A Genetic Algorithm for the Proactive Resource-Constrained Project Scheduling Problem with Activity Splitting

Abstract—**Proactive scheduling aims at the generation of robust baseline schedules, which has been studied for many years with the assumption that activity splitting is not allowed. In this paper, we focus on the proactive resource-constrained project scheduling problem in which each activity can be split at discrete time instants under the constraints of a maximum number of splitting and a minimum period of continuous execution. Besides, in this problem setup times are considered. Two properties of the established model and one lemma are proposed and applied in the developed genetic algorithm. After linearizing the proposed model, we use a commercial mathematical programming solver as a benchmark to solve the problem. From the computational results, we find that the developed genetic algorithm is effective and efficient in solving the defined problem, and activity splitting improves robustness. With the growth of the maximum number of splitting, the decline in the minimum execution time, the decrease in the setup times, and the extension of the project due date, robustness increases.**

Index Terms—**Activity splitting, setup time, genetic algorithm, proactive project scheduling, solution robustness**

Managerial Relevance Statement—**In consideration of the fact that some project activities can be split, our research proposes a model to split and schedule activities with the aim to generate robust baseline schedules that are protected against schedule disruptions. With our developed genetic algorithm, the contractor could generate satisfying baseline schedules within an acceptable computing time, which are likely to have low adjustment costs during project execution. From the computational results, we find that activity splitting improves robustness, which offers a method to improve schedule robustness when activity splitting is allowed. Furthermore, we find that, with the growth of the maximum number of splitting, the decline in the minimum execution time, the decrease in the setup times, and the extension of the project due date, schedule robustness increases. To summarize, our findings could help project managers better understand the benefits of making good use of activity splitting in the scheduling of activities in a resource-constrained project, and then help them do a better job in improving the robustness of baseline schedules.**

I. Introduction

It is a well-known fact that project activities are subject to considerable uncertainties, such as accidents, resource breakdowns, and bad weather conditions, which may lead to numerous schedule disruptions during project execution and therefore incur some costs when project managers adjust the starting times of the activities to deal with them. Accordingly, proactive scheduling has been the subject of many research efforts that aim to generate robust baseline schedules that are protected against schedule disruptions. The more robust the baseline schedules are, the lower the adjustment costs will be during project execution. These research efforts have led to many models and algorithms, which are summarized in [1] - [4].

Two robustness approaches are considered in this field, i.e., quality robustness and solution robustness [5]. For quality robustness, the robust multi-mode discrete time/cost trade-off problem is introduced and solved by exact and heuristic algorithms [6], [7]. Regarding solution robustness, various approaches are developed to cope with multiple disruptions, including activity duration disruptions [8], stochastic activity durations [9], [10], and stochastic resource availabilities [11], [12]. In contrast to the literature that addresses quality robustness or solution robustness separately, several studies have concentrated on the potential trade-off between these two types of robustness. Al-Fawzan and Haouari develop a bi-objective model with an aggregation function in the absence of available information regarding the nature or size of the uncertain events [13]. With the composite objective of maximizing both schedule stability and timely project completion probability, Van de Vonder *et al*. develop a heuristic algorithm for minimizing a stability cost function [14] and they discuss the results obtained by a large experimental design that is established to evaluate several predictive-reactive resource-constrained project scheduling procedures [15]. Furthermore, Chtourou and Haouari present a two-stage algorithm in which the first stage is designed to minimize the project makespan, while the second one aims to maximize schedule robustness [16]. Deblaere *et al*. propose an objective to minimize a cost function that consists of the weighted expected activity starting time deviations and the penalties or bonuses that are associated with late or early project completion [17]. One recent study defines a new robustness measure that is completely independent of the applied reactive policy and then introduces a branch-and-cut algorithm to solve a sample average approximation of the original problem [18]. More details about the literature on proactive scheduling can be found in Appendix A.

It is noteworthy that most of the literature on proactive scheduling does not consider activity splitting, which means that these activities are not divisible. However, if activity splitting is allowed, it may be more flexible for project managers to schedule activities and make good use of resources and slacks to generate much more robust baseline schedules. In other words, if activity splitting contributes to higher robustness, then activities will be split into certain parts for execution; otherwise, activities will be scheduled without splitting. Note that we consider to split activities actively during the stage of baseline schedule generation, which is different from interrupting activities passively to deal with disruptions during the stage of project execution.

