AN EFFICIENT GENETIC ALGORITHM FOR DECENTRALIZED MULTI-PROJECT SCHEDULING WITH RESOURCE TRANSFERS

JINGWEN ZHANG*, WANJUN LIU AND WANLIN LIU

School of Management
Northwestern Polytechnical University
Xi’an 710072, Shaanxi, China

(Communicated by Gerhard-Wilhelm Weber)

ABSTRACT. This paper investigates the decentralized resource-constrained multi-project scheduling problem with transfer times (DRCMPSPTT) in which the transfer times of the global resources among different projects are assumed to be sequence-independent, while transfers of local resources take no time within a project. First, two decision variables \( y_{ijg} \) and \( w_{ijg} \) are adopted to express the transition state of global resources between projects. \( y_{ijg} \) (takes a value of 0 or 1) represents whether activity \( i \) transfers global resource \( g \) to activity \( j \); accordingly, the transferred quantity is denoted as \( w_{ijg} \). Then, we construct an integer linear model with the goal of minimizing the average project delay for the DRCMPSPTT. Second, an adaptive genetic algorithm (GA) is developed to solve the DRCMPSPTT. To gain the schedules for the DRCMPSPTT, the traditional serial and parallel scheduling generation schemes (SGSs) are modified to combine with different resource transfer rules and to design multiple decoding schemes. Third, the numerical experiments are implemented to analyze the effects of eight decoding schemes, and we found that the scheme comprising the parallel SGS and maxRS rule can make the GA work the best; furthermore, the effectiveness of the GA_maxRS (GA embedded with the best scheme) is demonstrated by solving some instances with different sizes.

1. Introduction. With the development of globalization and information technology, companies sometimes execute multiple projects in different geographical locations simultaneously. These projects simultaneously share the global resources of the enterprise, are independent and have their own exclusive local resources. In project scheduling research, this problem is introduced as the decentralized resource-constrained multi-project scheduling problem (DRCMPSP), which is more complex than the classic resource-constrained project scheduling problem (RCPSP) and the traditional resource-constrained multi-project scheduling problem (RCMPSP) that only contains shared global resources [9]. The global and local resources required for each project’s execution are limited in the DRCMPSP. Therefore, resources are often transferred among the projects and activities, such as the physical transfer of resources from one location to another, or the setup for machines and human resources to prepare for new activities [26, 27]. In practice, these resource transfers

2020 Mathematics Subject Classification. Primary: 90B35, 90C11.

Key words and phrases. Decentralized multi-project scheduling, resource transfer times, genetic algorithm, scheduling generation schemes, resource transfer rules.

* Corresponding author: Jingwen Zhang.
usually take much time, especially when global resources are transferred between projects in different locations. Although some scholars have conducted in-depth studies on the DRCPSP [2, 9, 15, 16], they all assume that resource transfers occur instantaneously. In fact, this assumption is not valid in practice and thus may lead to unrealistic schedules. Therefore, the resources transfer times (especially for global resources) need to be considered in order to ensure the efficient utilization of scarce resources and obtain a feasible and better scheduling scheme in practice [26, 27].

The setup time, as a variant of the transfer time studied in this paper, has been widely studied in machine scheduling [28, 35, 39] and lot sizing [17]. However, the literature concerning setup time in the context of project scheduling is scant. Generally, the setup time is the time required for all preparations to execute an activity [3]. There are two types of setup time considered in the literature: sequence-independent and sequence-dependent setup time [32]. The former depends on the activity and the resource on which the activity will be processed, while the later depends not only on the activity and the resource but also on the sequence of activities processed on this resource. In this article, we focus on the sequence-independent transfer of global resources between projects in different locations, and so the term transfer time is used.

The studies about resource transfer times in project scheduling problems are summarized in Table 1. In single-project scheduling, some studies about transfer times or setup times have been done [3, 18, 23, 32, 36, 38, 43]. In the context of multi-project scheduling, however, the consideration of the resource transfer times is rarely found in the literature to date. Yang and Sum [44, 45] were the first who considered sequence-independent transfer times in the DRCPSP. The authors used a dual-level management structure in which a central resource pool allocates resources to projects and the manager of each project use the allocated resources to schedule the projects. They assumed that resources can only be transferred through the central resource pool, that positive resource transfer times occur only from the central resource pool to projects, and the resource transfers from projects to the resource pool are instantaneous. Mittal and Kanda [33] extended the work of Yang and Sum [44, 45]. They assumed that the resources were not only transferred from project to project but also between resource pool and projects based on the maximum and minimum requirements for a predetermined period of time in the future. According to the assumption, they proposed an integer linear programme for minimizing the penalty/reward for tardy/early projects and the costs of idleness and transferring resources. Moreover, they developed a heuristic procedure and reported the computational results. They concluded that while making resource transfer decisions, a longer period demand should be considered when the objective is to minimize the costs of resource transfers. Based on the classification of resource transfers and the role of resources in these transfers, Kruger and Scholl [26, 27] constructed three types of models, including two models of the resource constrained multi-project scheduling problem with transfer times (RCMPSP-TT) and a model of the resource constrained multi-project scheduling problem with transfer times and costs (RCMPSP-TTC). Furthermore, they developed a heuristic framework based on the priority rule for both the single and multi-project environments. Following Kruger and Scholl [26, 27], Cai and Li [8] constructed the integer programming of the RCMPSP-TT with the goal of minimizing the mean project delay of a multi-project. In addition, they proposed a hybrid genetic algorithm in which the elite population
AN EFFICIENT GENETIC ALGORITHM FOR DRCMPSPTT

Based dual population structure (EPS) and variable neighbourhood search (VNS) operators were introduced for both diversification and intensification considerations to improve the effectiveness. In the case of the DRCMPSPTT, there are few reports on resource transfer times. Adhau and Mittal [1] considered the decentralized resource constrained multi-project scheduling problem with resource transfer (DRCMPSPTT). The authors assumed that the transfer times of global resources are sequence-independent while the transfer of the local resources does not take time. They proposed a novel distributed multi-agent system using auctions based on negotiation (DMAS/RIA) approach for the resource intervals and allocating multiple different types of global resources amongst multiple competing projects.

Table 1. Literature review of the project scheduling with transfer time

| Authors          | Project          | Objective | Transfer times | Algorithm          |
|------------------|------------------|-----------|----------------|--------------------|
| Single           | Multi            | Decentralized multi- | Makespan | Others | sequence-dependent | sequence-independent |
| Kolisch (1995)   | ✓                | ✓         | ✓              | ✓                  | Parallel scheme based heuristic |
| Neumann et al    | ✓                | ✓         | ✓              | ✓                  | Branch-and-bound |
| (2003)           | Vanhoucke (2008) | ✓         | ✓              | ✓                  | Branch-and-bound |
| Mika (2008)      | ✓                | ✓         | ✓              | ✓                  | Tabu search |
| Afshar-Nadjafi et al (2014) | ✓    | ✓    | ✓              | ✓                  | GA |
| Poppenborg (2016) | ✓                | ✓         | ✓              | ✓                  | Tabu search |
| Kadri and Bector (2017) | ✓     | ✓     | ✓              | ✓                  | GA |
| Yang and Sun (1993, 1997) | ✓     | ✓     | ✓              | ✓                  | Computational experiment |
| Mittal and Kanda (2009) | ✓     | ✓     | Cost           | ✓                  | Heuristics |
| Kruger and Scholl (2009, 2010) | ✓     | ✓     | Cost           | ✓                  | Priority-rule based heuristic |
| Cai and Li (2012) | ✓                | APD       | ✓              | ✓                  | Hybrid GA |
| Adhau and Mittal (2013) | ✓     | ✓     | ✓              | ✓                  | DMAS/RIA |
| This research    | ✓                | ✓         | ✓              | ✓                  | GA |

As the extension and innovation of the classic RCPSP and the RCMPSPTT, DRCMPSPTT is also an NP-hard problem. In general, the methods for solving project scheduling problems in the literature are divided into three categories, namely, exact, heuristic, and meta-heuristic methods [34]. Although the exact methods can obtain the optimal solutions to some small-scale problems [7, 11], they can hardly solve the large-sized instances in a reasonable computational time. Hence, several researchers have proposed heuristic methods to solve the RCPSP. Heuristic approaches are primarily based on priority rules, and the most famous methods are the serial schedule generation scheme (SGS) and the parallel SGS [40]. In the literatures, meta-heuristics are the most effective methods to solve the project scheduling problem. Classical metaheuristics, such as genetic algorithm (GA) [41], tabu search [4], simulated annealing [42], and ant systems [31], have been widely used in solving project scheduling problems. In recent years, researchers have proposed different hybrid metaheuristic algorithms to solve complex real-world problems [20, 21, 22, 37]. In addition to the above methods, many scholars have introduced a multi-agent system (MAS) to solve the DRCMPSPTT in recent years [2, 9, 15, 16], and achieved good results. However, such methods still have some disadvantages. An MAS gives
each project agent the right to manage the internal project independently, and the independent competition among projects for global resources easily causes wasted resources.

