s') < f(s'') \).

3.2. Main framework

Let \( P \) denote a population of \( p \) candidate configurations. Let \( s^g, s^w \) separately be the best solution attainable so far and worst solution in \( P \). Let \( PairSet \) be a set of solution pairs \((s_i, s_j)\) initially consisted of all possible pairs in \( P \). Then, the proposed GVNS-PR can be described as Algorithm 1.

**Algorithm 1 Main sketch of GVNS-PR**

1: Input: instance \( I \), the population size \( p \), the search depth of GVNS \( \alpha \)
2: Output: the best schedule
3: repeat
4: \( P = \{s_1, \ldots, s_p\} \leftarrow \text{PopulationInitialization}(I) \)
5: if it is not the first loop then
6: \( s'' \leftarrow \arg \max f(s') : i = 1, \ldots, p ; P \leftarrow P \cup s^g \setminus s'' \)
7: end if
8: \( PairSet \leftarrow \{(s'_i, s'_j) : 1 \leq i < j \leq p \} \)
9: while \( PairSet \neq \emptyset \) do
10: Isolate a solution pair \((s'_i, s'_j)\in PairSet\) randomly, \( PairSet \leftarrow PairSet \setminus \{(s'_i, s'_j)\} \), \( s^o \leftarrow \text{PathRelinking}(s'_i, s'_j) \), \( s \leftarrow \text{GVNS_Operator}(s', \alpha) \), \( UpdatePopulation(s^o, P, PairSet, f(s)) \)
11: end while
12: Return \( s^* \leftarrow \arg \min (f(s) : s = 1, \ldots, p) \)
13: until a stopping criterion is satisfied

GVNS-PR builds an initial population \( P \) firstly, including \( p \) schedules, where any configuration is generated (Section 3.3) randomly and improved by the GVNS procedure (Section 3.5). Then, a while loop executes which constitutes the main body of the GVNS-PR. On each newly generation, the subsequent procedures are carried out. Firstly, from a configuration pair \((s'_i, s'_j)\), chosen from \( PairSet \) randomly, GVNS-PR builds the offspring \( s^o \) with a path relinking operator (Section 3.4). Next, the offspring will be set as a starting point improved by GVNS_operator (Section 3.5) and then the improved \( s^o \) is used to update \( P \) and \( PairSet \) (Section 3.6), where the depth of GVNS \( \alpha \) means the stopping criterion of the GVNS_operator, which equals to maximum number between two iterations without
improvement. The while loop will not continue until PairSet is empty. At the end of the whole while loop, \( P \) is recreated and above while loop is repeated until a maximum allowed number of path relinking rounds \( N_{pr} \) is achieved, where one “while” loop above means a round of path relinking. Eventually the algorithm ends and returns the best schedule \( s^b \).

3.3. Population initialization

To build an initial population (in Algorithm 2), the procedure is executed \( 3p \) times to generate a new solution. From the scratch, a new solution is constructed as follows. The operator starts from assigning each task with a random resource satisfying skill constraint. Then all tasks to same resource are sequenced in a non-increasing order of the priority level, which equals the number of tasks which can be handled after it is finished. The biggest advantage of sorting the tasks in this way is keeping the feasibility of schedule. Subsequently, the GVNS\(_{\text{operator}}\) with the evaluation function \( f(s) \) is used to optimize each generated solution to a local optimum. Eventually the \( p \) best obtained solutions are selected as the initial population.