In previous literature, some researchers have already considered the project scheduling problem with activity splitting. The difference from what we discussed before is that their objective function is mainly focused on makespan minimization and the term they use of activity splitting is activity preemption or activity interruption. For example, Demeulemeester and Herroelen describe a branch-and-bound procedure to solve the preemptive resource-constrained project scheduling problem (PRCPSP) with the objective of minimizing project duration [19]. Following the work that reveals the potential benefits of allowing one interruption in the scheduling of activities in a resource-constrained project [20], Buddhakulsomsiri and Kim present a priority rule-based heuristic for the multi-mode scheduling problem with the splitting of activities around unavailable resources allowed [21]. Based on an analysis of the characterizations of the solution set for the preemptive and non-preemptive resource-constrained project scheduling problem (RCPSP), Damay *et al*. present a linear programming based algorithm to solve the two problems [22]. Ballestín *et al*. mainly focus on problem 1_PRCPSP in which a maximum of one interruption per activity is allowed, and they propose a new model that covers most practical applications of discrete activity preemption [23], [24]. A genetic algorithm for the non-preemptive multi-mode scheduling problem is developed and extended to the preemptive case of this problem [25]. Recently, Haouari *et al*. use a linear programming model that is based on the PRCPSP to compute a lower bound for the RCPSP [26], and Moukrim *et al*. propose an effective branch-and-price algorithm based on minimal interval order enumeration that involves column generation as well as constraint propagation [27]. For more research efforts on project scheduling problems with activity splitting, we refer to [28] - [30]. In Appendix B, more details can be found about the literature on the resource-constrained project scheduling problem with activity splitting.

In practice, activities may be different with respect to activity splitting. Firstly, activities may need to be executed continuously for certain periods before the next splitting and the duration of the continuous execution time is different. Secondly, some activities, such as chemical reactions, may not be split at all due to technical reasons, but some activities, such as the transportation of materials, are technically feasible to be split into certain parts. Even though the activities are all feasible for splitting, the maximum number of splitting allowed may be still different. Thirdly, some activities, such as managerial operations, do not need setup times before execution while some activities, such as a bridge construction, may need certain periods for preparation. From the previous literature, we know that the first two differences have been considered and measured by two factors, i.e., the maximum number of activity splitting and the minimum continuous execution [24], [30], but the third one has not been considered in the scheduling research with activity splitting.

Typically speaking, activities that are split into certain parts cause additional setup times (and thus additional costs) when returning to their execution. In other words, if one activity technically needs a setup time before execution, then there will be additional setup times for the second and the subsequent parts of this activity. This implies that there will be a trade-off between the benefits of activity splitting and the drawbacks of the increasing setup times under the objective of solution robustness maximization.

Based on the facts above, this paper presents a proactive resource-constrained project scheduling problem with activity splitting. In this problem, each activity can be split at discrete time instants under the constraints of a maximum number of splitting and a minimum period of continuous execution. Besides, additional setup times are considered when the activities return to execution from splitting. Different from the existing proactive scheduling which aims to improve schedule robustness without activity splitting, this paper aims to take activity splitting into account to seek opportunities to further improve schedule robustness. Therefore, it can be regarded as a two-stage problem: the first one is to decide how to split activities and the second stage is proactive scheduling, i.e. how to schedule activities to construct an optimal baseline schedule with the objective of solution robustness maximization. The solution robustness is obtained by inserting time buffers into the baseline schedule with the consideration of precedence, renewable resources, and project deadline constraints, and it is measured by a free slack based function, an adjusted surrogate solution robustness measure that is proposed by Lambrechts *et al*. [31]. This problem can be defined as an extension of the proactive RCPSP because activity splitting becomes allowed. As activities are handled in different ways in terms of activity splitting, this problem is also a generalization of m _PRCPSP where all activities can be split m times. We believe that the proposed problem, which to the best of our knowledge has not thus far been investigated, may be more practical because it takes activity splitting into account and considers multiple cases of divisible activities.

Note that in previous literature on proactive scheduling activities are indivisible and treated as the basic project units. However, based on the theory of the work breakdown structure, activities are broken down by different levels. Therefore, in this paper activities are much more similar to work packages, which are not divided to the lowest level so that project managers can have the freedom to decide whether to further split the activities. Conversely, if activities have already been divided to the lowest level, we can regard them as subactivities, and then we can decide how to merge and schedule them to decrease the setup times and improve schedule robustness, which is just equivalent to scheduling activities without activity splitting in this paper.