To make up for the lack of existing research, in this paper, we consider the sequence-independent transfer times of global resources in the DRCMPSP, and the problem is named the decentralized resource constrained multi-project scheduling problem with transfer times (DRCMPSPTT). Considering the shortcomings of an MAS in solving the DRCMPSP, we choose a meta-heuristic algorithm to solve this problem. Through the literature review, it is found that the GA is effective at solving the RCPSPTT and the RCMPSPTT. Therefore, the GA is also chosen to solve the DRCMPSPTT in this paper. There are three main contributions in this research. First, considering the transfer times of global resources in the DRCMPSP, we established the integer linear programming model of the DRCMPSPTT. For small-sized problems, modern optimization software such as CPLEX can be used to get the optimal solution quickly. Second, an adaptive GA with two-point crossover was developed to solve the DRCMPSPTT. We modified the traditional serial and parallel SGS and combined them with four different resources transfer rules into eight decoding schemes. Third, numerical experiments were carried out to show that the decoding scheme that combined the parallel SGS and maxRS rule can make the performance of the GA the best. Moreover, the algorithm comparison based on 40 small-sized and 60 large-sized instances verified the effectiveness of the adaptive GA in solving the DRCMPSPTT. The remainder of the paper is organized as follows. Section 2 provides a detailed problem description and definition. Section 3 gives a mathematical formulation of the DRCMPSPTT. The adaptive genetic algorithm is described in Section 4. The computational experimentation and analysis of results are presented in Section 5. The last Section 6 is devoted to the conclusions and summary of this paper.

2. Problem description. In the DRCMPSPTT, we consider a multi-project consisting of a set of single-project $P = \{1, 2, \ldots, |P|\}$. Each project $p \in P$ has an arrival date $a_{dp} \geq 0$, at which it can be started. Each project $p$ consists of a set $J_p$ of non-preemptable real activities, a dummy start job $s_p$ and a dummy end job $e_p$. The multi-project comprises all project jobs, a global super source $s_0$ and a sink $e_0$, which are all summarized in set $J = \{1, 2, \ldots, n\}$. The global source $s_0$ and the sink $e_0$, as the starting and ending activities of the multi-project, respectively, transfers global resources to projects and collects all resources at the end, respectively. Fig. 1 shows an Aon network for a DRCMPSPTT example.

The finish-to-start precedence relationships with zero-lag exist only between the activities of a single-project. A set of direct predecessors $PC_j$ and a set of direct successors $SC_j$ are given for each job $j \in J$. Several types of renewable global resources $g \in G$ with constant capacity $r_g$ are available for multi-project execution and several types of renewable local resources $l \in L_p$ with constant capacity $r_{pl}$ are available for the execution of each sub-project $p$ in each period. Executing job $j$ requires a constant number of resource units $u_{gj}$ of global resource type $g$ and $u_{lj}$ of local resource type $l$ per period during its duration $d_j$. The transfer time of global resource $g$ from activity $i$ to activity $j$ is represented as $\Delta_{ijg}$. All transfer times are assumed to fulfil the triangular inequality, i.e., $\Delta_{ijg} < \Delta_{ilg} + \Delta_{ljg}, \forall i, j, l \in J$; and the symmetry principle, i.e., $\Delta_{ijg} = \Delta_{jig}, \forall i, j \in J$. As Adhau and Mittal [1] did, we assumed that non-zero times only occur for the global resource transfers.
between different projects. According to Kriger and Scholl [26, 27], transfers that flow from the global source $s_0$ to the global sink $e_0$ directly take no time because these flows absorb redundant resource units and are not executed in reality.

![Diagram of DRCMPSPTT](image)

**Figure 1.** An example of a DRCMPSPTT

All dummy jobs have zero duration and resource usage for each type of resources except the global source $s_0$ and sink $e_0$. These two dummy jobs take no time but require the entire available amount $r_g$ of every global resource $g$ because the global source needs to provide all global resources to the multi-project while the global sink collects them back. The objective of the DRCMPSPTT is to determine the schedules that minimize the average project delay (APD) of the multi-project while satisfying the precedence, global and local resources, and transfer times constraints.

3. **Mathematical formulation.** Kriger and Scholl [27] developed a mixed-integer linear programme of the RCMPSPTT, according to which the model of the DRCMPSPTT can be formulated. In addition to the notations presented above, some parameters and variables are also used and are as follows.

- $T$: upper bound on the multi-project duration
- $t$: index for periods, where $t = 0, 1, \cdots, T$
- $A(p, t)$: all ongoing activities in project $p$ at time $t$
- $PC^*_j$: set of direct and indirect predecessors of job $j \in J$
- $SC^*_j$: set of direct and indirect successors of job $j \in J$
- $EF_j$: earliest finishing time of job $j \in J$
- $LF_j$: latest finishing time of job $j \in J$
- $TI_j$: time window for finishing job $j \in J$, where $TI_j = [EF_j, LF_j]$
- $CP_p$: the critical path time of project $p$
- $R$: set of global resources
- $L_p$: set of local resources of project $p$
- $Jg_j$: set of real jobs (including the global sink) to which resources might be transferred after having performed job $j \in J - \{e_0\}$, where $Jg_j = J - \{s_0\} - \{j\} - PC_j$

The following decision variables are used.

- $x_{jt}$: $1$, if activity $j$ is terminated at the end of period $t$; $0$, otherwise
- $y_{ijg}$: $1$, if resources $g$ is transferred from activity $i$ to $j$; $0$, otherwise
Using these notations, the integer linear programme of the DRCMPSPTT can be expressed as follows:

\[
\textbf{Minimize} \quad \text{APD} = \frac{1}{|P|} \sum_{p \in P} (F_{e_p} - CP_p - \text{ad}_p) \tag{1}
\]

\textbf{Subject to:}

\[
\sum_{t \in TI_j} x_{jt} = 1, \forall j \in J \tag{2}
\]

\[
F_j = \sum_{t \in TI_j} t \cdot x_{jt}, \forall j \in J \tag{3}
\]

\[
\sum_{t \in TI_i} (t - d_j) \cdot x_{jt} - \sum_{t \in TI_i} t \cdot x_{it} \geq 0, \forall i \in PC_j \tag{4}
\]

\[
F_{e_p} \geq \text{ad}_p, \forall p \in P \tag{5}
\]

\[
\sum_{j \in A(p,t)} w_{jt} \leq R_{pl}, \forall l \in l_p, \forall p \in P, \forall t \in T \tag{6}
\]

\[
F_i + \Delta_{ijg} + d_j \leq F_j + T(1 - y_{ijg}), \forall i \in J - \{e_0\}, \forall j \in J_{gi}, \forall g \in G \tag{7}
\]

\[
w_{ijg} \leq \min(u_{ig}, u_{jg}) y_{ijg}, \forall i \in J - \{e_0\}, \forall j \in J_{gi}, \forall g \in G \tag{8}
\]

\[
y_{ijg} \leq w_{ijg}, \forall i \in J - \{e_0\}, \forall j \in J_{gi}, \forall g \in G \tag{9}
\]

\[
\sum_{h \in J - \{i\} - SC_i} w_{hig} = u_{ig}, \forall i \in J - \{s_0\}, \forall g \in G \tag{10}
\]

\[
\sum_{j \in J_{gi}} w_{ijg} = u_{ig}, \forall i \in J - \{e_0\}, \forall g \in G \tag{11}
\]

\[
x_{jt} \in \{0,1\}, \forall j \in J, \forall t \in TI_j \tag{12}
\]

\[
F_j \geq 0, \forall j \in J \tag{13}
\]

\[
y_{ijg} \in \{0,1\}, \forall i \in J - \{e_0\}, \forall j \in J_{gi}, \forall g \in G \tag{14}
\]

\[
w_{ijg} \geq 0, \forall i \in J - \{e_0\}, \forall j \in J_{gi}, \forall g \in G \tag{15}
\]

In the formulation, the objective function 1 minimizes the average project delay (APD) of the multi-project. Constraints 2, 3 and 4 are well-known from the RCPSP formulation, where constraint 2 ensures that each activity is executed without pre-emption and ends within the time window between its earliest and latest finish times, constraint 3 represents the actual finish time of each activity, and constraint 4 is the precedence constraints. Constraint 5 ensures that each project is executed after its arrival date, and constraint 6 represent the local resources limit constraint of each project. Constraint 7 ensures that the transfer time of global resource $g$
is considered when some of its units are transferred from activity $i$ to activity $j$. Constraint 8 implies that if $y_{ijg} = 0$, then $w_{ijg} = 0$. It also ensures that if $y_{ijg} = 1$, $w_{ijg}$ does not exceed either the amount of $g$ available at $i$ or the amount required for $j$. Constraint 9 implies that if $w_{ijg} = 0$, then $y_{ijg} = 0$. Constraints 10 and 11 are the resources-flow conservation constraints. Constraints 12, 13, 14 and 15 indicate the domain of the decision variables of the model.

Since the model of the DRCMPSPTT is an integer linear program, CPLEX can be used to solve the small-scale problems. However, for more realistic large-scale problems, it is difficult to solve them with exact algorithms. In view of the good effect of the GA in solving project scheduling problems, we developed some adaptations of the GA for the DRCMPSPTT.

