Big Data and Information Analytics
©American Institute of Mathematical Sciences
Volume **2**, Number **1**, January **2017**
doi:10.3934/bdia.2017007
pp. **39–58**

A MOVING BLOCK SEQUENCE-BASED EVOLUTIONARY ALGORITHM FOR RESOURCE INVESTMENT PROJECT SCHEDULING PROBLEMS

Xiaoxiao Yuan, Jing Liu* and Xingxing Hao

Key Laboratory of Intelligent Perception
and Image Understanding of Ministry of Education
Xidian University
Xi'an 710071, China

Abstract. Inspired by the representation designed for floorplanning problems, in this paper, we proposed a new representation, namely the moving block sequence (MBS), for resource investment project scheduling problems (RIPSPs). Since each activity of a project in RIPSPs has fixed duration and resource demand, we consider an activity as a rectangle block whose width is equal to the duration of the activity and height the resource needed by the activity. Four move modes are designed for activities, by using which the activity can move to the appropriate position. Therefore, the new representation of the project of RIPSPs consists of two parts: an activity list and a move mode list. By initializing the move modes randomly for each activity and moving it appropriately, the activity list can be decoded into valid solutions of RIPSPs. Since the decoding method of MBS guarantees that after moved, each activity is scheduled in the left-most and bottom-most position within a coordinate, which means that each activity in the corresponding project is arranged as early as possible when the precedence constraints and resource demands are satisfied. In addition, the multiagent evolutionary algorithm (MAEA) is employed to incorporate with the newly designed MBS representation in solving RIPSPs. With the intrinsic properties of MBS in mind, four behaviors, namely the crossover, mutation, competition, and self-learning operators are designed for agents in MAEA. To test the performance of our algorithm, 450 problem instances are used and the experimental results demonstrate the good performance of the proposed representation.

1. **Introduction.** The project scheduling problems (PSPs) are very general issues in the area of planning [1, 3, 26], management [22], engineering [4, 18], and designing [5]. To solve PSPs, the objectives may be minimizing the cost, total makespan or due date performance [25]. The resource investment project scheduling problem (RIPSP) is a variation of PSPs with no limitation on the resource capacity, and the objective is to minimize total resource costs with a due date.

In recent years, RIPSPs have attracted increasing attentions. Möhring in [19] discussed minimizing costs of resource requirements with a fixed completion duration in the early years. He considered this problem as the problem of scarce time and proved that this problem is NP-hard. Demeulemeester [7] presented another

2010 *Mathematics Subject Classification.* 90C59.
Key words and phrases. Resource investment project scheduling problems, moving block sequence representation, multiagent evolutionary algorithm.
* Corresponding author: Jing Liu.

exact algorithm for resource investment problems and considered this problem as the resource availability cost problem (RACP). Drexl *et al.* in [9] discussed the lower and upper bounds using Lagrangian relaxation and column generation methods. An extension of resource investment problems that included time dependent renting costs for resources and the maximum and minimum time lags was studied by Nübelin [20], which was labeled as the resource renting problem (RRP), and a depth first branch and bound algorithm was proposed. Brucker *et al.* [2] gave a comprehensive survey of problems of scarce time, scarce resources, and other PSPs. Leveraging design principles to optimize technology portfolio prioritization was studied by Depenbrock *et al.* [8] and a simulation based heuristic approach for handling RIP was presented in [23].

The genetic algorithm (GA) has been used to solve RIPSPs by Shadrokh *et al.* in [24], in which, a valid activity sequence and four available resource capacities generated between the lower and upper bounds were initialized as an individual. Then, the algorithm started with the initial individuals and optimized the makespan as well as the four available resource capacities. Serial schedule generation scheme (SGS) is used to obtain the makespan. And they changed the four available resource capacities one by one in the procedure of finding optimal solutions. In the fitness function, they jointed both the makespan and the four available resource capacities with two reasonable penalty values together.

Recently, Xiong *et al.* in [25] proposed an evolutionary multi-objective approach for stochastic extended resource investment project scheduling problems (SERIPSPs) which is a new version of RIPSPs. They employed scenarios to capture the space of possibilities and proposed a robustness measure for the solutions when uncertainties like duration perturbation, resource breakdown, and precedence alteration interact. Finally, the SERIPSPs have been formulated as multiobjective optimization problems with the objectives as makespan, cost, and robustness. The experiments show that their approach is effective in solving SERIPSPs.

The representation is a very important component in solving RIPSPs, which determines the size and topology of the searching space. Among all literature we mentioned above in solving RIPSPs or their variations, the representation used for the projects consists of two parts: the activity sequence and the resource capacity list. The SGS used to transform activity sequences to precedence feasible schedules is either serial schedule generation scheme (SSS) or parallel schedule generation scheme (PSS) [11, 12, 13]. The SSS can always generate an active schedule and guarantee that each active schedule corresponds to an appropriate activity sequence [11], but it has the disadvantage that the same schedule may be mapped from more than one activity sequence [6], which means that the size of encoding space is greater than the active schedules [10]. As for the PSS, it can always generate a feasible schedule if does exist. However, it suffers from the weakness that the schedule it generates might not be the optimal schedule [11].

In [16], a new representation, namely moving block sequence (MBS), has been proposed to solve the floorplanning problem which is a basic step in the physical design of very large scale integration (VLSI). Liu *et al.* in [16] analyzed the strength of MBS thoroughly and obtained the conclusion that the four moving modes of MBS are useful and the MBS is a successful extension of the bottom-left (BL) and bottom-left-fill (BLF) method, which are classical approaches in the field of cutting and packing, and the computational complexity for decoding the MBS to a floorplan is between linear and quadratic in considering the number of blocks. Through

numerous experiments they declared that the MBS is very useful for extending the applications of evolutionary algorithms (EAs) in the area of floorplanning. Thus, with the intrinsic properties of RIPSPs in mind and inspired by the MBS, we come up with the idea that considering an activity as a rectangle block whose width is equal to the activity duration and height the resource needed by the activity. For each activity, there are four types of move modes which are similar to [16]. Therefore, we can encode the projects into the representation that consist of two parts: the activity list and move mode list. Then, the activity list is decoded into an active schedule according to the corresponding move modes.

From [16] we can know that by using MBS representation in RIPSPs, the computational complexity of decoding an activity list into an active schedule is between linear and quadratic in terms of the activity number, and the decoding process guarantees that the activity will be arranged as early as possible when the precedence constraints are satisfied. With the decoding method, we do not need to initial the available resource capacities within a lower and upper bounds as proposed in [24] and then optimize them step by step. We just need to calculate the maximum width and height of the placed activities at the end of decoding. And according to the obtained results, a directional mutation operator is designed. Moreover, the decoding method and the MBS representation can be applied to any EAs in the area of PSPs. Theoretical analyses that conducted in [21] show that the representation of solution is important for the effectiveness of evolutionary algorithms.

Many previous works show that the multiagent evolutionary algorithm (MAEA) has great potential in solving NP-hard problems [15, 17, 27]. In this paper, we integrate MAEA with MBS to solve RIPSPs and the proposed approach is labeled as MBS_{MAEA}-RIPSP. Four behaviors, namely crossover, mutation, competition, and self-learning operator, are designed for the agents in considering the environment they live. To test the performance of MBS_{MAEA}-RIPSP, the experiments are conducted on Möhring instances and three generated test sets J10, J14 and J20. By comparing with other EAs that have been employed to deal with the same test sets, the experimental results demonstrate that the MBS_{MAEA}-RIPSP can obtain better performance.