Algorithm 2 Population\(_{\text{initialization}}\)

1: Input: The set \( V=\{v_1,\ldots,v_n\} \), \( K=\{k_1,\ldots,k_m\} \); skill and precedence relationship constraints; \( p \)
2: Output: the best \( p \) solutions
3: for \( \text{Iter}_{\text{pop}}=1:3p \) do
4:     Set \( RA:=V \)
5:     while \( RA\neq\emptyset \) do
6:         choose \( v_i \) from \( RA \) randomly; \( RA\leftarrow RA\setminus\{v_i\} \); isolate \( k\in K \) randomly and record \( A(v_i):=k \)
7:     end while
8:     for \( k=1,\ldots,m \) do
9:         Generate tasks sequence \( TQ_k \) for resource \( k \) in non-increasing order of tasks’ priority level
10:    end for
11: Calculate the objective value \( f(A,TQ)_{\text{Iter}_{\text{pop}}} \)
12: end for

3.4. The path relinking operator

A path relinking operator (PR) is dedicated to generate a path of intermediate solutions connecting two parent solutions. The general procedures of the operator are given in Algorithm 3.

Algorithm 3 Procedures of path relinking operator

1: Input: Instance \( I \); Two parent solutions \( s^1=(A^1,TQ^1) \) and \( s^2=(A^2,TQ^2) \)
2: Output: the offspring \( s^o=(A^o,TQ^o) \)
3: \( NC \leftarrow l: A^1/l \neq A^2/l, l=1,\ldots,n/l^{A^1}A^1/l \) denotes the assigned resource of \( l \)th task in \( s^1 \)
4: \( A(0)\leftarrow A^1; A\leftarrow A^1; m\leftarrow 1; r\leftarrow NC \)
5: while \( NC>0 \) do
6:     \( \ell^*\leftarrow\arg\min\{ f(A\oplus l): l\in NC \} \) \( A\oplus l \) means that the assigned resource of \( l \)th task of \( A \) is changed to the resource of \( l \)th task of the guiding solution \( A^2 \)
7:     \( NC\leftarrow NC\setminus\{\ell^*\}; A\leftarrow A\oplus\ell^*; A(m)\leftarrow A\oplus m\leftarrow m+l \)
8: end while
9: \( A^o\leftarrow A^2; \) generate \( TQ^o \) in same way as Algorithm 2.

As shown in Algorithm 3, PR generates a path of length \( HD+1 \) \( <A(0),A(1),\ldots,A(HD)> \) in a step way by starting from \( A(0) \), where the first and last assignments are called “initiating assignment” \( (A^1) \) and “guiding assignments” \( (A^2) \). And \( HD \) represents the Hamming Distance between \( A^1 \) and \( A^2 \). Moreover, two consecutive assignments in the path differ only by the assigned resource of the one single task. PR changes the corresponding resource of a task at each relinking step to build the path. To choose one resource, PR uses the following greedy criterion. Let \( NC \) denote the set of tasks whose assigned resources are different between the current assignment \( A(i), (i=1,\ldots,HD-1) \) and the guiding assignment \( A^2 \). PR examines the tasks included in \( NC \) and selects a new resource for task \( l \in NC \) such that changing the resource of task \( l \) of \( A \) to the resource of the task \( l \) of \( A^2 \) leads to the greatest improvement of the evaluation function. \( A\oplus l \) is used to denote the new assignment generated by changing the resource of
task \( l \) of A to the resource of task \( l \) of \( A' \). After that \( TQ(k) \) for each resource \( k \) will be generated in same way as Section 3.3 to constitute a complete solution \( s=(A,TQ) \). The above procedures repeat until \( NC \) becomes empty.

From a solution path, a reference solution should be selected for further improvement. In this article, we pick an great solution distanced from both the initiating and guiding solutions as reference solution [27]. Therefore a candidate list (CL) incorporating all intermediate solutions which have a distance of at least \( \frac{\xi}{HD} \) far from the initiating and guiding solutions is constructed, where \( \xi \) is a predetermined value and set empirically. Next the best schedule in \( CL \) is picked as the reference solution to be further improved by the procedures in Section 3.5.