4. An adaptive GA for the DRCMPSPTT. Introduced by Holland [14], the GA use techniques and procedures inspired by evolutionary biology to solve complex optimization problems. Several selection mechanisms, such as natural selection, crossover and mutation, are applied to recombine existing solutions to obtain new ones and to find an optimal solution. In this section, an adaptive GA is designed to solve the DRCMPSPTT. It starts with an initial population of $Pop$ solutions generated randomly, and then we go more deeply into the schedule generation scheme (SGS). The traditional serial and parallel SGSs are modified and adapt to the new problem. These two adaptive SGSs can be combined with different resource transfer priority rules into different decoding schemes. Next, the GA determines the fitness value of each individual, randomly selects a number of pairs of solutions, and uses a two-point crossover operator to generate two offspring solutions from each selected pair of parent solutions. Afterward, a mutation operation is applied to some of the newly generated solutions. This process is repeated until the stopping criterion is met. The details of our GA are revealed in 4.1-4.5.

4.1. Representation. The representation of an individual is crucial for the performance of the GA. Kolisch and Hartmann [24] distinguished five different schedule representations, from which the activity-list (AL) representation is the most widespread. Furthermore, Hartmann and Kolisch [13] concluded from experimental tests that programs based on the AL representation were superior to other programs. Therefore, the AL representation is used in this paper. To make the AL shorter, we first remove the dummy activities at the beginning and the end of each sub-project in the multi-project, and only keep the two dummy jobs at the beginning and the end of the whole multi-project. Then, all activities are renumbered into a new set $J'$ of indexes $\{1, 2, \cdots, N + 2\}$, where $N$ is the number of all real jobs in the multi-project and the two dummy activities are represented by 1 and $N + 2$. Fig. 2 shows the network obtained by renumbering the multi-project example in Fig. 1.

The solutions of the DRCMPSPTT can be coded by the precedence feasible activity list $\lambda = (a(1), a(2), \cdots, a(N + 2))$ of all the elements in $J'$. This encoding scheme ensures that the priority relation constraints between activities can be satisfied in the serial or parallel decoding process. As shown in Fig. 3, $a(p)$ is the activity ID for position $p$ in sequence $\lambda$, and the first activity $a(1)$ and last activity $a(N + 2)$ of $\lambda$ are always activities 1 and $N + 2$, respectively. Fig. 4 shows an AL of the multi-project example in Fig. 2. For an AL $\lambda$, a unique scheduling scheme can be obtained through a specific SGS, and $p$ is the priority of activity $a(p)$ when decoding the sequence $\lambda$. 
Figure 2. Renumbered network of the example

\begin{tabular}{cccccccccc}
  Position & 1 & 2 & \cdots & p & \cdots & N+1 & N+2 \\
  AL & a(1) & a(2) & \cdots & a(p) & \cdots & a(N+1) & a(N+2) \\
\end{tabular}

Figure 3. Individual representation for the DRCMPSPPT

\begin{tabular}{cccccccccc}
  Position & 1 & 2 & 3 & 4 & 6 & 7 & 8 & 5 & 9 \\
  AL & 1 & 2 & 10 & 3 & 4 & 6 & 7 & 8 & 9 \\
\end{tabular}

Figure 4. A precedence feasible AL of the multi-project example

4.2. Generation of the initial population. The genetic algorithm is started by building an initial population of Pop solutions. As mentioned above, each solution is coded by an activity list. In this paper, all activity lists that make up the initial population are generated randomly within the activity precedence.

4.3. Schedule generation scheme. The SGS translates the activity list $\lambda$ into a schedule $S$. Two different types of SGSs exist in the literature: the serial SGS [19] and the parallel SGS [5]. Kadri and Boctor [18] adapted the traditional serial and the parallel scheduling schemes to solve the RCPSPTT. In this paper, to construct a feasible schedule for the DRCMPSPPT, we use a new adaptation of the serial SGS and the parallel SGS. Before presenting the adaptation of these two schemes, we introduce the following notations.

- $v_{jg}$ Number of units of global resource $g$ still available for activity $j$
- $w_{ijg}$ Number of units of global resource $g$ transferred from activity $i$ to $j$
- $r_{pl}(t)$ The total amount of local resource $l$ of project $p$ used at time $t$
- $Z_{jg}$ Set of activities that can transfer resource $g$ directly to activity $j$
- $M_{jg}$ Set of pairs of activities that can transfer resource $g$ to activity $j$ by breaking existing resource flows.

4.3.1. The adaptive serial SGS. Fig. 5 presents the adapted serial SGS. The sequence $\lambda$ is an input to this procedure. There are four main steps in the procedure of the adapted serial SGS.

Step 1: In each iteration, we select the first non-scheduled activity $j$ in the sequence $\lambda$ and identify the project to which activity $j$ belongs. Then, its earliest
possible start time \( t \) can be determined according to the arrival date of the project \( p \) and the finish times of all direct predecessors of activity \( j \).

Step 2: Determine whether the local resources required by activity \( j \) are sufficient within its duration interval \([t+1, t+d_j]\). If the requirements are met, the next step can be performed; otherwise, \( t \) is increased until sufficient local resources become available.

Step 3: Determine whether the total amount of each global resource that can be delivered to activity \( j \) at time \( t \) meets the requirements of activity \( j \). Global resources can be delivered to activity \( j \) in two ways, either by breaking existing resource flows or by transferring them directly. For a pair \((i,m)\) of scheduled activities, if activity \( j \) can be inserted between them, the resource flow between them can be broken to deliver some units of global resource \( g \) from \( i \) to \( j \), and these resources can be later transferred from \( j \) to \( m \) without delaying activity \( m \), then we say that the activity pair \((i,m)\) can be broken. All such breakable pairs of the activity are identified and added to a set denoted \( M_{jg} \). In addition, all scheduled activities that can deliver at least one unit of global resource \( g \) to activity \( j \) at time \( t \) (or earlier) are identified and added to a set denoted as \( Z_{jg} \), which is called the collection of activities that can transfer resources directly. The maximum amount of global resource \( g \) that can be provided to \( j \) at period \( t \), \( \text{sum}^{19} \), is given by the following:

\[
\text{sum}^{19} = \sum_{i \in Z_{jg}} v_{ig} + \sum_{(i,m) \in M_{jg}} \sum_{(i,m) \in M_{jg}} w_{img}.
\]

If each global resource amount required by activity \( j \) can be satisfied at time \( t \), we can proceed to the next step. Otherwise, we push \( t \) forward by a unit and go back to the second step to check again. Activity \( j \) can be scheduled only when the various local and global resources required are all available.

Step 4: Arrange activity \( j \) to start at the beginning of period \( t \). To obtain smaller project delays, we first use the breakable flows between activity pairs of set \( M_{jg} \) that provide activity \( j \) with global resource \( g \). The activity pairs are selected in descending order of the value of \( w_{img} \), and ties are broken according to the smaller activity ID. If all such flows are used, and more units of resource \( g \) are needed to arrange \( j \), we select activities \( i \) of set \( Z_{jg} \) in ascending order of their transfer time \( \Delta_{ijg} \) (in the following experimental section, four different resource transfer priority rules are used here, respectively), and use the units available from them to provide activity \( j \) with global resource \( g \) until the requirements of activity \( j \) are satisfied. The tie-breaking rule is the same as above.

After completing the above four steps, activity \( j \) has been scheduled. Then, update the relevant variables and sets, and continue the next iteration until a complete schedule is constructed.

4.3.2. The adaptive parallel SGS. The procedure for the adaptive parallel SGS is presented in Fig. 6. Unlike the aforementioned serial SGS, it is possible to schedule several activities at a decision time \( t \). This procedure first identifies the set of eligible activities at each decision time, and then it checks them according to their order in the sequence. The steps for activities to be checked and arranged are basically the same as the serial decoding procedure. The difference is that the parallel scheduling scheme is time-driven; therefore, activity \( j \) that needs to be arranged cannot obtain global resources by breaking the scheduled activity pairs, and it can only obtain global resources that are delivered to it by the scheduled activities directly. When an eligible activity is selected and checked, regardless of whether or not it can be scheduled, it is removed from the current set of eligible activities, and the next
Algorithm 1 The serial scheduling scheme

1: Initialization: $n \leftarrow 0$, $F \leftarrow \emptyset$, $FS \leftarrow \emptyset$

2: $FS \leftarrow \{1\}$ (set of scheduled activities)
3: $w_{ijg} = \theta \forall i \in J \setminus \{1\}, jg \in F \setminus \{1\}$, $\forall g \in G$
4: $\mathbf{r}_i = \mathbf{r}_g = \mathbf{r}_j = \emptyset \forall i \in J \setminus \{1\}, jg \in F \setminus \{1\}$, $\forall g \in G$
5: $\mathbf{r}_1 = 0$, $\forall g \in P$, $\forall \in L_p$, $t = 1, 2, \ldots$