The rest of this paper is organized as follows. In Section II we present the notations and the problem formulation. Section III is devoted to the development of a genetic algorithm that is based on the analysis of the proposed scheduling model. Section IV conducts an extensive computational experiment. Finally, in Section V, general conclusions and directions for further research are presented.

II. Problem Formulation

A. *Optimization Model*

Consider a project represented in an activity-on-the-node

(AoN) format by means of a digraph $G = (N, A)$, where the set of nodes N represents the activities and the set of arcs A the finish-start, zero-lag precedence relations. The activities are numbered from the dummy start activity 1 to the dummy end activity n, and each activity i has a duration d_i and requires renewable resources to ensure that it is carried out. There are K different renewable resource types with an availability in each period $[t, t+1)$, $(t = 0,1,\cdots,D)$, of R_k^ρ units, $k = 1,2,\cdots,K$. Each activity i requires $r_{i,k}^\rho$ units of resource type k during each period in which it is processed. Dummy activities have zero duration and resource usage. We use subactivity (i, v) to denote the v-th part of activity i, which has the same resource usage as activity i. The only difference between the activity and its subactivities is the duration. The project deadline is denoted as D.

For practical reasons that activities are different with respect to activity splitting, we make the following three assumptions. Firstly, for each activity i, a required minimum execution time ε_i is predefined during which the activity must be in progress without any splitting. This forces the duration $dur_{i,v}$ of subactivity (i, v) to be at least ε_i. Secondly, each activity i can be split a maximum of $\eta_i (\eta_i < \left[\frac{d_i}{\varepsilon_i}\right])$ times at any discrete time instant, which results in $V_i\ (V_i \le \eta_i + 1)$ precedence-connected subactivities, each of which has a resource requirement $r_{i,k}^\rho$. The first two assumptions are responses to the fact that activities cannot be split too frequently. Obviously, the case $\eta_i = 0$ or $d_i < 2\varepsilon_i$ means that activity i must be processed without splitting. In addition, as a response to the fact that in projects activities may need setup times for preparation, we assume each activity technically needs setup time θ_i before execution. Note that the setup time is not included in the activity duration, which means the actual duration of one indivisible activity is its duration plus its setup time, and there will be additional setup times for activities that are split into certain parts. Obviously, the case $\theta_i = 0$ means that activity i technically does not need setup time.

The weight w_i, which is allocated to each activity i, denotes the marginal cost of deviating the completion time of activity i during project execution from its planned completion time in the baseline schedule. The cost can be regarded as the impact of such a delay on all its immediate and transitive successors. Because the successors of the subactivities are the same as those of their original activity, we assume that the weights of the subactivities are equivalent to those of their original activities. The free slack $FS_{i,v}$, which represents the time buffers after the duration of subactivity (i, v), is defined as the total amount of time this subactivity can be delayed without causing any precedence or resource constraint violations. Note that the free slack here is defined in the context of limited resources, which is an extension of the one in the framework of CPM (Critical Path Method). Referring to Lambrechts *et al.* [31], the utility of the free slacks may decrease marginally in exponent with the increase of their amounts. For example, if one activity has a free slack of 6, then the first slack will be much more beneficial than the sixth one to absorb the disruptions because it is less likely for the activity to delay six periods. Thus, the robustness that is generated by $FS_{i,v}$ can be calculated as $w_i \sum_{b=1}^{FS_{i,v}} e^{-b}$. Then, counting the utilities of all subactivities of all activities, the robustness of a schedule (hereafter denoted as $Robu$) can be defined as $\sum_{i=1}^{n}[w_i \left(\sum_{v=1}^{V_i} \sum_{b=1}^{FS_{i,v}} e^{-b}\right)]$.