The remaining parts of the paper are organized as follows. Section 2 gives the definition RIPSPs. Section 3 describes the MBS representation for RIPSPs. Section 4 presents the algorithm for transforming an MBS to a schedule. Section 5 introduces MBS_{MAEA}-RIPSP in detail, including the definition of agents, design of operators and the general framework of MBS_{MAEA}-RIPSP. Section 6 shows the experimental results. Finally, conclusions are given in Section 7.

2. **RIPSPs.** In a RIPSP [24], the project is represented as a directed activity-on-node (AON) network G shown in FIGURE 1. In an AON network, each node denotes an activity and the arrow line represents the precedence relationship between two activities i.e. if there is an arrow line from activity v to w, then v precedes w, which means activity w cannot start before v is finished. Thus, v is the predecessor of w and w is the successor of v. Note that one activity may have more than one predecessor and successor as well. For each activity i, let P_i represent the set of predecessors of it, and S_i the set of successors. The network G contains n non-dummy activities with two dummy activities labeled as 0 and $n+1$, which are the initial and terminal activities respectively. $K = \{1, 2, \ldots, \rho\}$ is a set of ρ renewable resource(s). Each activity i ($i = 0, 1, \ldots, n+1$) has a fixed duration D_i and requires r_{ik} units of renewable resource k ($k \in K$) during its duration. For

two dummy activities, we have $D_0 = D_n = 0$ and $r_{0,k} = r_{n+1,k} = 0$, $k \in K$. The objective of RIPSPs is to determine the start time of activity i, S_i, ($i = 0, 1, \ldots, n+1$), and the available resource capacity R_k ($k \in K$), such that the precedence relations of activities are satisfied and the total cost of all resources and tardiness penalty is minimized. A constant cost of C_d for each unit of time delay from a due date T is incurred and C_k is a resource cost for each unit of available capacity. Let $x_{i,t}$ be 1 if activity i starts at time t and 0 otherwise, then we have $S_i = \sum_{t=0}^{T} t \times x_{i,t}$. The mathematical model of RIPSPs can be described as follows,

$$\min \Big\{ \sum_{k=1}^{\rho} C_k R_k + C_d \times \max\{0, S_{n+1} - T\} \Big\} \tag{1}$$

$$s.t. \sum_{t=0}^{T} t \times x_{i,t} \geq D_j + \sum_{t=0}^{T} t \times x_{j,t},\ j \in P_i, i = 1, 2, \ldots, n+1 \tag{2}$$

$$\sum_{i=1}^{n} \sum_{u=t-D_i+1}^{t} r_{i,k} \times x_{i,u} \leq R_k,\ t = 0, \ldots, T, k \in \{1, 2, \ldots, \rho\} \tag{3}$$

$$\sum_{t=0}^{T} x_{i,t} = 1,\ i = 1, 2, \ldots, n+1 \tag{4}$$

$$x_{1,0} = 1 \tag{5}$$

$$x_{i,t} \in \{0, 1\},\ i = 1, 2, \ldots, n+1, t = 0, 1, \ldots, T \tag{6}$$

$$R_k \geq 0,\ k \in K = \{1, 2, \ldots, \rho\} \tag{7}$$

where objective (1) is to minimize the total cost of a project. Constraints (2) is the precedence relation between each pair of activities (i, j), where j immediately precedes i. And (3) limits the total resource usage within each period to the available amount. Constraints (4) and (5) make sure that each activity i can only have one start time. (6) and (7) are the range of certain variables.

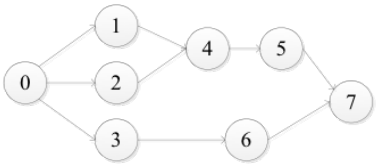

FIGURE 1. An example of precedence graph.

3. **Moving block sequence representation for RIPSPs.** In a RIPSP, there are n activities need to be scheduled. To solve RIPSPs with the idea of MBS designed for floorplanning problems, we need to first transform each activity into a hard rectangular block. It is well-known that in RIPSP, each activity has a fixed duration and demands for several renewable resources. Therefore, the fixed duration of an activity can be regarded as the width of a hard rectangular block and the maximum resource demand as the height of the block. After transforming all activities to blocks, the block will be placed in the first quadrant shown as

FIGURE 2, in which the X-axis stands for the time and the Y-axis is the resource. In addition, to locate the position of a block in the coordinate, two variables x_{lb} and y_{lb}, which corresponds to the abscissa and ordinate of the lower left corner of the block respectively, are necessary to be defined. Assume B^i, (i = 0, 1, ..., n+1), where B^0 and B^{n+1} are two dummy blocks, denotes the blocks that correspond to the n-activity project, the information structure of block B^i is defined as follows,

struct B^i
{
(x_{lb} , y_{lb}): coordinate of the left-bottom corner of block B^i;
width: width of block B^i;
height: height of block B^i.
}

where $B^0.width = B^{n+1}.width = 0$ and $B^0.height = B^{n+1}.height = 0$.

Each block starts from an initial position and moves in the first quadrant until it reaches an appropriate position. Four initial positions labeled as 0 to 3 shown in FIGURE 2 are designed and the corresponding move rules will be described in the following context.

In FIGURE 2, (Box^{RX}, $Box^{T}Y$) is the coordinate of the right-top corner of the shaded rectangle. And the coordinate of the left-bottom corner of shaded rectangle is always (0, 0). Suppose that i-1 blocks have been placed in the shaded rectangle. For the ith block B^{pi} in the block list, the four move rules corresponding to the four initial positions are described as follows,

Move rule 0 : the coordinates of initial positions ($B^{pi}.x_{lb}$, $B^{pi}.y_{lb}$) are set to (max_t, Box^{TY}), where max_t is the maximum finish time among all predecessors of activity i. From the initial position, the block can only move downward until no downward movement is possible.

Move rule 1: the coordinates of initial positions ($B^{pi}.x_{lb}$, $B^{pi}.y_{lb}$) are set to (Box^{RX}, 0). Then this block can only move leftward until no leftward movement is possible.

Move rule 2: the coordinates of initial positions ($B^{pi}.x_{lb}$, $B^{pi}.y_{lb}$) are set to (Box^{RX}- $B^{pi}.width$, Box^{TY}). Then this block can repeatedly move downward and leftward. Giving priority to downward movement so that this block can only moves leftward if no downward movement is possible. But sometimes the initial positions above may violate precedence relationship constraints. If that happens, then we need to adjust the initial position to (max_t, Box^{TY}) and it can only move downward until no downward movement is possible.

Move rule 3: the coordinates of initial position ($B^{pi}.x_{lb}$, $B^{pi}.y_{lb}$) are set to (Box^{RX}, emphBox^{TY}- $B^{pi}.y_{lb}$). Then this block can repeatedly move leftward and downward. Giving priority to left movement so that this block can only move downward if no left movement is possible.

Given the activity list and the move mode list, an ***MBS*** can be defined as follows,

Definition 3.1. An ***MBS*** has two vectors, namely a block list and a move mode list.

$$\boldsymbol{MBS} = ((B^{p0}, B^{p1}, \ldots, B^{pn}, B^{pn+1}), (M_0, M_1, \ldots, M_n, M_{n+1})) \in \boldsymbol{F} \qquad (8)$$

where the first part is the corresponding block list that consists of a valid permutation of n non-dummy blocks which satisfies precedence constraints, and two dummy blocks B^{p0} and B^{pn+1} which correspond to block B^0 and B^{n+1}, respectively. The

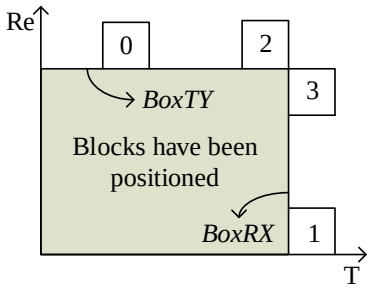

FIGURE 2. An example of precedence graph.

second part denotes the move mode list which is randomly initialized between 0 and 3. $\boldsymbol{F}$ represents the encoding space, namely, the search space.