6: Let $\lambda$ be the sequence to transform into a schedule
7: for $n = 2$ to $|\lambda|$ do
8: Let $j$ be the activity in position $n$ in the sequence $\lambda$
9: if $j \notin F$ then
10: $FS = FS \cup \{j\}$, $F_j = \max\{F_i \forall i \in F \setminus \{\lambda\}\}$
11: else
12: Determine the project $p$ to which activity $j$ belongs and the start time $a_{pj}$ of project $p$
13: Let $t = \max\{a_{pj} \forall i \in P\}$
14: while $j \notin FS$ do
15: if $w_{ijg} + a_{pj} \leq R_p$, $\forall g \in \{j + 1, t + d_i\}$, $\forall \in L_p$ then
16: $\mathcal{Z}_j = \emptyset$, $M_j = \emptyset$
17: for $g = 1$ to $|G|$ do
18: if $s = 1$ to $n - 1$ do
19: let $i$ be the activity in position $s$ in the sequence $\lambda$
20: if $w_{ijg} > 0$ AND $F_i + \lambda_{ijg} \leq t$ then $\mathcal{Z}_j = \mathcal{Z}_j \cup \{g\}$
21: for $h = 1$ to $n - 1$ do
22: let $m$ be the activity in position $h$ in the sequence $\lambda$
23: if $w_{ijg} > 0$ AND $F_m + \lambda_{ijg} \leq t \leq F_j - d_{ijg} - \lambda_{ijg}$ then $M_j = M_j \cup \{i, m\}$
24: end if
25: end for
26: end if
27: end for
28: sum$^{\mathcal{Z}_j} = \sum_{g \in \mathcal{Z}_j} w_{ijg} + \sum_{g \in \mathcal{Z}_j} w_{ijg}$
29: end for
30: if $n = n_{ijg}$, $\forall g \in G$ then
31: for $n = 1$ to $|G|$ do
32: Arrange all breakable-flow activity pairs $(i, m) \in M_j$ by the descending order of $w_{ijg}$
33: Arrange all activities $i \notin F_j$ in the descending order of transfer time $\lambda_{ijg}$
34: let $q_{ijg} = w_{ijg}$
35: if $u < 1$ to $|M_j|$ do
36: if $q_{ijg} \leq u_{ijg}$ then
37: $(i, m)$ be the pair in position $u$ in the set $M_j$
38: $x = \min\{w_{ijg}, q_{ijg} - v_{ijg}\}$
39: $w_{ijg} = w_{ijg} - x$, $\lambda_{ijg} = \lambda_{ijg} + x$, $\lambda_{ijg} = w_{ijg} + x$
40: $v_{ijg} = v_{ijg} - x$
41: end if
42: end for
43: $u = u + 1$
44: while $v_{ijg} < q_{ijg}$ do
45: let activity $i$ be the activity in position $b$ in the set $\mathcal{Z}_j$
46: $y = \min\{w_{ijg}, v_{ijg} - r_{ijg}\}$
47: $v_{ijg} = v_{ijg} - y$, $r_{ijg} = r_{ijg} + y$, $w_{ijg} = w_{ijg} + y$
48: $b = b + 1$
49: end while
50: if $FS = FS \cup \{j\}$, $F_j = t + d_j$, $v_{ijg} = r_{ijg} + t + d_j$, $\forall g \in L_p$ then
51: else $t = t + 1$
52: end if
53: end if
54: $t = t + 1$
55: end while
56: end if
57: end while
58: end for

Figure 5. The adaptive serial SGS with transfer times

eligible activity with the highest priority is selected for inspection and scheduling until the current set of eligible activities becomes empty.

If the set of eligible activities is empty, but there are some non-scheduled activities, the procedure goes to the next decision time. The traditional parallel SGS for the classic RCPSP goes to the next decision time by pushing the current moment $t$ to the minimum of the finish time of all ongoing activities (denoted as $t'$). However, in this procedure, we must distinguish different situations to determine to where we should push the current moment $t$. In the current $t$, if an eligible activity $j$ cannot be scheduled simply due to the constraint of the global resource transfer time, then we push the current moment $t$ to the next unit ($t + 1$) because it might be able to be arranged at the next moment. If we directly push the current $t$ to $t'$, it is likely to make the start time of activity $j$ later, thus extending project's duration. In addition, if there are no eligible activities that cannot be scheduled
due to resource transfer time in the current $t$, we can directly push the current $t$ to $t'$. The procedure iterates in decision time points and then ends once all activities are scheduled.

Algorithm 2 The parallel scheduling scheme

1. Initialize $t = 0$, $PS = \{\}$, $F_j = 0$ if $j \in J$
2. $C = \{\}$ (set of activities completed by time $t$)
3. $A = \{\}$ (set of activities going on at time $t$)
4. $E = \{v \in F|PC = C\} \cap \{v \in F\} \in J$, $\Delta_l = \ell$ (set of eligible activities at time $t$)
5. $E1 = \varnothing$, $E2 = \varnothing$
6. $n_{a} = 0$ if $a \in E - \{j\}$, $\forall \in G$
7. $n_{a} = R_{a}$ if $a \in G$ (set of activities going on at time $t$)
8. $r_{A}(t) = 0$ (set of activities on $a \in A$, $\forall \in G$, $t = 1, 2, \ldots, T$)
9. $r_{A}(t) = 0, \forall \in G$, $t = 1, 2, \ldots, T$
10. Let $\lambda$ be the sequence to transform into a schedule
11. Arrange activities in the list $E$ according to their order in $\lambda$
12. while $|PS| < |F|$ do
13. if $|PS| = |F| - 1$ then
14. $PS = PS \cup \{j\}$, $F_j = \max\{F|a \in F - \{j\}\}$
15. else
16. while $E \neq \varnothing$ do
17. Let $i$ be the first activity in the ordered list $E$
18. if $a_i = t_i (r_i) \leq R_{a_i}$ if $a_i \in \{t, 1, t + d_j\}, \forall \in G$ AND $a_{i_j} = r_{a_i} (r_i) \leq R_{a_i}$ if $a_i \in \{t, 1, t + d_j\}, \forall \in G$ then
19. for $g = 1$ to $|E|$ do
20. $Z_g = \varnothing$
21. for $a = 1$ to $|C|$ do
22. Let $i$ be the activity in position $a$ in the set $C$
23. if $a_i > 0$ AND $F_i + \Delta_{a_i} \leq t$ then $Z_g = Z_g \cup \{i\}$
24. end if
25. end for
26. end for
27. if $\sum Z_g \forall \in G$ then
28. for $g = 1$ to $|G|$ do
29. Arrange activities $i$ in $Z_g$ in the ascending order of transfer time $\Delta_{a_i}$
30. $b = 1$
31. while $a_{i_j} = w_{a_i}$ do
32. Let activity $i$ be the activity in position $b$ in the set $Z_g$
33. $x = \max\{w_{a_i} - w_{a_i}\}$
34. $y_{a_j} = x_0 - x, y_{a_j} = y_{a_j} + x_1, n_{a_j} = w_{a_j} + x$
35. $b = b + 1$
36. end while
37. end for
38. $PS = PS \cup \{j\}$, $A = A \cup \{j\}$, $F_j = t + d_j$
39. $r_{a_j} (t) = r_{a_j} (t) + d_j, \forall \in G$
40. $E1 = E1 \cup \{j\}$
41. else
42. $E2 = E2 \cup \{j\}$
43. end if
44. $E = E - \{j\}$
45. end while
46. if $E1 \neq \varnothing$ then
47. $t = t + 1$
48. else $t = \min\{F|a \in A\}$
49. end if
50. $E1 = \varnothing$, $E2 = \varnothing$
51. $A = A - \{a \in AF, t = t\}$
52. $C = C \cup \{\forall \in F, t = t\}$
53. $E = \{a \in F - PS|PC = C\} \cap \{v \in F - PS\} \in J$, $\Delta_l = \ell$
54. Arrange activities in the list $E$ according to their order in $\lambda$
55. end while

4.4. Evaluation and parent selection. Once the initial population has been generated, each individual must be evaluated. Therefore, a fitness value is calculated for each individual. The genetic evolution program then selects the better individuals to enter the next generation according to the selection rule. Roulette selection is used in this study.

In the project scheduling problem, the objective function or its variant is usually regarded as the fitness value for an individual, such as the makespan of a project for the RCPSP or average project delay for the RCMPSP. In this paper, the objective function is to minimize the APD. In roulette selection, individuals with greater fitness are generally more likely to be selected for the next generation. Therefore, we
take the reciprocal of the APD as the fitness value of individuals. Thus, individuals with smaller APDs have a greater chance of being selected.

4.5. Genetic operators. Crossover operation: There are many crossover methods applied to genetic algorithms in the related literatures. The uniform crossover [6] and the k-point crossover operators [12] are suggested to handle scheduling problems. In this study, we modified the two-point crossover operation from the relevant literature [30] to adapt to the representation of the solutions in this paper.