3.5. GVNS\(_{\text{operator}}\)

This section lies in ensuring a high quality local optima, wherever starting point is. Given three neighborhood structures \( N_1, N_2, N_3 \) and an initial solution \( s' \), GVNS\(_{\text{operator}}\) does the follow refinement. To start with, the random sequence order \( RS \) is generated to apply these neighborhood structures. For example, if \( RS \) equals \( (2, 3, 1) \), the search starts from \( N_2 \) and finishes at \( N_1 \). For each neighborhood, a new local optima \( s'' \) is obtained through the corresponding local search to the incumbent solution \( s' \), set at \( s_0 \) at the beginning of GVNS\(_{\text{operator}}\). If \( s'' \) is better, \( s' \) is replaced by \( s'' \), accepted as a descent to go on the search in current neighborhood; otherwise the search turns to the next neighborhood. One iteration ends until the last neighborhood structure in \( RS \) is explored. Then the search continues until the stopping criteria is met. The general sketch of GVNS\(_{\text{operator}}\) is described in Algorithm 4.

```
Algorithm 4 GVNS\(_{\text{operator}}\)
1: Input: Initial solution \( s_0 \); a set of neighborhood structures \( N_k (k=1, 2, 3) \); \( \alpha \)
2: Output: The best solution \( s_b \)
3: Calculate the objective value \( f(s) \), \( s_b \leftarrow s_0; s^b \leftarrow s' \)/* \( s' \) is the current solution */
4: \( d=0 \) /* \( d \) counts the consecutive iterations where \( s' \) is not improved */
5: repeat
6: Generate \( RS \) and apply the relevant mechanism, update \( s' \) if a better schedule is attained
7: if \( f(s') < f(s^b) \) then
8: \( s^b \leftarrow s' \) and reset the counter \( d:=0 \)
9: else
10: \( d:=d+1 \)
11: end if
12: until \( d = \alpha \)
```

Three neighborhood structures \( N_k (k=1, 2, 3) \) are adopted in GVNS\(_{\text{PR}}\). The neighborhood \( N_1 \) is defined by the swap operator. The neighborhood \( N_2 \) is designed based on reversion. As for the neighborhood \( N_3 \), it is designed by the alter operator. Notably, all the neighborhood moves can be executed on the premise of guaranteeing the precedence relationships.

3.6. Updating population and PairSet

As illustrated in Algorithm 1, \( P \) and \( PairSet \) will be updated when an excellent offspring is obtained through PR and improved further by GVNS\(_{\text{operator}}\). The relevant procedure is organized as follows. First of all, if it is better than \( s^* \) and the distance \( Distance(s,P) \) between \( s \) and the population \( P \) is larger than \( 0.1n \), for any improved offspring solution \( s \), \( s'' \) is replaced by \( s \), where \( Distance(s,P) \) is defined by \( Distance(s,P)=\min\{distance(s,s_i):s_i \in P\} \) in which \( distance(s,s_i) \) equals the Hamming Distance between \( (s,s_i) \). If not, \( s \) is abandoned. When the updated population is given, \( PairSet \) should be updated accordingly: all pairs containing \( s^* \) are deleted and all pairs generated by combining \( s \) with others in \( P \) are incorporated.

4. Computational experiments

This section plans to evaluate the GVNS\(_{\text{PR}}\) through comparing with the state-of-the-art methods. Besides that, we study two key ingredients of the GVNS\(_{\text{PR}}\) to get some insight into its behavior. Two groups of instances will be employed. For the lack of literature researching to MS-RCPSPMR to have
the contrast, we firstly solve the MS-RCPSP on exist benchmark instances with GVNS-PR in favor of argument to its effectiveness. Next, the multiple restrictions integrated with MS-RCPSP are taken into account.

4.1. Benchmark instances

The first experimental set is consist of 30 benchmark instances irrespective of multiple restrictions, available in [14], and the other set is generated with some modifications. The detail is described in Section 4.4.

4.2. Parameter settings and experiment protocol

GVNS-PR was programmed in MATLAB R2015b and all experiments below were executed on a personal computer equipped with an Inter Core i3 processor (3.10 GHz CPU and 2GB RAM) in Windows 7. To eliminate the randomness as much as possible, twenty replications are carried out.