Theorem 3.2. *The size of search space* $\boldsymbol{F}$ *for activities satisfies* $4^n \ll |\boldsymbol{S}| \ll n!4^n$.

Proof. From **Definition 3.1** we can know that one ***MBS*** consists of a list of activities and a list of move mode. In reality, the activity list is a permutation of all activities, thus, if we consider an extreme condition that for each non-dummy activity there is one and only one predecessor and one successor activity, which means only one possible permutation exists, the lower bound of the size of search space is equal to 1×4^n, in which 4^n is the number of combination of move modes. For another extreme case, if the precedence relationship among activities are not taken into consideration, the number of combination of activities is $n!$ and integrate with the effect of move mode, the upper bound is obtained equals to $n!4^n$. However, in actual RIPSPs these two extreme conditions can barely happen, for there exist intricate precedence constraints among activities and many types of resource constraints for each activity. Therefore, the size of search space is much larger than the lower bound and much smaller than the upper bound, simultaneously. □

To generate a valid block list of an ***MBS*** which satisfies precedence constraints, the Algorithm 2 described in [10] which enumerates all the topological orders of a network G is employed.

4. **Algorithm transforming an MBS to a schedule.** The location relationships between blocks need to be considered when moving them. We know that each hard rectangular block has four edges, namely top, bottom, left, and right edge. According to the designed four move rules, all blocks need to be moved leftward or downward. The problem of finding the reasonable left-most or the bottom-most of a block is changed to that of judging the relative position between two edges. Liu *et al.* [16] designed suitable structures $e^{//X}$ and $e^{//Y}$ to record the positions of edges that parallel to X-axis and Y-axis, respectively. In MBS_{MAEA}-RIPSP, we also employ the structures of $e^{//X}$ and $e^{//Y}$ to record edge information, with which we can easily judge if a block can further move or not. The description of $e^{//X}$ and $e^{//Y}$ are as follows,

struct $e^{//X}$
{
x_l: X-coordinate of the left point;
x_r: X-coordinate of the right point;

y: Y-coordinate of the bottom edge.
}
struct $e^{//Y}$
{
x: X-coordinate of the edge;
y_b: Y-coordinate of the bottom point;
y_t: Y-coordinate of the top point.
}

If a block is projected to the X-axis vertically or to the Y-axis horizontally, the blocks in the projection area will affect the bottom-most or the left-most position where this block can move to. Thus, two types of overlap relations are defined in **Definition 4.1**, and FIGURE 3 shows all kinds of top-overlaps and right-overlaps, in which the shadow areas are the projection of block $\boldsymbol{A}$.

Definition 4.1. Let $a^{//X}$ and $b^{//X}$ be two edges parallel to the X-axis. If $a^{//X}$ and $b^{//X}$ satisfy (9), $a^{//X}$ top-overlaps $b^{//X}$; otherwise, $a^{//X}$ un-top-overlaps $b^{//X}$. Let $a^{//Y}$ and $b^{//Y}$ be two edges parallel to the Y-axis. If $a^{//Y}$ and $b^{//Y}$ satisfy (10), $a^{//Y}$ right-overlaps $b^{//Y}$; otherwise, $a^{//Y}$ un-right-overlaps $b^{//Y}$.

$$(a^{//X}(y) \geq b^{//X}(y)) \text{ and } (a^{//X}(x_r) > b^{//X}(x_l)) \text{ and } (a^{//Y}(x_l) < b^{//X}(x_r)) \quad (9)$$

$$(a^{//Y}(x) \geq b^{//Y}(x)) \text{ and } (a^{//Y}(y_t) > b^{//Y}(y_b)) \text{ and } (a^{//Y}(y_b) < b^{//Y}(y_t)) \quad (10)$$

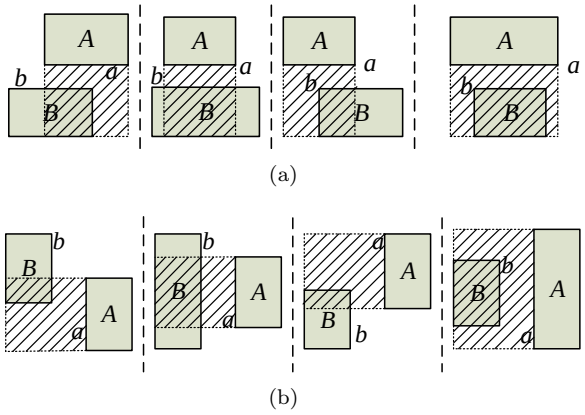

FIGURE 3. The types of overlaps. (a) All kinds of top-overlaps, (b) all kinds of right-overlaps.

Assume that *CoverRightX* and *CoverTopY* stand for the left-most and the bottom-most positions where blocks can move to, respectively, i.e., *CoverRightX* is the X-coordinate of the left-most position where a block can move leftward and *CoverTopY* is the Y-coordinate of the bottom-most position where a block can move downward. With the intrinsic properties of hard rectangular blocks and the precedence constraints among activities in mind, the appropriate position where a block can move to is given as follows.

When moving leftward, there are two conditions to stop block $\boldsymbol{A}$. The first condition is the left edge of block $\boldsymbol{A}$ right-overlaps the right edge of block $\boldsymbol{B}$, as can be seen in FIGURE 3 (b). We mark the position as $CoverRightX_i$; the second condition is that the X-coordinate of the left edge of block $\boldsymbol{A}$ is equal to max_t, where max_t is the maximum finish time of the predecessors of activity $\boldsymbol{A}$. According to the above cases, $CoverRightX \leftarrow max(CoverRightX_i, max\_t)$, where $max(m, n)$ returns the maximum between m and n. Finding the bottom-most position $CoverBottomY$ is analogous to $CoverRightX$, except that the precedence constraints can be omitted.

When block $\boldsymbol{A}$ top-overlaps block $\boldsymbol{D}$, there may exist some other continuous blocks have the same Y-coordinate with block $\boldsymbol{D}$, as shown in FIGURE 4. The minimum X-coordinates of the left edges of those blocks is labeled as $CoverLeftX$. In the same way we can get the $CoverTopY$. If $CoverLeftX > CoverRightX$ and $CoverLeftX$ - $CoverRightX \geq \boldsymbol{A}.duration$, the downward movement of $\boldsymbol{A}$ is feasible after moved leftward, otherwise, block $\boldsymbol{A}$ cannot move downward furthermore, i.e. it will stop at $CoverRightX$. Similarly, if $CoverTopY > CoverBottomY$ and $CoverTopY$ - $CoverBottomY \geq \boldsymbol{A}.height$, the leftward movement of $\boldsymbol{A}$ is feasible after moved downward, otherwise, block $\boldsymbol{A}$ will stop at $CoverBottomY$.

In general, almost every block will move repeatedly leftward and downward until reach the stopping criterion. For example, the blocks that have the initial position 3 will move leftward firstly to the $CoverRightX$, then performing the judgment that whether the blocks can move downward or not. If the blocks can further move, then the corresponding movement will be conducted until they are placed in the appropriate positions. For the blocks that have other initial positions, similar procedures are conduceted.

FIGURE 4 (a) and (b) are two cases without precedence constraints, in which block $\boldsymbol{A}$ moves left until right-overlaps block $\boldsymbol{B}$. FIGURE 4(c) and (d) are cases considering the precedence constraints, where $\boldsymbol{C}$ is a predecessor of block $\boldsymbol{A}$, thus block $\boldsymbol{A}$ will stop when reaches the right edge of block $\boldsymbol{C}$.