As shown in Fig. 7, M and F represent the selected parent individuals, and D and S are two children individuals generated after crossing M and F. Two crossing points $r_1$ and $r_2$ ($1 < r_1 < r_2 < N + 2$) are generated randomly. These two crossing points divide the sequences of M and D into three parts, and the inheritance of D and S are also divided into three corresponding parts. First, individual D inherits the activities in first part of M (from position 1 to $r_1$). Then, the second part of D (from position $r_1 + 1$ to $r_2$) inherits the activities from F selectively. Checking starting with the activity at the first position of F, if the activity already appears in the first part of D, it will be skipped. Otherwise, it will be placed in the second part of D in turn until the inheritance of the second part is completed. Finally, the third part of D (from position $r_2 + 1$ to $N + 2$) inherits activities from M selectively. Similarly, checking starting with the activity at the position $r_1 + 1$ of M, if the activity already appears in the second part of D, it will be skipped. Otherwise, it will be placed in the third part of D in turn until the inheritance of the third part is completed. In this way, a child individual D is produced, and this individual is still a precedence feasible activity list. The production process of the individual S is opposite to that of D. In the numerical tests of the next section, we use the probability of crossover $P_c = 0.8$.

Mutation operation: Once the new population has replaced the previous one, a mutation operator is applied to each individual. The mutation operation mutates each gene position of every individual according to the probability $P_m$. First, for the position $i$ where activity $j$ is located in an individual, a random number $x$ from a uniform distribution from 0 to 1 is generated. If $x < P_m$, the mutation operation is performed on gene position $i$. Next, position $i_1$ of the last predecessor and position $i_2$ of the first successor of activity $j$ in the sequence are determined, respectively. In the end, we randomly select a new position $i_3$ from the position interval $[i_1, i_2]$ and move activity $j$ from position $i_1$ to position $i_3$. In this way, each gene position of the individual is examined in turn and the mutation is implemented according to the probability of mutation. In addition, the new individual after mutation is still a precedence feasible activity list. In the numerical tests of the next section, we use $P_m = 0.05$. 

![Figure 7. Two-point crossover operator](image-url)
5. Numerical experiments. In this section, we evaluate the performance of the adaptive GA. The numerical experiment consists of three parts. The first part (subsection 5.1) is the comparison and analysis of different decoding schemes. Four resource transfer rules in the literature [26] are combined with the adaptive serial and parallel SGSs, respectively, to obtain eight decoding schemes. These schemes are embedded into the GA, respectively, and their effects are compared by solving the DRCMPSPTT instances. The DRCMPSPTT instances in this part are based on the DRCMPSP problem sets taken from the multi-project scheduling problem library (MPSPLIB). The second part (subsection 5.2) is the experiment for small-sized problems. We embed the best one of the eight decoding schemes into the GA to compare the effects of the adaptive GA and CPLEX for solving small-sized problems. Due to the large scale of the DRCMPSP instances in the MPSPLIB, the small-sized test problems of this part are generated by ProGen and the necessary parameter configurations are carried out. The third part (subsection 5.3) is the experimental test for large-sized problems. Similarly, based on the best decoding schemes, the effects of our adaptive GA and the existing DMAS/RIA algorithm in the literature [1] are compared. The test instances are the same as those in the first part. The numerical experiments are carried out on a personal computer with an Intel Core i7 (3.20 gigahertz) processor, and 8 gigabytes of RAM. In addition, the adaptive GA is coded and compiled in MATLAB 2015.

5.1. Experiments for decoding schemes. In this subsection, we describe four resource transfer rules in detail first. Then, we introduce the basic information of the DRCMPSP problem sets in the MPSPLIB and explain how to configure the resource transfer time parameters to build the DRCMPSPTT instances. Furthermore, we compare the experimental results of the GA based on different scheduling schemes.

5.1.1. Resource transfer rules. In the adaptive serial and parallel SGSs, when an activity $j$ can be scheduled at time $t$, determination must be made regarding which activities in $Z_{jg}$ (the collection of activities from which resource $g$ can be transferred directly to activity $j$) are chosen to deliver global resource $g$ to $j$. This determination is clearly necessary when the total amounts of $g$ provided by all the activities in $Z_{jg}$ exceed the requirement of $j$. To determine which activities to use to transfer global resource $g$ to $j$, all activities in $Z_{jg}$ are sorted by a priority rule called the resource transfer rule. This paper selects four priority rules in the literature [26]. As shown in Table 2, the first two are time-oriented while the last two are resource-oriented. The minTT rule selects an activity $i$ with the lowest transfer time $\Delta_{ijg}$ first. The minGAP rule prefers activities with the minimal gap between the earliest delivery time and the start time $t$ of $j$. The RS rule sorts the activities in $Z_{jg}$ according to their stocks of idle resources after task execution, that is, the number of resource units that have not been transferred to other activities. minRS and maxRS select the activity with the minimum and maximum stocks first, respectively.

5.1.2. Instances generation. The 60 DRCMPSP instances from the MPSPLIB generated by Homberger [15, 16] are adopted to conduct the numerical experiments, which are available on a public web portal (http://www.mpsplib.com). The information about these instances is given in Table 3. The detailed procedure for generating the instances is given in [16], and we just explain it briefly here.

These 60 test examples come from 12 different types of sets, each of which contains 5 examples of the same type. The multi-project instances consist of 2, 5, 10 and 20 examples of the classic RCPSP (a single project) generated by Kolisch
Table 2. Resource transfer rules

| Transfer rules | Priority value $\pi_i$ | Extremum |
|----------------|------------------------|----------|
| minTT          | $\pi_i = \Delta_{ijg}$ | min      |
| minGAP         | $\pi_i = gap_{ijg} = t - (F_i + \Delta_{ijg})$ | min      |
| maxRS          | $\pi_i = v_{ig} = u_{ig} - \sum_{l \in PS} w_{ilg}$ | max      |
| minRS          | $\pi_i = v_{ig} = u_{ig} - \sum_{l \in PS} w_{ilg}$ | min      |

and Sprecher [25], and a single project contains 30, 90, and 120 real activities, respectively. Each test example contains four kinds of renewable resources, where the global resource set is $G$ and the local resource set of project $p$ is $L_p$. Global resources are randomly selected from four renewable resources. If one of them is a global resource, the remaining three are local resources. The Utilization Factor $UF$ indicates the maximum tightness of the constraints on each required global resource [10]. $UF$ is computed according to Eqs. 16 and 17, and the $GCP$ represents the global critical path.

$$UF = \max_{g \in G} UF_g$$  \hspace{1cm} (16)$$

$$UF_g = \sum_{j \in J} u_{jg} / (r_g * GCP)$$  \hspace{1cm} (17)$$

Table 3. The instances for the DRCMPSP

| Problem subset | NOI | Characterization per instance | $UF_{AV}$ |
|----------------|-----|-------------------------------|-----------|
|                | $|P|$ | $|J_p|$ | $(|G|; |L_p|)$   |           |
| MP30.2         | 5   | 2 30                           | (1;3)/(2;2)/(3;1) | 0.84      |
| MP90.2         | 5   | 2 90                           | (1;3)/(2;2)/(3;1) | 0.57      |
| MP120.2        | 5   | 2 120                          | (1;3)/(2;2)/(3;1) | 1.31      |
| MP30.5         | 5   | 5 30                           | (1;3)/(2;2)/(3;1) | 0.82      |
| MP90.5         | 5   | 5 90                           | (1;3)/(2;2)/(3;1) | 0.61      |
| MP120.5        | 5   | 5 120                          | (1;3)/(2;2)/(3;1) | 1.32      |
| MP30.10        | 5   | 10 30                          | (1;3)/(2;2)/(3;1) | 2.38      |
| MP90.10        | 5   | 10 90                          | (1;3)/(2;2)/(3;1) | 1.14      |
| MP120.10       | 5   | 10 120                         | (1;3)/(2;2)/(3;1) | 1.91      |
| MP30.20        | 5   | 20 30                          | (1;3)/(2;2)/(3;1) | 3.37      |
| MP90.20        | 5   | 20 90                          | (1;3)/(2;2)/(3;1) | 0.9       |
| MP120.20       | 5   | 20 120                         | (1;3)/(2;2)/(3;1) | 0.87      |

NOI no. of instances, $|P|$ no. of projects, $|J_p|$ no. of real activities in project $p$, $|G|$ no. of global resources, $|L_p|$ no. of local resources in project $p$, and $UF_{AV}$ average utilization factor.

According to Lova and Tormos [29], a value of $UF$ less than 1 indicates a low to medium overload while a value greater than 1 indicates a medium to high overload. In Table 3, $UF_{AV}$ represents the average $UF$ values of a subset. The other data of these instances are all given in the data file of the MPSPLIB, including the arrival time of each subproject, the precedence among the activities in a subproject, an
activity’s duration and demands for various resources, and the availability of each kind of resource per period.

Since there is no resource transfer time information in these DRCMPSP instances, we must generate the transfer time parameters of the global resources to build the DRCMPSPTT instances. As Adhau and Mittal [1] did, we assume the transfer times are equal for all global resource type. In addition, the global resource transfer times among different projects obey a uniform distribution \( U \left[0, 10\right] \) and fulfil the triangular inequalities and symmetry principle.