Table 1 shows the settings and descriptions of the parameters in GVNS-PR.

| Parameters | Section | Description                  | Values |
|------------|---------|------------------------------|--------|
| \( p \)    | 3.3     | Population size for GVNS-PR  | 20     |
| \( \alpha \) | 3.5     | Depth of GVNS                | 50,000 |
| \( M \)    | 3.1     | Penalty value for infeasibility | 10,000 |
| \( N_p \)  | 3.2     | Maximum path relinking rounds | 3,000  |
| \( \xi \)  | 3.4     | Distance parameter in PR operator | 0.2    |

4.3. Experimental results on known MS-RCPSP benchmark

First experimental group assesses the performance of GVNS-PR on the set of 30 known instances without the consideration of multiple restrictions. In these instances, GVNS-PR sets the duration time of each task as a constant leaving out the differences among resources. Moreover, the project budget and capacities of resources are set to infinity. Table 2 records the computational results of GVNS-PR with the goal of time optimization, as well as the results achieved by other reference algorithms.

Notice that, columns 1 gives the instance name and column 2 equals the previous best values \( f_{\text{preb}} \). Columns 3 to 5 give the best configurations obtained by DEGR [20], ITS [23] and GRASP [19]. The results of the GVNS-PR are shown in columns 6 to 7, incorporating minimum value \( f_{\text{best}} \) over 20 runs and the average value \( f_{\text{avg}} \). The best are indicated in bold. Additionally, to verify whether there exists an essential difference between the best results of GVNS-PR and other reference algorithms, the relative percentage deviation (RPD) is defined by \( \text{RPD}(\%) = \frac{(f_{\text{preb}} - f_{\text{best}})}{f_{\text{best}}} \times 100 \) where the positive value of RPD means the improvement of result achieved by GVNS-PR.

Table 2 discloses that the outcomes of GVNS-PR are noteworthy compared to the state-of-the-art results. GVNS-PR improves the best results for 12 cases and matches for 8 instances. Compared with the 10 out of 30 cases solved by DEGR, 5 best achieved by ITS and 5 schedules obtained by GRASP, GVNS-PR delivered the best configurations for 20 instances.

4.4. Computational results on MS-RCPSPMR instances

4.4.1. Experiments design. Since the extra included restrictions, the differences of resources should be reflected firstly to make it meet the actual requirements. [14] gives the basic processing time \( d_j \) of task \( j \) and the cost \( c_{jk} \) of resource \( k \). Then the duration \( d_{jk} \) that resource \( k \) with highest cost spent dealing with task \( j \) equals \( d_j \). At this basis, the duration for other resources to process task \( j \) can be calculated by: \( d_{jk} = d_j + d_j \gamma, k \in K, k \neq k' \), where \( \gamma \) is a factor obtained randomly from \((0, c_k / c_{k'})\). The cost \( c_{jk} \) for resource \( k \) to handle with task \( j \) is generated in similar way. Select the task \( j \) with longest duration time \( d_j \) as a base and set \( c_{jk} = c_k \). Then the cost \( c_{ik} \) for other tasks \( i \) using resource \( k \) is defined as: \( c_{ik} = c_{ik} + c_k \epsilon, i \in V, i \neq j \) where \( \epsilon \) is a parameter valued randomly from \((0, d_{ik} / d_j)\). As for the restrictions about \( C_{api} \), they are generated from \((\theta \ast \text{Time}_{\text{low}}/n, \theta \ast \text{Time}_{\text{high}}/n)\), where \( \theta \) is attained randomly from \([1, \text{Time}_{\text{high}}/\text{Time}_{\text{low}}]\), as well as \( \text{Time}_{\text{low}} \) is calculated by the sum of tasks processing times with the
assumption that each task is assigned with the resource accompanying with shortest time cost. As for $Time_{high}$, it equals the sum of the longest durations for all tasks.