$BtoT^{//X}$ and $LtoR^{//Y}$ denote two ordered sets of edges which are parallel to the X-axis and Y-axis, respectively. They are designed to record the top edges from bottom to top and the right edges from left to right of the blocks that have been placed. **Algorithm 1** describes the algorithm of transforming an ***MBS*** from an initial block list and a corresponding move mode list to a schedule and FIGURE 5 shows the overall process clearly. All blocks will be placed and moved in the first quadrant. Since B^{p0} and B^{pn+1} are two dummy blocks, they are omitted during the transforming procedure and B^{p1} is placed directly at the most left-bottom corner of the first quadrant, then B^{pi}, $i \in [2, n]$ will be moved iteratively according to the move mode list at the first quadrant in the order they occur in block list.

In **Algorithm 1**, when all blocks have been placed in the first quadrant, Box^{RX} is the total makespan of the project. From above we know that the maximum resource demand of every activity is regarded as the height of the related block, therefore, the Box^{TY} is not the actual resource capacity that needed by the project. Thus, there is a need to calculate the actual resource capacity R_k ($k \in K$).

In the proposed MBS_{MAEA}-RIPSP, the idea of Effective Activity (EA) and the set of all EA for resource k, AEA_k that described in [24] are employed to find the available resource capacity R_k ($k \in K$). For an ***MBS*** and resource k, activity j is defined to be an EA in AEA_k, if this activity or parts of it, is scheduled during time intervals $[S_i, S_i + D_i]$, i.e., width interval $[B^{pi}.x_{lb}, B^{pi}.x_{lb} + B^{pi}.width]$ on X-axis. In this way, we find each AEA_{ki} for each activity i. Before finding the

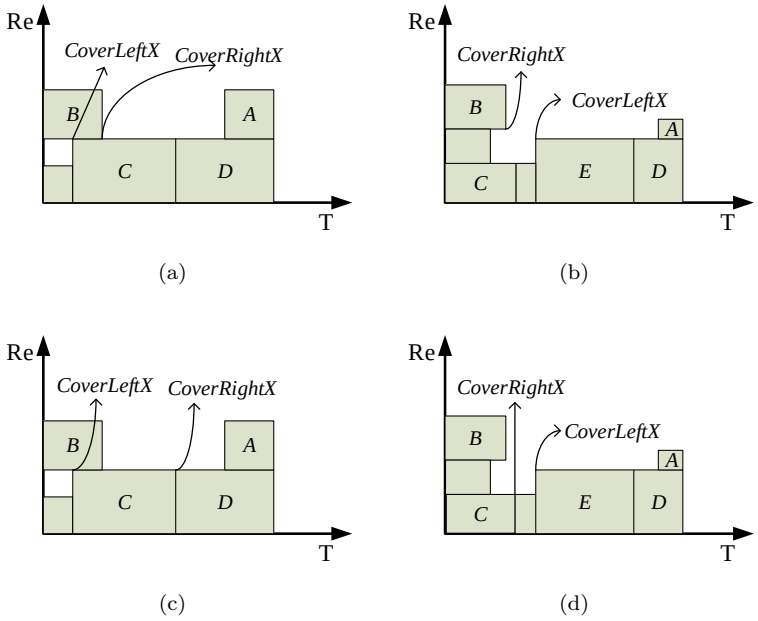

FIGURE 4. Relative positions of $CoverLeftX$ and $CoverRightX$. (a) and (b) are the cases without violating precedence constraints. (c) and (d) are the cases violating precedence constraints.

maximum amount of resource k, we need to eliminate the activities with precedence constraints in each AEA_{ki}. After that, we add up resource usage for each activity in an AEA_{ki} and label it as Q_{ki}. Then the maximum Q_{ki} is the available resource capacity R_k.

Taking into account the makespan and available resource capacity R_k ($k \in K$), the value of fitness of $\boldsymbol{MBS}$ can be defined as follows,

$$f(\boldsymbol{MBS}) = \sum_{k=1}^{\rho} C_k R_k + C_d \times max\{0, S_{n+1} - T\} \tag{11}$$

In FIGURE 5, parts one to four stand for the four designed move rules. The values of (Box^{RT}, Box^{TY}), $BtoT^{//X}$ and $LtoR^{//Y}$ are updated in part five after schedule of a new block in the block list. After all blocks have been placed in the appropriate positions, the fitness of the $\boldsymbol{MBS}$ will be calculated in the end part.

5. MBS$_{\text{MAEA}}$-RIPSP.

Since many previous works show that the MAEA has great potential in solving complex problems [15, 17, 27], we integrated the MBS with MAEA to solve the RIPSPs, which is labeled as MBS$_{\text{MAEA}}$-RIPSP.

5.1. Agents based on MBS.

With the intrinsic properties of the MBS and RIPSPs in mind, the agent in MBS$_{\text{MAEA}}$-RIPSP is defined as follows,

Definition 5.1. An agent, labeled as $\boldsymbol{I}$, represents a candidate solution in the search space $\boldsymbol{F}$, and is encoded using two integer vectors:

$$\boldsymbol{I} = \boldsymbol{MBS} = ((B^{p0}, B^{p1}, \ldots, B^{pn}, B^{pn+1}), (M_0, M_1, \ldots, M_n, M_{n+1}))\ \boldsymbol{I} \in \boldsymbol{F} \tag{12}$$

Algorithm 1 Transforming an ***MBS*** to a schedule

Input: ***MBS***: An ***MBS*** in solution space $\boldsymbol{F}$;
Output: $f(\boldsymbol{MBS})$: The fitness of the ***MBS***;

$(B^{p0}.x_{lb}, B^{p0}.y_{lb}) \leftarrow (0,0)$; /*$B^{p0}$ is a dummy block.*/
$(B^{p1}.x_{lb}, B^{p1}.y_{lb}) \leftarrow (0,0)$; /*Put B^{p1} at the left-bottom corner of the first quadrant.*/
$(Box^{RX}, Box^{TY}) \leftarrow (B^{p1}.width, B^{p1}.height)$;
Add the top edge of B^{p1} into $BtoT^{//X}$;
Add the right edge of B^{p1} into $LtoR^{//Y}$;
for $(i = 2, i < n, i++)$ **do**
  if $M_i = 0$ **then** /*Only move downward.*/
    $(B^{pi}.x_{lb}, B^{pi}.y_{lb}) \leftarrow (max\_t, Box^{TY})$;
    Calculate $CoverTopY$ for B^{pi} from $BtoT^{//X}$;
    $B^{pi}.y_{lb} \leftarrow CoverTopY$;
  else if $M_i = 1$ **then** /*Only move leftward.*/
    $(B^{pi}.x_{lb}, B^{pi}.y_{lb}) \leftarrow (Box^{RX}, 0)$;
    Calculate $CoverRightX$ for B^{pi} from $LtoR^{//Y}$;
    $B^{pi}.x_{lb} \leftarrow CoverRightX$
  else if $M_i = 2$ **then** /*Two cases are considered according to whether violating precedence constraints or not.*/
    if $max\_t \geq (Box^{RX} - B^{pi}.width)$ **then** /*Violating precedence constraints, move downward only.*/
      $(B^{pi}.x_{lb}, B^{pi}.y_{lb}) \leftarrow (max\_t, Box^{TY})$;
      Similar to case 0;
    else /*Do not violate precedence constraints, repeat downward and leftward movement.*/
      $(B^{pi}.x_{lb}, B^{pi}.y_{lb}) \leftarrow (Box^{RX} - B^{pi}.width, Box^{TY})$;
      $CanMove \leftarrow true$;
      while $(CanMove = true)$ **do**
        Calculate $CoverTopY$ for B^{pi} from $BtoT^{//X}$;
        $B^{pi}.y_{lb} \leftarrow CoverTopY$;
        Calculate $CoverRightX$ for B^{pi} from $LtoR^{//Y}$;
        Calculate $CoverLeftX$ for B^{pi};
        if $(CoverRightX \geq CoverLeftX)$ or $((CoverRightX < CoverLeftX)$ and $((CoverLeftX - CoverRightX) < B^{pi}.width))$ **then**
          $B^{pi}.x_{lb} \leftarrow CoverRightX$;
          $CanMove \leftarrow false$
        else
          $B^{pi}.x_{lb} \leftarrow CoverLeftX - B^{pi}.width$;
          $CanMove \leftarrow true$;
        end if
      end while
    end if
  else if $M_i = 3$ **then** /*Repeat leftward and downward movement.*/
    $(B^{pi}.x_{lb}, B^{pi}.y_{lb}) \leftarrow (Box^{RX}, max(0, Box^{TY} - B^{pi}.height))$;
    $CanMove \leftarrow true$;
    while $(CanMove = true)$ **do**
      Calculate $CoverRightX$ for B^{pi} from $LtoR^{//Y}$;