5.1.3. Results and analysis. Four kinds of resource transfer rules and two kinds of SGSs are combined into eight decoding schemes, which are embedded into the GA to solve the 60 DRCMPSPTT instances in 12 sets, and the average value of the APD of 5 instances in each set is taken as the APD of the set.

Table 4 shows the results obtained by the adaptive GA embedded with different schemes with 5000 visited solutions. By comparing the solution results of these 12 sets, it can be seen that when using the adaptive serial SGS, the minGAP rule performs the best, and the minRS rule performs the worst. Accordingly, the maxRS rule performs the best and minRS rule performs the worst when using the adaptive parallel SGS. Therefore, minRS obtain the worst results among the four resource transfer rules no matter which SGS is used. In addition, Fig. 8 shows the performance of the eight combination schemes on the 12 sets. Obviously, the results of the adaptive parallel SGS are much better than those of the adaptive serial SGS, regardless of which resource transfer rule is used.

**Table 4. APDs obtained by the adaptive GA with different schemes**

| Problem subsets | Serial | parallel |
|-----------------|--------|----------|
|                 | minTT | minGAP | maxRS | minRS | minTT | minGAP | maxRS | minRS |
| MP30_2          | 14.8  | 13.7   | 14.8  | 16.1  | 13.9  | 13.3   | 13.7  | 14    |
| MP90_2          | 10    | 8.8    | 10.1  | 11.3  | 6.9   | 6.8    | 7.1   | 7.3   |
| MP120_2         | 86.3  | 83.6   | 85.1  | 92.3  | 57.2  | 56.7   | 56.6  | 57.2  |
| MP30_5          | 34.24 | 34.2   | 36.44 | 39.04 | 23.16 | 23.28  | 22.48 | 23.36 |
| MP90_5          | 23.96 | 21.76  | 22.92 | 25.44 | 12.4  | 12.4   | 11.76 | 13    |
| MP120_5         | 104.04| 104.44 | 106.56| 112.64| 67.72 | 68.32  | 67.56 | 67.8  |
| MP30_10         | 131.78| 126.4  | 134.3 | 145.34| 85.1  | 85.22  | 83.48 | 87.62 |
| MP90_10         | 104.14| 106.72 | 108.96| 114.3 | 61.34 | 60.4   | 61.12 | 62.1  |
| MP120_10        | 252.43| 254.44 | 256.73| 262.54| 164.38| 166.78 | 166.84| 171.96|
| MP30_20         | 333.61| 340.69 | 338.15| 352.6 | 205.06| 208.3  | 203.74| 213.16|
| MP90_20         | 65.24 | 65.04  | 66.76 | 72.84 | 44.16 | 43.95  | 44.06 | 44.49 |
| MP120_20        | 71.28 | 70.49  | 71.48 | 77.56 | 47.48 | 48.28  | 47.24 | 47.52 |
| Average         | 102.65| 102.52 | 104.36| 110.17| 65.73 | 66.14  | 65.47 | 67.46 |

Furthermore, in order to compare the adaptability of the adaptive serial and parallel SGSs in solving cases with different \( UF \) s, we used serial-minGAP and parallel-maxRS to represent the best schemes based on the serial and parallel schemes, respectively, and compared the results obtained by the two schemes. Fig. 9 shows the results of the two schemes and their deviation as \( UF_{AV} \) increase. It can be seen that, overall, the deviation of the results obtained by the two schemes tends to increase as \( UF_{AV} \) increase. Therefore, when the scale and \( UF \) of the problem are large, the parallel SGS can better solve the DRCMPSPTT than the serial SGS.
From these comparisons and analyses, it can be concluded that when solving the DRCMPSPTT with the GA, the adaptive parallel SGS with the maxRS rule is the best scheme compared with the others, and it has good adaptability to different sizes and $UF_s$ of instances.

5.2. Experiments for small-sized problems. To evaluate the effect of the adaptive GA, the adaptive parallel SGS with the maxRS rule is embedded into the GA and named GA$\_maxRS$. Since the problem model established in this paper is an integer programming, optimization software such as CPLEX can be used to get the optimal solutions for small-scale problems. Therefore, the results of GA$\_maxRS$ and CPLEX for solving small-sized problems are compared.

5.2.1. Small-sized problems and comparing measures. The small-sized problems used in this experiment are generated by the project scheduling problem generator ProGen [25]. The problems generated by ProGen are RCMPSPs, and the relevant parameter configurations are shown in Table 5. There are 4 values for the number of real jobs in a single project ($J = 5, 8, 10$ or $14$), and 2 values for the number of single projects contained in a multi-project ($P = 2$ or $3$); therefore, 8 combinations can be obtained. Five instances are generated for each combination, and so a total of 40 instances can be obtained. Each instance contains four types of resources ($K = 4$). We assume that two of them are global resources and the others are local resources,
thus transforming the generated RCMPSP into the DRCMPSP. The duration $d_j$ and the resources requirements $r_j$ of activity $j$ obey uniform distributions of $[1, 5]$ and $[1, 8]$, respectively. The network complexity, resource factor and resource strength of the instances are 1.5, 0.25, and 0.2, respectively. In addition, there is no resource transfer time information in these instances. Therefore, according to the value range of the activity duration, we assume that the global resource transfer time between different projects is 2 and thus construct 40 DRCMPSPTT instances for the algorithm testing.

**Table 5. Parameter configurations to generate test problems**

| Parameter | $J$ | $P$ | $K$ | $r_j$ | $d_j$ | NC | RF | RS |
|-----------|-----|-----|-----|-------|-------|----|----|----|
| value     | 5, 8, 10 or 14 | 2 or 3 | 4 | $[1, 8]$ | $[1, 5]$ | 1.5 | 0.25 | 0.2 |

The 40 DRCMPSPTT instances are solved by GA$_\text{maxRS}$ and CPLEX, respectively. To reduce the impact of the variance, GA$_\text{maxRS}$ is run 10 times for each instance. Because of the small size of the instances, GA$_\text{maxRS}$ only searches for 100 solutions in each run. Then, the best, worst, and mean values of the objective function in 10 runs for each problem are recorded, as well as the average computational times for solving the problem. In addition, to make the comparison fairer, CPLEX is also run 10 times for each instance. Note that the optimal objective function value obtained by CPLEX in all the 10 runs is unique for an instance.

In this subsection, three types of percentage relative errors (PREs) are used to evaluate the performance of the algorithm. They are defined in Eqs. 18, 19, 20 to quantify the gap between the objective function values provided by GA$_\text{maxRS}$ and CPLEX. In Eqs. 18, 19 and 20, $W_{\text{obj}}$, $B_{\text{obj}}$ and $M_{\text{obj}}$ represent the worst, best and mean values of the objective function that obtained by GA$_\text{maxRS}$ respectively, and opt is the optimal objective function value determined by CPLEX. In addition, the average computational time ($ct_{\text{ave}}$) in seconds needed for each method to solve a problem is another performance measure.

\[
P_{\text{PRE}_w} = \frac{W_{\text{obj}} - \text{opt}}{\text{opt}} \times 100\tag{18}
\]

\[
P_{\text{PRE}_b} = \frac{B_{\text{obj}} - \text{opt}}{\text{opt}} \times 100\tag{19}
\]

\[
P_{\text{PRE}_m} = \frac{M_{\text{obj}} - \text{opt}}{\text{opt}} \times 100\tag{20}
\]

5.2.2. Results and analysis. The results of the two methods solving the 40 DRCMPSPTT instances are shown in Table 6, and their performances are graphically depicted in Fig. 10. From Table 6, it is clear that CPLEX can solve these small-sized instances in a reasonable amount of time, which indicates that the integer linear programming model of the DRCMPSPTT established in Section 3 is effective. Compared with the optimal function value obtained by CPLEX, it can be seen that in the 10 runs of GA$_\text{maxRS}$, only two of the 40 instances failed to obtain the optimal solution, and the corresponding $P_{\text{PRE}_b}$ are 25% and 1%, respectively. As seen from Fig. 10. (a), when the scale of the instance is very small, the results of GA$_\text{maxRS}$ in 10 runs are very stable. As the scale of the instance increases,
the randomness of GA\textsubscript{maxRS} will reduce the stability of the solution results, and the corresponding \( \text{PRE}_w \) and \( \text{PRE}_m \) can reach 50\% and 38\% at most, respectively. Nevertheless, GA\textsubscript{maxRS} can obtain the optimal solution for most small-sized instances by running the algorithm many times to reduce the influence of randomness. On the whole, the results indicates that GA\textsubscript{maxRS} has a good effect at solving small-sized problems.