There are three groups of decision variables in this problem, i.e., V_i, $dur_{i,v}$, and $s_{i,v}$, which respectively represent the number of subactivities of activity i, the duration of subactivity (i, v), and the starting time of this subactivity. Then, the goal is to decide the optimal values for V_i, $dur_{i,v}$, and $s_{i,v}$ to obtain a baseline schedule with the maximum schedule robustness $Robu$. The optimization model for the proactive resource-constrained project scheduling problem with activity splitting is constructed as follows. It is important to note that in our model setup times are not included in d_i but are included in $dur_{i,v}$.

$$\text{Maximize } Robu = \sum_{i=1}^{n}[w_i \left(\sum_{v=1}^{V_i} \sum_{b=1}^{FS_{i,v}} e^{-b}\right)] \tag{1}$$

Subject to:

$$s_{1,1} = 0 \tag{2}$$

$$s_{i,V_i} + dur_{i,V_i} \le s_{j,1} \quad (i,j) \in A \tag{3}$$

$$s_{i,v} + dur_{i,v} \le s_{i,v+1} \quad i = 1,\cdots,n; \ \ v = 1,\dots,V_i - 1 \tag{4}$$

$$s_{n,1} \le D \tag{5}$$

$$\sum_{i\in S(t)} r_{i,k}^\rho \le R_k^\rho \quad k = 1,2,\cdots,K; \ \ t = 0,1,\cdots,D \tag{6}$$

$$\sum_{v=1}^{V_i} dur_{i,v} = d_i + V_i \times \theta_i \quad i = 1,2,\cdots,n \tag{7}$$

$$V_i \le \eta_i + 1 \quad i = 1,2,\cdots,n \tag{8}$$

$$dur_{i,v} - \theta_i \ge \varepsilon_i \quad i = 1,2,\cdots,n; \ \ v = 1,2,\dots,V_i \tag{9}$$

$$V_i,\ dur_{i,v}, \text{and } s_{i,v} \text{ are nonnegative integers} \quad \forall i, \forall v \tag{10}$$

In the formulation, the objective function (1) is to maximize solution robustness. Equation (2) forces the project to start at time 0. The precedence constraints given by (3) indicate that the start of activity j must wait for the end of the last subactivity of all its preceding activities, and in constraints (4) one subactivity of an activity does not start before the end of the previous subactivity of the same activity. Constraint (5) imposes a deadline on the project. As $S(t)$ is the set of activities that are in progress during time interval $[t, t+1)$, constraints (6) force the total units of utilized resources to be no greater than the available resource capacity for every period. The conditions for activity splitting are reflected in (7), (8), and (9). Equation (7) ensures that the duration of all the subactivities of activity i must equal the sum of the processing time of activity i and its total setup times. The constraints (8) guarantee that the times of splitting for a given divisible activity is no more than a predefined level called η_i, while in (9) for each subactivity the duration without setup time must be at least its minimum execution time. The range of values for V_i, $dur_{i,v}$, and $s_{i,v}$ are given in the constraints (10).

In this non-linear model, we need to take constraints (7), (8), and (9) into account to decide how to split activities and decide how to schedule those subactivities based on the constraints (2) - (6). In the first decision, there will be a trade-off between the benefits of activity splitting and the drawbacks of the increasing setup times. In the second decision, there will be a trade-off between inserting time buffers and the deadline constraint. For the objective function, $FS_{i,v}$ will be calculated by an algorithm that is developed in the next section. Note that $FS_{i,v}$ may not equal the values of time buffers. Time buffers are inserted based on the rule of marginally decreasing slack utility, activity weights, and the changes of the schedule after inserting time buffers, which together influence the improvement of the objective function value. The bigger the improvement, the bigger the possibility to insert time buffers to this activity.

B. An Example

We use an example to illustrate the problem that is identified above. The AoN network of the example is depicted in Fig. 1 where activities 1 and 6 are the dummy start and end activities respectively. The activities in the project require one renewable resource and their durations as well as resource requirements, activity weights, the maximum numbers of splitting, the minimum periods of continuous execution, and the setup times are labeled with the nodes. Other data of the project are as follows: $K = 1,\ R_1^\rho = 4,\ D = 14$. To demonstrate that activity splitting is beneficial to schedule robustness, we give the most robust baseline schedules without and with activity splitting which are depicted as schedules (a) and (b) respectively in Fig. 2 and compare the results produced below.

1) The Case without Activity Splitting

In this case, we suppose that activities are indivisible during execution. Therefore, we have $\eta_i = 0$ for each activity i. Under this circumstance, schedule (a) is the optimal baseline schedule in terms of solution robustness where each activity has only one subactivity and the part with slashes represents the setup time of activity 5. Obviously, only activity 2 has a free slack of 2. The corresponding objective function value is equal to 2.00 and was calculated as shown in Table I.