$B^{pi}.x_{lb} \leftarrow CoverRightX$;
Calculate $CoverTopY$ for B^{pi} from $BtoT^{//X}$;
$B^{pi}.y_{lb} \leftarrow CoverTopY$;
if $B^{pi}.x_{lb} = max\_t$ **then**
  break;
else
  Calculate $CoverBottomY$ for B^{pi};
end if
if $(CoverBottomY \leq CoverTopY)$ or $((CoverBottomY > CoverTopY)$ and $((CoverBottomY - CoverTopY) < B^{pi}.height))$ **then**
  $B^{pi}.y_{lb} \leftarrow CoverTopY$;
  $CanMove \leftarrow false$;
else
  $B^{pi}.y_{lb} \leftarrow CoverBottomY - B^{pi}.height$;
  $CanMove \leftarrow true$;
end if
end while
end if
Update (Box^{RT}, Box^{TY});
Add the top edge of B^{pi} into $BtoT^{//X}$, and keep the order of $BtoT^{//X}$;
Add the right edge of B^{pi} into $LtoR^{//Y}$, and and keep the order of $LtoR^{//Y}$;
end for
Calculate available resource capacity R_k for each kind of resource;
Calculate $f(\boldsymbol{MBS})$;

For each agent $\boldsymbol{I}$, we can get a makespan and an available resource capacity for each kind of resource. The energy of $\boldsymbol{I}$ is equal to the negative value of its

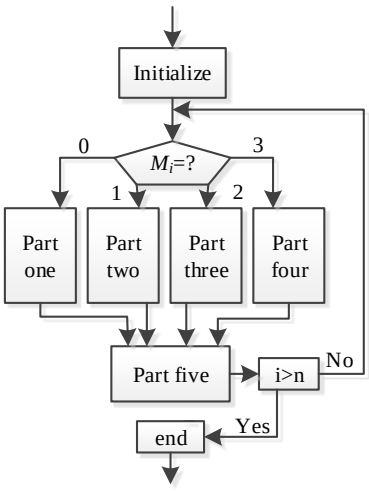

FIGURE 5. The decoding process.

associated objective function value, namely $Energy(\boldsymbol{I}) = -f(\boldsymbol{I})$. The purpose of $\boldsymbol{I}$ is to increase its energy as much as possible by the behaviors it can take.

As described in [15, 17, 27], all agents in MBS$_{\text{MAEA}}$-RIPSP live in a lattice-like environment, which is called the agent lattice. Each agent is fixed on a lattice point and can only interact with its neighbors. The agent lattice can be represented as FIGURE 6. In this paper, we define the size of agent lattice as $R_{size} \times C_{size}$. Each agent has eight neighbors. Suppose that agent $\boldsymbol{I}$ locates at (i, j), then the sets of eight neighborhood domains are described as follows. For example, in FIGURE 6, the agent in position (2, 2) is connected to its eight neighbors with imaginary lines.

$$neighbors(\boldsymbol{I}) = \{(i', j), (i', j'), (i, j'), (i'', j'), (i'', j), (i'', j''), (i, j''), (i', j'')\} \quad (13)$$

where

$$\begin{aligned} i' &= \begin{cases} i-1 & i \neq 1 \\ R_{size} & i = 1 \end{cases}, \; j' = \begin{cases} j-1 & j \neq 1 \\ C_{size} & j = 1 \end{cases} \\ i'' &= \begin{cases} i+1 & i \neq R_{size} \\ 1 & i = R_{size} \end{cases}, \; j'' = \begin{cases} j+1 & j \neq C_{size} \\ 1 & j = C_{size} \end{cases} \end{aligned} \quad (14)$$

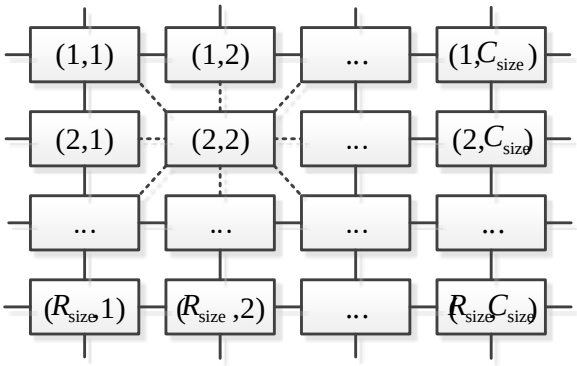

FIGURE 6. The agent lattice of MBS$_{\text{MAEA}}$-RIPSP

5.2. **Operators for agents.** In MBS$_{\text{MAEA}}$-RIPSP, four operators are employed to explore the search space. In addition to the crossover and mutation operators, the competition and self-learning operators are designed for agents to gain more energy.

5.2.1. *Crossover.* In MBS$_{\text{MAEA}}$-RIPSP, two-point crossover based on MBS representation is designed. Assume this operator conduct on two parents $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, where $\boldsymbol{p}_2$ is the agent that with the largest energy among the neighbors of $\boldsymbol{p}_1$. Since there are precedence constraints among activities, one of the significant properties of the crossover operator should be maintaining the precedence constraints, in other words, the children produced by crossover operator should be feasible. Therefore, the two-point crossover operator introduced by Hindi *et al.* in [10] is employed to the activity list. As for the move mode list, the general two-point crossover operator is utilized.

5.2.2. *Mutation.* The mutation operator is designed to reintroduce diversity to the population. It is conducted both on the activity list and the move mode list. For the part of the activity list, the classical mutation operation introduced in [10] is employed, in which an activity is moved randomly between the position of its last predecessor and first successor according to the temporary order of the activity list. Thus the newly generated activity list is still feasible.

As mentioned before that all activities move in the first quadrant and the objective of RIPSPs is to minimize the summation of the project makespan and the resource capacity. Moreover, the position that an activity will be placed can be partially decided by the move modes. Therefore, according to whether the makespan of a project exceed the due date or not, a direction-based mutation operator is designed for the move mode list. If the project makespan is larger than the due date, then our goal is to increase the upper bounds of resources, i.e., the available resource capacities R_k $(k \in K)$, while decreasing the makespan. Otherwise, our goal is to increase the makespan while decreasing the upper bound of resource. The details of the mutation operator designed for the move mode list are given in **Algorithm 2**.