Table 6. Results of the two methods

| instances | CPLEX \( \text{opt} \) \( ct_{av}(s) \) | GA\textsubscript{maxRS} (100 solutions) | \( \text{PRE}_w \) \( \text{PRE}_b \) \( \text{PRE}_m \) |
|-----------|-----------------|-----------------|-----------------|-----------------|-----------------|
| J5\textsubscript{2} | 3 0.08 | 3 3 3 0.11 | 0 0 0 | \cell{3} {2} {1} |
| J8\textsubscript{2} | 3 2.5 0.08 | 2.5 2.5 2.5 0.2 | 0 0 0 | \cell{3} {2} {1} |
| J10\textsubscript{2} | 3 10.5 2.23 | 10.5 11 1.06 14 0 5 | \cell{3} {2} {1} |
| J14\textsubscript{2} | 3 4.5 1.91 | 5.5 4.5 4.7 1.18 22 0 4 | \cell{3} {2} {1} |

In terms of the computation time, as shown in Fig. 10. (b), the instances with very small sizes can be solved quickly by CPLEX. However, as the problem size...
increases, the computation time of CPLEX increases rapidly. When solving an instance \( (J14-3) \) containing 42 real jobs, the computation time reached 3600 s. For larger scale problems, it is difficult to obtain the optimal solutions by CPLEX in a reasonable time. In contrast, all these small-sized instances can be solved by GA\(_{\text{maxRS}}\) in a very short time, and the computation time increases slowly as the scale of the problem increases. The computation time is less than 10 s when solving an example with 42 real jobs. The comparison of the computation time indicates that GA\(_{\text{maxRS}}\) is far superior to CPLEX regarding its efficiency and adaptability to the problem size.

![Figure 10](image)

**Figure 10.** (a) The three types PREs, and (b) computation times of the solution methods

### 5.3. Experiments for large-sized problems.

#### 5.3.1. Large-sized problems and comparing measures.

To further illustrate the effect of the adaptive GA in solving the DRCMPSPTT, the numerical experiments in this subsection are carried out based on the 60 large-sized DRCMPSPTT instances constructed in subsection 5.1. Since CPLEX cannot solve these large-sized problems in a reasonable time, we compare the results obtained by GA\(_{\text{maxRS}}\) with those obtained by the DMAS/RIA algorithm \([1]\). Although the transfer times in both this paper and the literature \([1]\) obey a uniform distribution of \([0, 10]\), we cannot guarantee that the transfer times we generated are exactly the same as those generated in the literature \([1]\). Therefore, the results of the 60 instances without transfer times obtained by the DMAS/RIA and GA\(_{\text{maxRS}}\) are taken as the benchmark for comparisons.

Due to the large scale of the problem, GA\(_{\text{maxRS}}\) is run 30 times for each instance, and 5000 solutions are searched for each run. The best objective function value (average project delay, APD) obtained in 30 runs is taken as the result of the example. The mean value of the results of the 5 instances in each problem set is taken as the result of the set. According to the solution results of the 12 problem sets, three types of PREs are used to evaluate the performance of the algorithm, and they are defined in Eqs. 21, 22 and 23. In Eq. 21, \( \text{APD}_{\text{GA}_{\text{maxRS}}}^{\text{TT}} \) and \( \text{APD}_{\text{GA}_{\text{maxRS}}} \) represent the APD obtained by GA\(_{\text{maxRS}}\) with and without transfer times, respectively. The \( \text{PRE}_{\text{GA}} \) quantifies the gap between the objective function values provided by GA\(_{\text{maxRS}}\) in two different scenarios. In Eq. 22, \( \text{APD}_{\text{DMAS/RIA}}^{\text{TT}} \) and \( \text{APD}_{\text{DMAS/RIA}} \) denote the APD computed by DMAS/RIA with and without transfer times, respectively. \( \text{PRE}_{D} \) quantifies the gap between
the APD provided by DMAS/RIA in the two scenarios. In Eq. 23, \( PRE_A \) quantifies the gap between the APD provided by the two different algorithms without transfer times.

\[
PRE_{GA} = \frac{APD_{GA,maxRS}^{TT} - APD_{DMAS/RIA}^{TT}}{APD_{GA,maxRS}} \times 100 \quad (21)
\]

\[
PRE_D = \frac{APD_{DMAS/RIA}^{TT} - APD_{DMAS/RIA}}{APD_{DMAS/RIA}} \times 100 \quad (22)
\]

\[
PRE_A = \frac{APD_{GA,maxRS} - APD_{DMAS/RIA}}{APD_{DMAS/RIA}} \times 100 \quad (23)
\]

5.3.2. Results and analysis. Table 7 shows the APD of the 12 sets computed by GA\(_{maxRS}\) and DMAS/RIA with different scenarios, and the performances of the two algorithms are compared in Fig. 11. On the whole, when the resource transfer time is ignored, the APD obtained by GA\(_{maxRS}\) is less than those obtained by DMAS/RIA. It can also be seen from Fig. 11 (b) that \( PRE_A \) is always less than 0, which indicates that results of 12 problem sets computed by GA\(_{maxRS}\) are all better than those provided by DMAS/RIA. In addition, when resource transfer times are taken into account, \( PRE_{GA} \) is smaller than \( PRE_D \) for 8 of the 12 problem sets. Moreover, \( PRE_{GA} \) has a minimum of 1% and a maximum of 24%, which is more stable than \( PRE_D \). These further indicate that GA\(_{maxRS}\) has obvious advantages in solving the DRCMPSPTT.

| Problem subsets | GA\(_{maxRS}\) with TT | GA\(_{maxRS}\) without TT | DMAS/RIA with TT | DMAS/RIA without TT | \( PRE_A \) (%) |
|-----------------|-------------------|-----------------|-----------------|-----------------|---------------|
| MP30_2          | 13                | 10.9            | 23.3            | 18              | 29            |
| MP90_2          | 6.9               | 6.7             | 17.1            | 11.8            | 45            |
| MP120_2         | 56.1              | 55.6            | 89.6            | 80.3            | 12            |
| MP30_5          | 22.2              | 18.08           | 35.28           | 21.64           | 63            |
| MP90_5          | 11.68             | 9.36            | 22              | 13.32           | 65            |
| MP120_5         | 66.84             | 65.48           | 86.12           | 77.68           | 11            |
| MP30_10         | 81.58             | 78.16           | 92.68           | 83              | 12            |
| MP90_10         | 59.34             | 50.36           | 62.66           | 61.52           | 2             |
| MP120_10        | 164.28            | 140.3           | 162.6           | 144.1           | 13            |
| MP30_20         | 199.63            | 189.6           | 206.9           | 193             | 7             |
| MP90_20         | 43.38             | 34.92           | 42.88           | 43.13           | -1            |
| MP120_20        | 46.42             | 39.71           | 48.09           | 42.11           | 14            |
| Average         | 64.28             | 58.26           | 74.1            | 65.8            | 13            |

Furthermore, we evaluate the effectiveness of GA\(_{maxRS}\) in solving the DRCMP-SPTT with different scales and different resource constraint tensions. Fig. 11 (c) shows the \( PRE_{GA} \) of the 12 problem sets. When the number of real activities in the single project is fixed, \( PRE_{GA} \) does not show an obvious upward trend with the increase of the project and the problem scale, indicating that GA\(_{maxRS}\) can solve the DRCMPSPTT of different scales well. Furthermore, Fig 11 (d) shows the
variation trend of the $PRE_{GA}$ of the 12 problem sets as $UF_{AV}$ increases. It can be seen that with the gradual increase of $UF_{AV}$, $PRE_{GA}$ shows a downward trend and becomes increasingly more stable. These results indicate that $GA_{maxRS}$ is more effective at solving the DRCMPSPTT with more fierce resource competition.

Figure 11. (a) mean value of the APD obtained by different methods, and (b) The three types PREs, (c) The $PRE_{GA}$ of the 12 problem sets, and (d) The change trend of $PRE_{GA}$ as $UF_{AV}$ increases.

6. Conclusions. In this paper, the decentralized resource-constrained multi-project scheduling problem with transfer times (DRCMPSPTT) was studied. The integer linear programming of the DRCMPSPTT was established, and an adaptive genetic algorithm was designed to solve it. We combined the modified traditional serial and parallel SGSs with four resource transfer rules into eight decoding schemes, and then embedded these combination schemes into the GA, respectively, to compare their effects. The computational results show that the combination of the parallel SGS and the maxRS transfer rule can make the GA more effective than other combinations. In addition, based on the best scheme, we compared the performance of $GA_{maxRS}$ and CPLEX in solving small-sized problems. The results show that $GA_{maxRS}$ has good quality and efficiency in solving the small-sized DRCMPSPTT. Furthermore, we compared the performance of $GA_{maxRS}$ and DMAS/RIA in the relevant literature for the large-sized DRCMPSPTT. The experimental results show that $GA_{maxRS}$ is far superior to DMAS/RIA, and it has strong adaptability to problems with different scales and different degrees of resource constraint.

The project scheduling problem is an NP-hard problem in the combinational optimization field, and there are many similar problems in related management
fields, such as the garbage truck path optimization problem in waste management, the batch scheduling problem in supply chains, the production scheduling problem and so on. Fast and efficient heuristic algorithms are indispensable to solve such problems. Therefore, the development of more efficient optimal methods and more efficient heuristics to solve the large-sized problems is the focus of future research. In addition, this research on the DRCMPSP-TT is based on a deterministic environment, and the results are only applicable to deterministic project management. However, with the development of information technology, uncertain environments are often encountered in the project implementation process, such as random duration and dynamic arrival of the project. Therefore, considering the robustness of project duration under various random factors is also an important research direction.