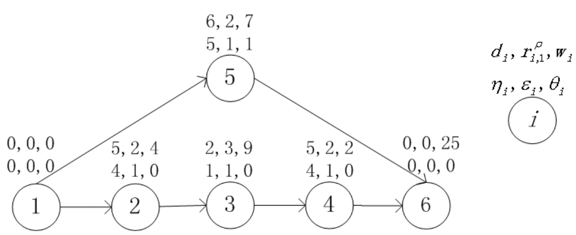


Fig. 1. An example.

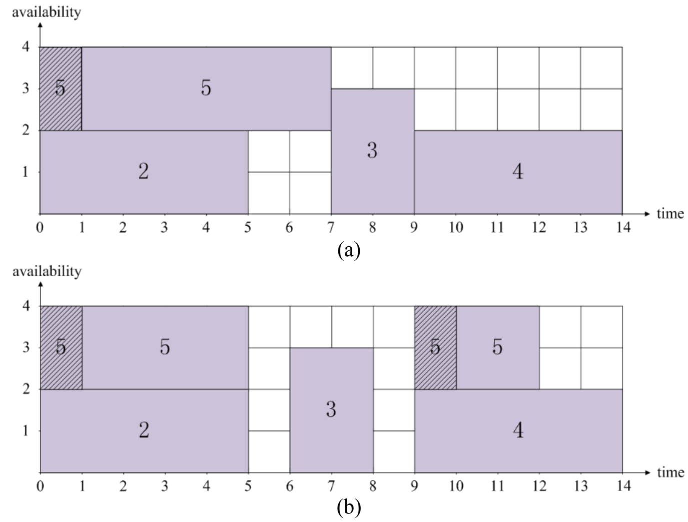


Fig. 2. Two feasible schedules for the project.

2) The Case with Activity Splitting

In this case, it is assumed that activity splitting is allowed. Based on the data shown in Fig. 1, schedule (b) is the most robust baseline schedule where activity 5 is split into two subactivities. Because of activity splitting, another setup time is needed before the execution of the second subactivity of activity 5. The corresponding objective function value is 10.90, the computation of which can be found in Table I as well.

Comparing the results discussed above, we can find that an improvement of 445% is obtained for the free slack based objective function value, which verifies the potential benefits of making good use of activity splitting in proactive scheduling to some extent. The reason is that activity splitting enhances the flexibility of scheduling activities, which is beneficial to making good use of resources to shorten the project duration and thus spare more space to insert time buffers. Next, we will make an analysis about the cost saving when taking activity splitting into account. In this example, compared with schedule (a), schedule (b) is likely to have lower adjustment costs. For example, if the activity duration increases by 1 both for activity 3 and activity 5, we need to adjust the starting times of activity 3 and 4 in schedule (a), but do nothing in schedule (b), which incurs a lower cost of 13 (9+2*2) for schedule (b).

TABLE I
CALCULATION OF THE OBJECTIVE FUNCTION

i	w_i	Schedule (a)			Schedule (b)		
		$FS_{i,1}$	$\sum_{b=1}^{FS_{i,1}} e^{-b}$	$w_i \sum_{b=1}^{FS_{i,1}} e^{-b}$	$FS_{i,v}$	$\sum_{v=1}^{V_i} \sum_{b=1}^{FS_{i,v}} e^{-b}$	$w_i \sum_{v=1}^{V_i} \sum_{b=1}^{FS_{i,v}} e^{-b}$
1	0	0	0.00	0.00	0	0.00	0.00
2	4	2	0.50	2.00	1	0.37	1.48
3	9	0	0.00	0.00	1	0.37	3.33
4	2	0	0.00	0.00	0	0.00	0.00
5	7	0	0.00	0.00	1+2	0.87	6.09
6	25	0	0.00	0.00	0	0.00	0.00
$Robu$				2.00			10.90

III. THE DEVELOPED GENETIC ALGORITHM

In the first part, we propose two properties of the scheduling model and one lemma, which can be used for the development of the algorithm. In the second part, we firstly explain why we choose a genetic algorithm to solve the problem and then present the framework of the developed algorithm. Afterwards, technical details are given to describe the developed genetic algorithm in seven parts where the proposed properties and the lemma are used for the local search procedure.

A. The Properties and the Lemma

To explain the properties and the lemma more clearly, we provide three definitions in advance.