In **Algorithm 2**, if the makespan of the project exceed the due date, the decreasing of the total makespan is desired. Thus, the move modes of the predecessors and the successors of the randomly chosen activity v are set to 0 and 2 respectively, which means that the initial positions of these activities preferred to be in the top edge of the placed blocks, and according to the related move rules, these activities will be placed within Box^{RX}, which guarantees the decreasing of the total makespan. Otherwise, the move modes of predecessors and successors of activity w are set to 3, in which the positions of these activities are initialized to the left edge of the placed blocks, which guarantees the decreasing of the available resource capacities.

5.2.3. *Competition.* For an agent $\boldsymbol{I}$ on the agent lattice, an eight-neighbor competition operator is designed. The comparison between $\boldsymbol{I}$ and its strongest neighbor $\boldsymbol{I}'$, which has the maximum energy among all eight neighbors, is conducted. If $Energy(\boldsymbol{I}) > Energy(\boldsymbol{I}')$, $\boldsymbol{I}$ is a winner and it can survive on the agent lattice, otherwise, it is a loser and will be occupied by agent $\boldsymbol{I}'$. After the competition operator is conducted, agents that with low energy are replaced by better agents, which generally speaking will increase the total energy of the whole population. Moreover, by replacing the low-energy agents, more opportunities can be given to the better agents, which to some extent can schedule the computational efforts more appropriately.

5.2.4. *Self-Learning.* It is well-known that the incorporation with local search can accelerate the convergence speed of EAs. Therefore, the self-learning operator is designed as a local search to further increase the energy of an agent. From above we can know that the mutation operator is self-adaptive to some extent, thus it is employed in self-learning operator to generate alternative agents.

In self-learning operator, the designed mutation operator will be conducted iteratively for $SelfLTime$ times. After each iteration, if the energy of the newly generated agent is larger than the old one, then the old agent will be replaced by the newly generated one; otherwise, no operation will be conducted. Therefore, after performing the self-learning operator on an agent, its energy will be increased as much as possible. In order to reduce the computational cost, the self-learning

Algorithm 2 Mutation operator for move mode list

Input: $\boldsymbol{I}$: An agent;
Output: $\boldsymbol{I}'$: The agent after performing the mutation operator;

if makespan($\boldsymbol{I}$) $> T$ **then**
  Choose an activity v at random from 1 to n;
  Find P_v and S_v;
  for $(i = 0; i < n + 1; i++)$ **do**
    if $s[i] \in P_v$ **then**
      Set the move mode of activity $s[i]$ in agent I' to 0;
    else if $s[i] \in S_v$ **then**
      Set the move mode of activity $s[i]$ in agent I' to 2;
    end if
  end for
else
  Choose an activity w at random from 1 to n;
  Find P_w and S_w;
  for $(i = 0; i < n + 1; i++)$ **do**
    if $s[i] \in P_w$ or $s[i] \in S_w$ **then**
      Set the move mode of activity $s[i]$ in agent I' to 3;
    end if
  end for
end if
Transform the agent $\boldsymbol{I}'$ to a schedule;

operator is only performed on the best agent in each generation. The details of self-learning operator are given in **Algorithm 3**.

Algorithm 3 Self-learning operator

Input: $\boldsymbol{I}$: Agent $\boldsymbol{I}$;
  $SelfLTime$: The maximum number of iterations;
Output: Agent $\boldsymbol{I}$;

for $(i = 0; i < SelfLTime; i++)$ **do**
  $\boldsymbol{I}' \leftarrow$ Mutation($\boldsymbol{I}$);
  if $Energy(\boldsymbol{I}') > Energy(\boldsymbol{I})$ **then**
    $\boldsymbol{I} \leftarrow \boldsymbol{I}'$;
  end if
end for

5.3. **Implementation of MBS$_{\mathbf{MAEA}}$-RIPSP.** In MBS$_{\mathrm{MAEA}}$-RIPSP, the population is initialized firstly, in which the algorithm described in [10] is employed in the initialization of the activity lists and the move mode for every activity is randomly initialized from 0 to 3. Then, the designed decoding algorithm that detailed in **Algorithm 1** is used to generate the feasible schedules and the energy is calculated for every agent. The competition operator is conducted on every agent according to probability P_{com}, after which the agents that with low energy are occupied by their strongest neighbors, as a consequence of increasing the total energy of the

whole population. Afterwards, the crossover and mutation operator are performed for every agent with probability P_{cro} and P_{mut}, respectively. At the end of each generation, the self-learning operator is conducted on the best agent of the current generation with probability P_{sel}. The whole evolution process will stop until reach the stopping criteria. The details of the MBS_{MAEA}-RIPSP are described as **Algorithm 4**.

Algorithm 4 MBS_{MAEA}-RIPSP

Input: G: An AON network;
P_{cro}: Crossover probability;
P_{mut}: Mutation probability;
P_{com}: Competition probability;
P_{sel}: Self-learning probability;
R_{size} and C_{size}: The number of rows and columns of the agent lattice;
MaxGen: The maximum number of generations;
Output: The best agent found;

Initialize the agent lattice;
For each agent, calculate $f(I)$;
repeat
  for $i = 1, 2, \ldots, R_{size}, j = 1, 2, \ldots, C_{size}$ **do**
    Conduct competition operator on agent $\boldsymbol{I}(i, j)$ with probability P_{com};
  end for
  for $i = 1, 2, \ldots, R_{size}, j = 1, 2, \ldots, C_{size}$ **do**
    Conduct crossover operator on agent $\boldsymbol{I}(i, j)$ with probability P_{cro};
  end for
  for $i = 1, 2, \ldots, R_{size}, j = 1, 2, \ldots, C_{size}$ **do**
    Conduct mutation operator on agent $\boldsymbol{I}(i, j)$ with probability P_{cro};
  end for
  if $U(0, 1) < P_{sel}$ **then**
    Conduct the self-learning operator on the best agent in the current generation;
  end if
until (reach the stopping criteria)

6. **Experiments and results.** In this section, Möhring instances [19] and the generated instances J10, J14, J20, which are generated by ProGen software [14], are used to test the performance of MBS_{MAEA}-RIPSP. Möhring instances are a small bridge construction project, consisting of 18 activities and 4 kinds of resources. For each kind of resource, a per unit availability cost is defined, which is 1 or 3. Therefore, the combinations of unit availability costs for the four resource types are 16. The cost combination is labeled as $C_1/C_2/C_3/C_4$. Note that 1/1/1/1 is the same as 3/3/3/3, so the total number of combination is 15.

For the generated instances J10, J14 and J20, there are 10, 14 and 20 non-dummy activities, respectively, and 4 kinds of renewable resources for every instance. Activity durations and resource requirements of the 4 types of resources are obtained using a discrete uniform distribution within [1, 10]. The project networks have three non-dummy start activities and three non-dummy finish activities. The network

complexity NC is set to 1.5, the resource factor RF is 1 and resource strength RS is set to 0.2. The maximum number of predecessors/successors is three. Original resource availabilities of single mode resource constraint project scheduling problem (SMRCPSP), which are generated by ProGen, are used as the unit cost of the corresponding resource type. The tardiness cost, C_d, is considered as 1/4 of the sum of the unit cost of the resources. Each network contains 20 instances.

T is the due date of a project and $T = \theta \times EFT$, where EFT is the earliest finish time of the project having infinite resource capacities, and $\theta \in \{1.0, 1.1, 1.2, 1.3, 1.4, 1.5\}$. Therefore, there are totally $15 \times 6 = 90$ test sets for Möhring instances and $20 \times 6 = 120$ test sets for J12, J16 and J22, respectively. The total number of instances is $90 + 120 + 120 + 120 = 450$.