Acknowledgments. This work was supported by National Natural Science Foundation of China under Grants Nos. 71971173, 71671117 and 71572148, Fundamental Research Funds for the Central Universities under Grant No. 3102019J002, Natural Science Basic Research Plan in Shaanxi Province of China under Grant No. 2020JM-146, and The Seed Foundation of Innovation and Creation for Graduate Students in Northwestern Polytechnical University Nos. ZZ2019038 and CX2020026.

REFERENCES

[1] S. Adhau, M. L. Mittal and A. Mittal, A multi-agent system for decentralized multi project scheduling with resource transfers, International Journal of Production Economics, 146 (2013), 646–661.

[2] S. Adhau, M. L. Mittal and A. Mittal, A multi-agent system for distributed multi-project scheduling: An auction-based negotiation approach, Engineering Applications of Artificial Intelligence, 25 (2012), 1738–1751.

[3] B. Afshar-Nadjafi and M. Majlesi, Resource constrained project scheduling problem with setup times after preemptive processes, Computers and Chemical Engineering, 69 (2014), 16–25.

[4] C. Artigues, P. Michelon and S. Reussner, Insertion techniques for static and dynamic resource-constrained project scheduling, European Journal of Operational Research, 149 (2003), 249–267.

[5] D. Bedworth and J. Bailey, Integrated Production Control Systems Management, Analysis, Design, 2nd edition, John Wiley & Sons, Inc., New York, 1999.

[6] C. Bierwirth, D. Mattfeld and H. Kopfer, On permutation representations for scheduling problems, in Parallel Problem Solving from Nature, Lecture Notes in Computer Science, 1996, 310–318.

[7] P. Brucker, S. Knust, A. Schoo and O. Thiele, A branch and bound algorithm for the resource-constrained project scheduling problem, European Journal of Operational Research, 107 (1998), 272–288.

[8] Z. Cai and X. Li, A hybrid genetic algorithm for resource-constrained multi-project scheduling problem with resource transfer time, IEEE International Conference on Automation Science and Engineering, Seoul, 2012, 569–574.

[9] G. Confessore, S. Giordani and S. Rismondo, A market-based multi-agent system model for decentralized multi-project scheduling, Annals of Operations Research, 150 (2007), 115–135.

[10] E. W. Davis, Project network summary measures constrained resource scheduling, AIIE Transactions, 7 (1975), 132–142.

[11] E. L. Demeulemeester and W. S. Herroelen, An efficient optimal solution procedure for the preemptive resource-constrained project scheduling problem, European Journal of Operational Research, 90 (1996), 334–348.

[12] L. Djerid, M. C. Portman and P. Villon, Performance analysis of permutation cross-over genetic operators, Journal of Decision Systems, 5 (1996), 131–140.
[13] S. Hartmann and R. Kolisch, Experimental evaluation of state-of-the-art heuristics for resource-constrained project scheduling problem, European Journal of Operational Research, 127 (2000), 394–407.
[14] J. B. Holland and J. H. Holland, Adaption in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, Mich., 1975, 126–137.
[15] J. Homberger, A multi-agent system for the decentralized resource constrained multi-project scheduling problem, International Transactions in Operational Research, 14 (2007), 565–589.
[16] J. Homberger, A (μ, λ)-coordination mechanism for agent-based multi-project scheduling, OR Spectrum, 34 (2012), 107–132.
[17] R. Jans and Z. Degraeve, Modeling industrial lot sizing problems: A review, International Journal of Production Research, 46 (2008), 1619–1643.
[18] R. L. Kadri and F. F. Boctor, An efficient genetic algorithm to solve the resource-constrained project scheduling problem with transfer times: The single mode case, European Journal of Operational Research, 265 (2018), 454–602.
[19] J. E. Kelley, The critical-path method: Resources planning and scheduling, Industrial Scheduling, 13 (1963), 347–365.
[20] S. Khalilpourazari, B. Naderi and S. Khalilpourazary, Multi-objective stochastic fractal search: A powerful algorithm for solving complex multi-objective optimization problems, Soft Computing, 24 (2020), 3037–3066.
[21] S. Khalilpourazari and S. H. R. Pasandideh, Modeling and optimization of multi-item multi-constrained EOQ model for growing items, Knowledge-Based Systems, 164 (2019), 150–162.
[22] S. Khalilpourazari and S. H. R. Pasandideh, Sine-cosine crow search algorithm: Theory and applications, Neural Computing and Applications, 32 (2019), 7725–7742.
[23] R. Kolisch, Project scheduling with setup times, in Project Scheduling Under Resource Constraints, Production and Logistics, Physica, Heidelberg, 1995, 177–185.
[24] R. Kolisch and S. Hartmann, Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis, in Project Scheduling, International Series in Operations Research & Management Science, Springer, Boston, 1999, 147–178.
[25] R. Kolisch and A. Sprecher, PSPLIB–A project scheduling library, European Journal of Operational Research, 96 (1996), 205–216.
[26] D. Kruger and A. Scholl, A heuristic solution framework for the resource constrained (multi-)project scheduling problem with sequence-dependent transfer times, European Journal of Operational Research, 197 (2009), 492–508.
[27] D. Kruger and A. Scholl, Managing and modelling general resource transfers in (multi-)project scheduling, OR Spectrum, 32 (2010), 369–394.
[28] C. J. Liao, C. W. Chao and L. C. Chen, An improved heuristic for parallel machine weighted flowtime scheduling with family set-up times, Computers and Mathematics with Applications, 63 (2012), 110–117.
[29] A. Lova and P. Tormos, Analysis of scheduling schemes and heuristic rules performance in resource-constrained multiproject scheduling, Annals of Operations Research, 102 (2001), 263–286.
[30] A. Lova, P. Tormos, M. Cervantes and F. Barber, An efficient hybrid genetic algorithm for scheduling projects with resource constraints and multiple execution modes, International Journal of Production Economics, 117 (2009), 302–316.
[31] D. Merkle, M. Middendorf and H. Schmeck, Ant colony optimization for resource-constrained project scheduling, IEEE Transactions on Evolutionary Computation, 6 (2002), 333–346.
[32] M. Mika, G. Waligra and J. Weglarz, Tabu search for multi-mode resource-constrained project scheduling with schedule-dependent setup times, European Journal of Operational Research, 187 (2008), 1238–1250.
[33] M. L. Mittal and A. Kanda, Scheduling of multiple projects with resource transfers, International Journal of Mathematics in Operational Research, 1 (2009), 303–325.
[34] K. Mounemene and J. A. Ferland, Activity list representation for a generalization of the resource-constrained project scheduling problem, European Journal of Operational Research, 199 (2009), 46–54.
[35] M. S. Nagano, A. A. Silva and L. A. N. Lorena, A new evolutionary clustering search for a no-wait flow shop problem with set-up times, Engineering Applications of Artificial Intelligence, 25 (2012), 1114–1120.
[36] K. Neumann, C. Schwindt and J. Zimmermann, Project Scheduling with Time Windows and Scarce Resources, Springer-Verlag, Berlin, 2002.
[37] S. H. R. Pasandideh and S. Khalilpourazari, Sine cosine crow search algorithm: A powerful hybrid meta heuristic for global optimization, preprint, arXiv:1801.08485.
[38] J. Poppenborg and S. Knust, A flow-based tabu search algorithm for the RCPSP with transfer times, OR Spectrum, 38 (2016), 305–334.
[39] V. Roshanaei, B. Naderi, F. Jolai and M. Khalili, A variable neighborhood search for job shop scheduling with set-up times to minimize makespan, Future Generation Computer Systems, 25 (2009), 654–661.
[40] M. Rostami, M. Bagherpour, M. M. Mazdeh and A. Makui, Resource Pool Location for Periodic Services in Decentralized Multi-Project Scheduling Problems, Journal of Computing in Civil Engineering, 31 (2017), 04017022.
[41] V. Valls, F. Ballestin and M. S. Quintanilla, A hybrid genetic algorithm for the RCPSP, Technical report, Department of Statistics and Operations Research, University of Valencia, 2003.
[42] V. Valls, F. Ballestin and M. S. Quintanilla, Justification and RCPSP: A technique that pays, European Journal of Operational Research, 165 (2005), 375–386.
[43] M. Vanhoucke, Setup times and fast tracking in resource-constrained project scheduling, Computers & Industrial Engineering, 54 (2008), 1062–1070.
[44] K. K. Yang and C. C. Sum, A comparison of resource allocation and activity scheduling rules in a dynamic multi-project environment, Journal of Operation Management, 11 (1993), 207–218.
[45] K. K. Yang and C. C. Sum, An evaluation of due date, resource allocation, project release, and activity scheduling rules in a multi project environment, European Journal of Operational Research, 103 (1997), 139–154.

Received January 2020; 1st revision July 2020; 2nd revision July 2020.

E-mail address: zhangjingwen@nwpu.edu.cn
E-mail address: wanjunl@mail.nwpu.edu.cn
E-mail address: 991wl@163.com