Definition 1: In a given schedule, time period T is feasible for a minimum part of activity i, whose duration equals the minimum execution time of activity i, to be executed if: 1) the successors of activity i do not start before the end of time period T, 2) the remaining resources in time period T can still satisfy the resource requirements of activity i, 3) the length of time period T is no less than $\varepsilon_i + \theta_i$, and 4) the starting time of the time period is after the completion time of the last subactivity of activity i.

Definition 2: Subactivity (i, v) is divisible if $V_i \leq \eta_i$ and $dur_{i,v} - \theta_i \geq 2\varepsilon_i$.

Definition 3: Subactivity (i, v) is abundant in free slacks if $FS_{i,v} \geq 2 + \theta_i$.

Based on the above definitions, we develop two properties of the model, which are named as Pioneering and Balancing respectively based on the mechanism of each operation. After that, one lemma is proposed for improving the schedule robustness.

Property 1 (Pioneering): If subactivity (i, v) is divisible, and there is a feasible period T, whose length is denoted as ξ_T, for activity i to be executed, then schedule robustness can be improved in three steps: Firstly, keep other activities unchanged. Secondly, divide this subactivity into two parts, which are denoted as (i, v_1) and (i, v_2), whose durations are $dur_{i,v} - dd$ $(\varepsilon_i \leq dd \leq \min\{dur_{i,v} - \varepsilon_i - \theta_i,\ \xi_T - \theta_i\})$ and $dd + \theta_i$ respectively. Thirdly, schedule the two parts of this subactivity in the original and the new periods. In this way, the objective function value of the schedule can be improved.

Proof of Property 1: After the Pioneering operation, the free slack of subactivity (i, v_1) will be $FS_{i,v} + dd$, while the free slack of subactivity (i, v_2) will be $\xi_T - dd - \theta_i$. The utility of the free slacks before the operation is $U_1 = \sum_{b=1}^{FS_{i,v}} e^{-b}$, which is smaller than that after the operation $U_2 = \sum_{b=1}^{FS_{i,v}+dd} e^{-b} + \sum_{b=1}^{\xi_T - dd - \theta_i} e^{-b}$. Hence, Property 1 can be used as a rule to maximize schedule robustness.

Property 2 (Balancing): If subactivity (i, v) is divisible, and its free slack is not abundant, while the reverse is true for subactivity (i, p), then schedule robustness may be improved by transferring one unit of time from the duration of subactivity (i, v) to that of subactivity (i, p).

Proof of Property 2: From the prerequisites of Property 2, we can obtain the four following constraints: $dur_{i,v} - \theta_i \geq 2\varepsilon_i$, $0 \leq FS_{i,v} \leq 1$, $\varepsilon_i \leq dur_{i,p} - \theta_i < 2\varepsilon_i$, and $FS_{i,p} \geq 2$. After the Balancing operation, the free slack of subactivity (i, v) will be $FS_{i,v} + 1$, while the free slack of subactivity (i, p) will be $FS_{i,p} - 1$. Then, the utility of the free slacks after the operation can be calculated as: $U_2 = \sum_{b=1}^{FS_{i,v}+1} e^{-b} + \sum_{b=1}^{FS_{i,p}-1} e^{-b} = \sum_{b=1}^{FS_{i,v}} e^{-b} + \sum_{b=1}^{FS_{i,p}} e^{-b} + \left(e^{-FS_{i,v}-1} - e^{-FS_{i,p}}\right)$. Because of the two following constraints, i.e., $-FS_{i,v} \geq -1$ and $-FS_{i,p} \leq -2$, U_2 will be no less than the utility before the operation $U_1 = \sum_{b=1}^{FS_{i,v}} e^{-b} + \sum_{b=1}^{FS_{i,p}} e^{-b}$. Hence, Property 2 can be used to improve the objective function value.

To summarize, Pioneering facilitates the discovery of new periods for activities to be executed, and Balancing is used to balance the length of the durations between two subactivities of one activity. As the two properties can help to transform subactivities into divisible ones with abundant free slacks, they pave the way for the following lemma, which is used to divide one subactivity into subactivities that specifically share the buffer of the original subactivity as equally as possible such that the schedule robustness can be improved.