Table 2. Comparison of the GVNS-PR with other algorithms on MS-RCPSP benchmark dataset [14]

| Instances   | $f_{preb}$ | DEGR | ITS | GRASP | GVNS-PR | RPD(%) |
|-------------|------------|------|-----|-------|---------|--------|
|             | $f_{best}$ | $f_{avg}$ | $f_{best}$ | $f_{avg}$ | $f_{best}$ | $f_{avg}$ | $f_{best}$ | $f_{avg}$ |
| 100/10/26/15 | 236        | 236  | 239 | 250   | 238     | 241    | -0.85    |
| 100/10/47/9  | 256        | 256  | 259 | 263   | 254     | 255    | 0.78     |
| 100/10/48/15 | 247        | 247  | 251 | 255   | 246     | 248    | 0.40     |
| 100/10/64/9  | 250        | 250  | 251 | 254   | 247     | 253    | 1.20     |
| 100/10/64/15 | 248        | 248  | 258 | 256   | 246     | 248    | 0.81     |
| 100/20/22/15 | 133        | 134  | 133 | 134   | 133     | 136    | 0.75     |
| 100/20/46/15 | 164        | 164  | 176 | 170   | 161     | 163    | 1.83     |
| 100/20/47/9  | 138        | 138  | 139 | 180   | 134     | 138    | 2.90     |
| 100/20/65/15 | 205        | 240  | 213 | 213   | 205     | 205    | 3.76     |
| 100/20/65/9  | 134        | 134  | 143 | 143   | 134     | 137    | 0.00     |
| 100/5/22/15  | 484        | 484  | 485 | 503   | 484     | 486    | 0.00     |
| 100/5/46/15  | 529        | 529  | 544 | 552   | 528     | 534    | 0.19     |
| 100/5/48/9   | 471        | 491  | 471 | 509   | 490     | 490    | 0.20     |
| 100/5/64/15  | 483        | 483  | 486 | 501   | 481     | 484    | 0.41     |
| 100/5/64/9   | 475        | 475  | 476 | 494   | 475     | 477    | 0.00     |
| 200/10/128/15 | 462       | 462  | 507 | 491   | 479     | 489    | -3.68    |
| 200/10/50/15 | 488        | 488  | 519 | 522   | 488     | 499    | 0.00     |
| 200/10/50/9  | 489        | 489  | 490 | 506   | 489     | 494    | 0.00     |
| 200/10/84/9  | 508        | 517  | 508 | 526   | 509     | 511    | 1.55     |
| 200/10/85/15 | 479        | 479  | 482 | 486   | 478     | 482    | 0.21     |
| 200/20/145/15 | 245       | 245  | 291 | 262   | 254     | 262    | -3.67    |
| 200/20/54/15 | 270        | 270  | 292 | 304   | 283     | 298    | -4.81    |
| 200/20/55/9  | 257        | 262  | 273 | 257   | 257     | 270    | 0.00     |
| 200/20/97/15 | 336        | 336  | 345 | 347   | 334     | 336    | 0.60     |
| 200/20/97/9  | 253        | 253  | 271 | 253   | 256     | 268    | -1.19    |
| 200/40/133/15 | 159       | 159  | 182 | 163   | 153     | 159    | 3.77     |
| 200/40/45/15 | 164        | 164  | 167 | 164   | 159     | 163    | 3.05     |
| 200/40/45/9  | 144        | 168  | 158 | 144   | 146     | 150    | -1.39    |
| 200/40/90/9  | 145        | 160  | 173 | 145   | 148     | 155    | -2.07    |
| 200/40/91/15 | 149        | 153  | 149 | 153   | 156     | 160    | -1.96    |

In terms of the restriction about project budget $Bud$ for any instance, it is generated randomly from $[Bud_{low}+ (Bud_{high}−Bud_{low})/min(m,3)]$, where $Bud_{low}$ means that each task is assigned with the cheapest resource irrespective of the skill constraints and $Bud_{high}$ equals the sum of the expensive cost. For simplicity of distinguishing, the attained instances are identified by adding a suffix. For example, the instance 100/10/26/15/m represents the instance 100/10/26/15 with imposed multiple restrictions.