6.1. **Experiments on Möhring instances.** For Möhring instances, there are totally 90 instances. 20 independent experiments are conducted for each instance with the 1000 times of fitness evaluation as the stooping criterion. The size of agent lattice is set to $R_{size} = 4$, $C_{size} = 4$, and $P_{cro} = 0.95$, $P_{mut} = 0.85$, $P_{com} = 1.0$, $P_{sel} = 0.5$, $SelfLTime = 12$. The percentages of the number of finding the optimal solutions in all replications for each instance are shown in TABLE 1. The percentage value will be 1 if the MBS_{MAEA}-RIPSP can find the optimal solution for all 20 times; otherwise, it will be smaller than 1.

From TABLE 1 we can know that for all 90 Möhring instances, there are 43 instances with the percentage values equal to 1, which means that the proposed MBS_{MAEA}-RIPSP can solve these instances easily. And, there are 25 instances with the percentage values various from 0.5 to 0.95 and 19 instances with the values below 0.5, which means for these instances the MBS_{MAEA}-RIPSP can find the optimal solutions with certain probabilities. However, the rest three instances are relatively difficult that the proposed MBS_{MAEA}-RIPSP cannot find optima in all replications. In one word, the MBS_{MAEA}-RIPSP is able to solve most of the Möhring instances.

In order to further verify the performance of MBS_{MAEA}-RIPSP, the comparison between MBS_{MAEA}-RIPSP and the GA designed by Shadrokh *etal.* in [14] is conducted. According to [14], the numbers of generation are recorded for both two algorithms once the optimal solution is found for each instance. The parameter settings of MBS_{MAEA}-RIPSP are identical to the aforementioned experiment. The results are shown in TABLE 2, in which the bold numbers represent the smaller values of the number of generations and MBS represents the MBS_{MAEA}-RIPSP for convenience. From TABLE 2 we can know that there are 40 instances that the proposed MBS_{MAEA}-RIPSP finds the optimal solutions using less generations, which means it is more effective; 24 instances that two algorithms have the same values of generations, and 26 instances the MBS_{MAEA}-RIPSP needs more generations than GA to find the optimal solutions. Overall, for most Möhring instances the proposed MBS_{MAEA}-RIPSP performs better than GA.

6.2. **Experiments on J10, J14, and J20 instances.** For J10, J14, and J20 test sets, we generated and tested 20 instances for each group, and the due date T is still set to $T = \theta \times EFT$, where $\theta \in \{1.0, 1.1, 1.2, 1.3, 1.4, 1.5\}$. Therefore, totally $3 \times 20 \times 6 = 360$ problems were solved. The self-learning probability P_{sel} is set to 0.5 and the other parameter settings of MBS_{MAEA}-RIPSP for J10, J14 and J20 are shown in TABLE 3, which are selected empirically. The stopping criteria is either the optimal solution is found or the predetermined maximum number of generation is reached. The percentages of finding the optimal solutions for each

TABLE 1. The Percentages of Finding Optimal Solutions for MBS_{MAEA}-RIPSP on Möhring Instances with 1000 Evaluations

$C_1/C_2/C_3/C_4$	$\theta = 1.0$	$\theta = 1.1$	$\theta = 1.2$	$\theta = 1.3$	$\theta = 1.4$	$\theta = 1.5$
1/1/1/1	1.00	1.00	1.00	1.00	0.067	0.033
3/1/1/1	1.00	0.90	0.90	0.80	0.133	0.067
1/3/1/1	1.00	1.00	1.00	0.80	0.700	0.333
1/1/3/1	1.00	1.00	1.00	1.00	0.950	0.067
1/1/1/3	1.00	1.00	1.00	0.60	1.00	0.067
3/3/1/1	1.00	0.50	0.80	0.80	0.00	0.333
3/1/3/1	1.00	1.00	0.90	0.85	0.60	0.333
3/1/1/3	1.00	0.75	0.95	0.75	0.533	0.067
1/3/3/1	1.00	1.00	1.00	1.00	0.70	0.033
1/3/1/3	1.00	1.00	1.00	1.00	0.80	0.00
1/1/3/3	1.00	1.00	1.00	1.00	0.60	0.00
3/3/3/1	1.00	1.00	0.90	1.00	0.333	0.20
3/3/1/3	1.00	0.45	1.00	0.75	0.133	0.033
3/1/3/3	1.00	1.00	0.95	0.85	0.167	0.067
1/3/3/3	1.00	1.00	1.00	1.00	0.70	0.067

TABLE 2. The Comparison of Numbers of Generation to Reach to the Optimal Solutions between MBS_{MAEA}-RIPSP and GA for Möhring Test Sets

$C_1/C_2/C_3/C_4$	$\theta = 1.0$		$\theta = 1.1$		$\theta = 1.2$		$\theta = 1.3$		$\theta = 1.4$		$\theta = 1.5$	
	MBS	GA	MBS	GA	MBS	GA	MBS	GA	MBS	GA	MBS	GA
1/1/1/1	1	1	**1**	2	1	1	**1**	6	**6**	21	14	**1**
3/1/1/1	1	1	6	**1**	**1**	2	**1**	2	**7**	20	27	**1**
1/3/1/1	**1**	2	**1**	33	**1**	4	1	1	**2**	43	13	**1**
1/1/3/1	1	1	1	1	1	1	1	1	**1**	23	**4**	3
1/1/1/3	1	1	1	1	**1**	3	2	**1**	12	**1**	**4**	5
3/3/1/1	1	1	**1**	28	**1**	2	**1**	2	5	**1**	7	**1**
3/1/3/1	1	1	1	1	1	2	1	1	**8**	11	**8**	9
3/1/1/3	**1**	2	**1**	2	**1**	6	2	**1**	9	**8**	10	**2**
1/3/3/1	1	1	1	1	**1**	8	2	**1**	**1**	15	**2**	3
1/3/1/3	1	1	**1**	2	1	1	1	1	8	**6**	20	**1**
1/1/3/3	**1**	3	**1**	3	**1**	2	**1**	2	**5**	37	12	**1**
3/3/3/1	1	1	**1**	2	**1**	2	**1**	2	**9**	14	13	**1**
3/3/1/3	1	1	4	**1**	1	1	3	**2**	8	8	22	**1**
3/1/3/3	**1**	2	**1**	21	1	1	2	**1**	31	**18**	**4**	5
1/3/3/3	**1**	2	**1**	2	**1**	3	3	**1**	19	**3**	49	**23**

TABLE 3. Parameter Settings for J10, J14 and J20 Test Sets

Test Set	#Agent	MaxGen	ExcuteNum	$SelfLTime$	P_{cro}	P_{mut}	P_{com}
J10	20×20	10	10	12	0.95	0.85	0.9
J14	20×20	10	8	12	0.95	0.85	1.0
J20	20×19	10	8	12	0.95	0.85	1.0

group with different θ values "Opt.%" , the percentage of the average deviation "Dev.%", and the number of fitness evaluations "Eva." are shown in TABLE 4. The comparisons between MBS_{MAEA}-RIPSP and GA are shown in TABLE 5, in which the bold numbers mean the better performance in the specific measurement and MBS represents the MBS_{MAEA}-RIPSP for convenience. From TABLE 5 we can see that for all three test sets, MBS_{MAEA}-RIPSP has larger values of "Opt.%" than GA, which means the proposed method has better performance. In terms of "Dev.%", the MBS_{MAEA}-RIPSP outperforms GA only in J20 test set and for J10 and J14, GA has smaller values. Overall, the MBS_{MAEA}-RIPSP has better performance than GA.