Lemma 1: For any subactivity (i, v) that is divisible and abundant in free slacks, we can first divide this subactivity into $num_{i,v} = \min\left(\left\lfloor\frac{dur_{i,v}-\theta_i}{\varepsilon_i}\right\rfloor, \frac{FS_{i,v}+\theta_i}{1+\theta_i},\ \eta_i - V_i + 2\right)$ parts whose durations are no less than ε_i, and then schedule them continuously and make sure their free slacks are as equal as possible, i.e., the difference between the maximum and the minimum free slack value of the newly generated subactivities is no more than one. In this way, schedule robustness will be improved.

Proof of Lemma 1: As $\sum_{b=1}^{\infty} e^{-x} = \frac{1}{e-1} < 2e^{-1}$ is true, it would be always beneficial for improving schedule robustness by splitting a divisible subactivity with abundant free slacks into certain parts. To maximize robustness, $FS_{i,v} - \left(num_{i,v} - 1\right)\theta_i$ should be no less than $num_{i,v}$, so $num_{i,v} \leq \frac{FS_{i,v}+\theta_i}{1+\theta_i}$. Furthermore, if the difference between the maximum and the minimum free slack value of the newly generated subactivities is more than one in the optimal improvement, for example, $fs_1 > fs_2 + 1$, then $\sum_{b=1}^{fs_1} e^{-b} + \sum_{b=1}^{fs_2} e^{-b} = \left(\sum_{b=1}^{fs_1-1} e^{-b} + e^{-fs_1}\right) + \sum_{b=1}^{fs_2} e^{-b} < \sum_{b=1}^{fs_1-1} e^{-b} + \left(\sum_{b=1}^{fs_2} e^{-b} + e^{-fs_2-1}\right) = \sum_{b=1}^{fs_1-1} e^{-b} + \sum_{b=1}^{fs_2+1} e^{-b}$. As there will be a contradiction, to obtain an optimal improvement based on the proposed lemma, the original subactivity should be divided into $num_{i,v}$ parts whose free slacks are as equal as possible.

Note that it is possible that none of the subactivities in a schedule is divisible and abundant in free slacks, and under this circumstance we cannot apply Lemma 1 to improve solution robustness of this schedule.

B. The Developed Genetic Algorithm

As shown in Appendix C, the proposed problem can be simplified into the RCPSP with the objective of makespan minimization. As the latter is known to be *NP*-hard in the strong sense [32], [33], the proposed proactive scheduling problem with activity splitting is *NP*-hard in the strong sense as well, which makes the achievement of optimal solutions a computationally difficult proposition, especially for large projects. For this reason, we use a well-known metaheuristic, i.e., a genetic algorithm as introduced by Holland [34], to solve the problem. We choose the genetic search methodology for two reasons. Firstly, this technique has been successfully applied to many project scheduling problems [24], [25], [29], [30], [35], [36], and second, it is easy to generate activity splitting at each iteration by using the crossover operator.

Genetic algorithms work with a “population” of individuals. In our algorithm, we set the size of the population as μ, the individual of which can be initially generated by a procedure called IGP. At each iteration, which is denoted as $iter$, the best φ individuals of the population in terms of fitness (objective function value) are chosen to be included in the population of the next iteration, while $(\mu - \varphi)$ individuals of the population are selected following the roulette wheel sampling method to generate children with the aid of a crossover operator called CRP. Then, a mutation operator called MTP is used to apply a certain change to the generated children. Each child will be decoded into a solution using the procedure DCP. If it is feasible, the solution will be buffered with the procedure BFP and improved with a local search procedure that includes three operators called LSP_1, LSP_2, and LSP_3, respectively. As far as the termination criterion of the developed genetic algorithm is concerned, we define δ as the required number of iterations and stop the algorithm once δ is reached. It is noteworthy that we work with the notion of life span to solve the problem of super-individuals. Super-individuals far exceed, in fitness, other solutions of the population, and their existence might result in premature convergence to a local optimum. We set the life span of an individual at "birth" at 0. At each iteration, the life span of each surviving individual is increased by 1. When the life span reaches a certain number, $maxlife$, the individual dies and is replaced by a newly generated individual with the aid of the procedure IGP.

1) Solution Representation

Referring to [24], [31], we use three lists below to codify the solutions, the length of which is denoted as $nsub$.