Due to zero comparative data and competing heuristic algorithms to settle down MS-RCPSPSMR, comparisons with known solutions are impossible. Thus, a general PR and GVNS were programmed as reference algorithms to analyze the effects of two important elements of the GVNS-PR.

4.4.2. Importance of the path relinking framework. As demonstrated in [33], the GVNS shows great performance. So it is meaningful to shed light on whether GVNS-PR has an advantage over GVNS. Here, a comparison experiment between GVNS-PR and GVNS has been carried out and each instance of MS-RCPSPSMR is solved 20 times independently by the GVNS and GVNS-PR. Table 3 reports the computational results, including the best values available in column 2 and 4, the avenue makespan in column 3 and 5. Finally, a parameter $Dev$ (column 6) to measure the performance deviation is defined by: $Dev(\%)=\frac{f_{best2}}{f_{best1}}\times100$. The positive value of $Dev(\%)$ means that GVNS has a better performance in terms of the quality of best configuration.
Table 3. Comparison of the GVNS-PR with PR on the modified MS-RCPSPMR dataset.

| Instances          | GVNS-PR | PR    | Dev(%) |
|--------------------|---------|-------|--------|
| 100/10/26/15/m     | 326     | 379.6 | 389    | 461.6 | -19.33 |
| 100/10/47/9/m      | 314     | 347.8 | 402    | 498.5 | -28.03 |
| 100/10/48/15/m     | 321     | 387.6 | 437    | 500.1 | -36.14 |
| 100/10/64/9/m      | 319     | 402.3 | 461    | 514.0 | -44.51 |
| 100/10/64/15/m     | 289     | 306.5 | 373    | 426.8 | -29.07 |
| 100/20/22/15/m     | 187     | 216.4 | 290    | 364.7 | -55.08 |
| 100/20/46/15/m     | 296     | 328.7 | 364    | 407.9 | -22.97 |
| 100/20/47/9/m      | 201     | 287.8 | 315    | 381.1 | -56.72 |
| 100/20/65/15/m     | 236     | 302.2 | 362    | 413.2 | -53.39 |
| 100/20/65/9/m      | 185     | 254.4 | 299    | 384.5 | -61.62 |
| 100/5/22/15/m      | 583     | 639.5 | 703    | 779.4 | -20.58 |
| 100/5/46/15/m      | 617     | 700.5 | 679    | 744.3 | -10.05 |
| 100/5/48/9/m       | 559     | 608.7 | 638    | 691.1 | -14.13 |
| 100/5/64/15/m      | 556     | 626.3 | 647    | 700.8 | -16.37 |
| 100/5/64/9/m       | 577     | 583.1 | 633    | 679.5 | -9.71  |
| 200/10/128/155/15/m| 613     | 680.2 | 691    | 751.3 | -12.72 |
| 200/10/50/15/m     | 594     | 686.6 | 702    | 774.4 | -18.18 |
| 200/10/50/9/m      | 546     | 621.6 | 660    | 711.9 | -20.88 |
| 200/10/84/9/m      | 579     | 649.7 | 677    | 800.4 | -16.93 |
| 200/10/85/15/m     | 534     | 600.4 | 607    | 618.2 | -13.67 |
| 200/20/145/15/m    | 292     | 322.5 | 368    | 413.0 | -26.03 |
| 200/20/54/15/m     | 371     | 413.3 | 476    | 566.6 | -28.30 |
| 200/20/55/9/m      | 337     | 365.8 | 405    | 472.2 | -20.18 |
| 200/20/97/15/m     | 393     | 447.9 | 506    | 569.9 | -28.75 |
| 200/20/97/9/m      | 334     | 399.2 | 471    | 518.3 | -41.02 |
| 200/40/133/15/m    | 209     | 253.4 | 292    | 358.4 | -39.71 |
| 200/40/45/15/m     | 194     | 240.2 | 250    | 319.7 | -28.87 |
| 200/40/45/9/m      | 215     | 266.4 | 259    | 306.9 | -20.47 |
| 200/40/90/9/m      | 183     | 210.5 | 218    | 288.8 | -19.13 |
| 200/40/91/15/m     | 227     | 279.8 | 278    | 367.5 | -22.47 |