TABLE 4. Experimental Results of MBS_{MAEA}-RIPSP for J10, J14 and J20 Test Sets

θ	J10			J14			J20		
	Opt.%	Dev.%	Eva.	Opt.%	Dev.%	Eva.	Opt.%	Dev.%	Eva.
1.0	40.0	6.8896	6988	76.5	0.7970	3719	32.5	4.9126	7135
1.1	41.0	3.4006	7225	63.0	1.0457	4981	35.0	5.4335	7327
1.2	55.0	2.5922	5821	43.125	2.5851	7683	33.0	4.9258	7273
1.3	51.5	2.4822	6170	49.375	2.2975	7369	43.0	2.6173	7188
1.4	56.0	2.8036	6065	47.5	1.9321	6771	43.0	2.0079	7279
1.5	68.5	1.8743	4496	55.0	1.8272	7465	43.0	1.9728	7154

TABLE 5. Comparisons between MBS_{MAEA}-RIPSP and GA for J10, J14 and J20 Test Sets

Test Set	Opt.%		Dev.%	
	MBS	GA	MBS	GA
J10	**52.00**	48.20	3.3404	0.23
J14	**55.75**	40.00	1.7474	0.32
J20	**38.25**	33.33	**3.6445**	4.68

7. **Conclusion.** In this paper, a new MBS representation is first designed to deal with PSPs, and an MAEA based on MBS representation is proposed to solve the single-mode RIPSPs. The decoding method guarantees to generate feasible schedules based on the MBS representation. Therefore both the newly designed representation and the decoding method can be applied to other EAs that designed to solve PSPs.

In experiments, 450 test instances are used to test the performance of the proposed MBS_{MAEA}-RIPSP. TABLE I and II indicate that the proposed method can solve almost all of the Möhring instances and has better performance than GA in terms of the efficiency for finding the optimal solutions. Moreover, the generated test sets J10, J14 and J20 are employed to further verify the ability of MBS_{MAEA}-RIPSP and the experimental results indicate the outperformance of the proposed method.

RIPSP is just one general issue of PSPs and the newly designed MBS representation and the corresponding decoding method can be extended to other PSPs. Therefore, the extending of MBS representation and the decoding method for solving other variations of PSPs will be our future work.

Acknowledgments. This work is partially supported by the Outstanding Young Scholar Program of National Natural Science Foundation of China (NSFC) under Grant 61522311, the General Program of NSFC under Grant 61773300, the Overseas, Hong Kong & Macao Scholars Collaborated Research Program of NSFC under Grant 61528205, and the Key Program of Fundamental Research Project of Natural Science of Shaanxi Province, China under Grant 2017JZ017.

REFERENCES

[1] H. A. Abbass, A. Bender, H. Dam, S. Baker, J. M. Whitacre and R. A. Sarker, Computational scenario-based capability planning, in *Genetic and Evolutionary Computation Conference (GECCO)*, ACM, Atlanta, Georgia, 2008, 1437–1444.

[2] P. Brucker, A. Drexl, R. Möhring, K. Neumann and E. Pesch, Resource-constrained project scheduling: Notation, classification, models, and methods, *European Journal of Operational Research*, **112** (1999), 3–41.

[3] L. T. Bui, M. Barlow and H. A. Abbass, A multi-objective risk-based framework for mission capability planning, *New Mathematics and Natural Computation*, **5** (2009), 459–485.

[4] F. Chicano, F. Luna, A. J. Nebro and E. Alba, Using multi-objective metaheuristics to solve the software project scheduling problem, in *GECCO '11 Proceedings of the 13th annual conference on Genetic and evolutionary computation*, ACM, Dublin, Ireland, 2011, 1915–1922.

[5] S.-H. Cho and S. D. Eppinger, A simulation-based process model for managing complex design projects, *IEEE Trans. Engineering Management*, **52** (2005), 316–328.

[6] D. Debels, B. D. Reyck, R. Leus and M. Vanhoucke, A hybrid scatter search/electromagnetism meta-heuristic for project scheduling, *European Journal of Operational Research*, **169** (2006), 638–653, Feature Cluster on Scatter Search Methods for Optimization.

[7] E. Demeulemeester, Minimizing resource availability costs in time-limited project networks, *Management Science*, **41** (1995), 1590–1598.

[8] B. Depenbrock, T. Balint and J. Sheehy, Leveraging design principles to optimize technology portfolio prioritization, in *2015 IEEE Aerospace Conference*, 2015, 1–10.

[9] A. Drexl and A. Kimms, Optimization guided lower and upper bounds for the resource investment problem, *The Journal of the Operational Research Society*, **52** (2001), 340–351.

[10] K. S. Hindi, H. Yang and K. Fleszar, An evolutionary algorithm for resource-constrained project scheduling, *IEEE Transactions on Evolutionary Computation*, **6** (2002), 512–518.

[11] R. Kolisch, Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation, *European Journal of Operational Research*, **90** (1996), 320–333.

[12] R. Kolisch and S. Hartmann, Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis, *Project Scheduling*, (1999), 147–178.

[13] R. Kolisch and S. Hartmann, Experimental investigation of heuristics for resource-constrained project scheduling: An update, *European Journal of Operational Research*, **174** (2006), 23–37.

[14] R. Kolisch, A. Sprecher and A. Drexl, Characterization and generation of a general class of resource-constrained project scheduling problems, *Management Science*, **41** (1995), 1693–1703.

[15] J. Liu, W. Zhong and L. Jiao, A multiagent evolutionary algorithm for combinatorial optimization problems, *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, **40** (2010), 229–240.

[16] J. Liu, W. Zhong, L. Jiao and X. Li, Moving block sequence and organizational evolutionary algorithm for general floorplanning with arbitrarily shaped rectilinear blocks, *IEEE Transactions on Evolutionary Computation*, **12** (2008), 630–646.

[17] J. Liu, W. Zhong and L. Jiao, A multiagent evolutionary algorithm for constraint satisfaction problems, *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, **36** (2006), 54–73.

[18] L. L. Minku, D. Sudholt and X. Yao, Evolutionary algorithms for the project scheduling problem: runtime analysis and improved design, in *GECCO '12 Proceedings of the 14th annual conference on Genetic and evolutionary computation*, ACM, Philadelphia, Pennsylvania, USA, 2012, 1221–1228.

[19] R. H. Möhring, Minimizing costs of resource requirements in project networks subject to a fixed completion time, *Operational Research*, **32** (1984), 89–120.

[20] H. Nübel, The resource renting problem subject to temporal constraints, *OR-Spektrum*, **23** (2001), 359–381.

[21] C. Qian, Y. Yu and Z.-H. Zhou, Variable solution structure can be helpful in evolutionary optimization, *Science China Information Sciences*, **58** (2015), 112105, 17 pp.

[22] B. D. Reyck and R. Leus, R&d project scheduling when activities may fail, *IIE Transactions*, **40** (2008), 367–384.

[23] S. R. Schultz and J. Atzmon, A simulation based heuristic approach to a resource investment problem (rip), in *Proceedings of the Winter Simulation Conference*, 2014, 3411–3422.

[24] S. Shadrokh and F. Kianfar, A genetic algorithm for resource investment project scheduling problem, tardiness permitted with penalty, *European Journal of Operational Research*, **181** (2007), 86–101.

[25] J. Xiong, J. Liu, Y. Chen and H. A. Abbass, A knowledge-based evolutionary multiobjective approach for stochastic extended resource investment project scheduling problems, *IEEE Transactions on Evolutionary Computation*, **18** (2014), 742–763.

[26] J. Xiong, K. wei Yang, J. Liu, Q. song Zhao and Y. wu Chen, A two-stage preference-based evolutionary multi-objective approach for capability planning problems, *Knowledge-Based Systems*, **31** (2012), 128–139.
[27] W. Zhong, J. Liu, M. Xue and L. Jiao, A multiagent genetic algorithm for global numerical optimization, *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, **34** (2004), 1128–1141.

E-mail address: yuanxiao1127@yeah.net
E-mail address: neouma@163.com
E-mail address: ystar1991@126.com