In Table 3, results tell that GVNS-PR has an overwhelming advantage over GVNS. GVNS-PR attains better and worse solutions in terms of both the minimum value and average on 30 and 0 instances. The average value of Dev(%) equals to -27.83%, accomplishing with a high of -61.62% for instance 100/20/65/9/m. These data all indicate that PR is appropriate for solving the researched issue.

4.4.3. Importance of the GVNS_operator. Section 4.4.2 demonstrated the superiority of PR. Naturally, the problem whether GVNS is useful for MS-RCPSPMR comes out. Table 4 summarizes the results achieved by PR and GVNS-PR. $f_{best}$ means the minimum value and $f_{avg}$ is computed as average value of 20 runs. The Dev(%) is calculated with same meaning with Section 4.4.2.

Table 4. Comparison of the GVNS-PR with PR on the modified MS-RCPSPMR dataset.

| Instances          | GVNS-PR | PR     | Dev(%) |
|--------------------|---------|--------|--------|
| 100/10/26/15/m     | 326     | 379.6  | 343    | 391.4 | -5.21  |
| 100/10/47/9/m      | 314     | 347.8  | 347    | 361.5 | -10.51 |
| 100/10/48/15/m     | 321     | 387.6  | 365    | 377.1 | -13.71 |
| 100/10/64/9/m      | 319     | 402.3  | 372    | 424.1 | -16.61 |
| 100/10/64/15/m     | 289     | 306.5  | 303    | 329.7 | -4.84  |
| 100/20/22/15/m     | 187     | 216.4  | 213    | 229.4 | -13.90 |
| 100/20/46/15/m     | 296     | 328.7  | 309    | 330.9 | -4.39  |
| 100/20/47/9/m      | 201     | 287.8  | 243    | 279.8 | -20.90 |
| 100/20/65/15/m     | 236     | 302.2  | 274    | 297.9 | -16.10 |
Generally, GVNS-PR significantly outperforms PR. First, compared with the general PR, the configuration with minimum completion time $f_{best1}$ of GVNS-PR is better than $f_{best2}$ of PR for 29 out of 30 instances, and the average value is better for 30 out of 30 instances. Moreover, the average value of $Dev(\%)$ is -12.70%, with a high of -26.32% to solve the instance 200/40/133/15/m. All above experimental results confirm that GVNS-PR, including GVNS as a local refinement procedure and PR framework to diversify search, is quite effective to solve the MS-RCPSP and MS-RCPSPMR.

5. Conclusions
The proposed GVNS-PR for MS-RCPSP and MS-RCPSPMR concludes an effective local search procedure and a greedy path relinking operator. We tested on 30 benchmark instances and modified known dataset further. The results show the effectiveness compared to the state-of-the-art algorithms. The investigations of two essential ingredients research on the behavior of GVNS-PR. First, the GVNS is useful when solving MS-RCPSPMR. Second, the population evolution based PR is contribute to the algorithm's performance significantly.

In this paper we discussed MS-RCPSPMR. It would be suggested to analyze the structural feature of optimal schedule and solved it with other metaheuristics.

Acknowledgments
This work was partially supported by the National Natural Science Foundation of China (No. 51675450) and Sichuan Science and Technology Program (No. 2019YFG0300, No. 2019YFG0285).

References
[1] Brucker, P, Drexl, A and Mohring, R 1999 J. EJOR, 112 3-41.
[2] Al-Anzi, F S, Al-Zame, K and Allahverdi, A 2010 J. Communications & Network, 03 1125-30.
[3] Hindi, K S, Yang, H and Fleszar, K 2002 J. IEEE Transactions on Evolutionary Computation, 6 512-18.
[4] Akeshtech, Z and Mardukhi, F 2017 J. Int. J. of Computer Science and Network Security, 17 125-34.
[5] Poppenborg, J and Knust, S 2016 J. Or Spectrum, 38 305-34.
[6] Laurent, A, Deroussi, L and Grangeon, N 2017 J. Computers & Industrial Engineering, 112 634-
44.

[7] Farughi, H, Amiri, A and Abdi, F 2018 J. Int. J. of Engineering, 31 780-91.
[8] Đumić, M, Sisejko, D, Coric, R and Jakobovic, D 2018 J. Future Generation Computer System 86 211-21.
[9] Ata, B and Coban, R 2015 Proc. IEEE Int. Conf. on Automation, 2014.
[10] Gonzalez-Pardo, A, Ser, J D and Camacho, D 2017 J. Applied Soft Computing, 60.
[11] Chen, Q, Lin, K and Wei, C C 2017 J. Computer Engineering and Appl. 53 249-54.
[12] Kolisch, R and Hartmann, S 2006 J. EJOR. 174 23-37.
[13] Bellenguez, O and Néron, E 2004, Proc. Int. Conf. on Practice & Theory of Automated Timetabling.
[14] Myszkowski, P B, Skowroński, M and Sikora, K 2015 Proc. Computer Science and Information Systems.
[15] Bellenguez-Morineau, O and Neron, E 2007 J. RAIRO-Operations Research. 41 155-70.
[16] Li, H and Womer, K 2009 J. J. of Scheduling. 12 281-98.
[17] Correia, I, Lourenço, L L and Saldanha-Da-Gama, F 2012 J. Or Spectrum. 34 635-63.
[18] Myszkowski, P B, Skowronski, M E and Olech, L P 2015 J. Soft Computing 19 3599-619.
[19] Myszkowski, P B and Siemieński, J J 2016 Proc. Int. Conf. computational Collective Intelligence p.402-11.
[20] Myszkowski, P B, Olech, L P, Laszczyk, M and Skowronski, M E 2017 J. Applied Soft Computing. 62.
[21] Myszkowski, P B, Olech, L P, Laszczyk, M and Skowronski, M E 2017 Proc. Federated Conf. on Computer Science and Information Systems. 2017.
[22] Zheng, H Y, Wang, L and Zheng, X L 2017 J. Soft Computing 21 1537-48.
[23] Dai, H F, Cheng, W M and Peng G 2017 J. Int. J. of Industrial & Systems Engineering 26 508.
[24] Glover, F, Laguna, M and Martí R 2000 J. Control & Cybernetics. 29 653-84.
[25] Ruben, I and Celso R C 2017 J. Int. Transctions in Operational Research 24 1307-23.
[26] Lacomme, P, Prins, C, Prodhon, C and Ren, L 2015 J. Engineering Applications of Artificial Intelligence. 38 237-51.
[27] Wang, Y, Lü, Z, Fred, G and Hao, J K 2012 J. EJOR. 223 595-604.
[28] Muritiba, A E F, Rodrigues, C D, Araujo da C and Franciio 2018 J Computers & Operations Research. 92 145-54.
[29] Berthaut, F, Pellerin, R, Hajji, A and Perrier, N 2018 J. Int. J. of Project Organisation and Management. 10 1-36.
[30] Hansen, P and Mladenović, N 2008 J. EJOR. 191 593-95.
[31] Kirlik, G and Oguž, C 2012 J. Computers & Operations Research 39 1506-20.
[32] Nenad, M, Dragan, U, Sai’d, H and Aleksandar, I 2012 J. EJOR. 220 270-85.
[33] Lei, H, Laporte, G and Bo, G J T 2012 J. TOP. 20 99-118.
[34] Glover, F 1997 Proc. Ewuerpean Conf. on Artificial Evolution pp 1-51.