- Subactivity list (L): This list is the sequence of subactivities. The j-th element in L represents the subactivity $L_j = (i, v)_j$.
- Duration list (DL): This list stores the duration $dur_{(i,v)_j}$ of the corresponding subactivity $(i, v)_j$ in L.
- Buffer list (BL): This list indicates which subactivities should be buffered and by how much their finish times can be delayed beyond their earliest finish times as dictated by the serial schedule generation scheme (SSGS). For convenience, let $buf_{(i,v)_j}$ denote the buffer of the corresponding subactivity $(i, v)_j$ in L. Note that $buf_{(i,v)_j}$ represents the inserted buffer, which is different from $FS_{(i,v)_j}$.

Given the above lists, a solution can be obtained using a decoding approach, which is an extension of SSGS and is described in Algorithm 1 in Appendix D.

2) Objective Function

For a solution that is represented by the combination of the three lists, the key to calculating its objective function value is to compute the free slack $FS_{i,v}$ of each subactivity. Once they are obtained, the objective function value can be easily computed based on the formula (1). We develop a procedure to compute the free slack of every subactivity, which is an extension of the procedure developed by Lambrechts *et al.* [31] and is indicated in Algorithm 2 in Appendix D.

As the decoded schedule may cause a project deadline violation, we transform the deadline constraint into a soft constraint that is based on a deadline feasibility test function DFT, which is defined as $DFT = \max\{0, s_{n,1} - D\}$. During the searching process, if the DFT of a solution is greater than 0, the objective function value of the solution will be penalized based on the following formula:

$$Robu = \sum_{i=1}^{n}[w_i \left(\sum_{v=1}^{V_i}\sum_{b=1}^{FS_{i,v}} e^{-b}\right)] - np \cdot nc \cdot DFT.$$

Here, np is the penalty factor, and nc denotes the number of iterations that are used by the genetic algorithm since the last major improvement was found.

3) Buffering

For a feasible solution, we use a procedure called BFP, which is described in Algorithm 3 in Appendix D, to insert enough buffers into the schedule to improve its robustness, which serves as a local search of the buffer list. We firstly select a subactivity randomly and add one unit of time buffer to that. Then we calculate the objective function value $Robu'$ of the improved solution. If the deadline constraint is violated or the objective function value has not been improved, the number of failure times ζ that is initialized at zero will increase by one. If ζ reaches a predefined maximum allowed number Z of failures, the procedure ends. Otherwise, another subactivity is chosen and the procedure continues.

4) Initial Population Generation

The individual g of the initial population can be generated through the procedure IGP, which is described in Algorithm 4 in Appendix D. To decide whether to split the activities in the initial solution, we take the constraints of the maximum number of splitting and the minimum execution time into consideration. If the two constraints are satisfied, then we generate random numbers and compare them with a predefined parameter $itrpt$ to make the decision of activity splitting. Note that this procedure builds an individual in $nsub$ iterations, where $nsub$ is unknown until the end of the procedure.

5) Crossover

Children can be generated by operating on the selected individuals with the aid of a crossover procedure called CRP. This procedure is described in Algorithm 5 in Appendix D, which is similar to the one that is developed by Ballestín *et al.* [24] except that we now have a third list called buffer list. In our procedure, we copy the same proportion of time buffer of the parent to the child as that of the duration. Note that the selected individuals are randomly paired as parents, and each of them can be a father or a mother.

6) Mutation

We make a change on the children with the procedure MTP, which is described in Algorithm 6 in Appendix D. We must emphasize that it is a deliberate choice that the mutation operation only considers the operators of changing the sequences and time buffers of the subactivities and does not introduce more operators. We considered many operators such as introducing more activity splitting, merging some subactivities, and changing the duration of subactivities. However, the preliminary tests with such operators did not lead to improved results. A reason could be that the local search procedure that is developed in the next section plays the same roles as those of these operators. For example, the procedures LSP_1 and LSP_3 can be regarded as operations that introduce more splitting of activities, and the procedure LSP_2 is structured to change the duration of the subactivities.

7) *Local Search*

For each feasible child, we adopt a local search procedure that includes three operators called LSP_1, LSP_2, and LSP_3, respectively, to improve its schedule robustness. The operator LSP_1, which is based on Property 1 and described in Algorithm 1, facilitates the discovery of new periods for activities to be executed. Let $C_1 = \{T_1, T_2, \cdots, T_c\}$ denote the set of feasible periods, as defined in Definition 1, and let $C_2 = \{(i,v) \mid dur_{i,v} - \theta_i \geq 2\varepsilon_i \text{ and } V_i \leq \eta_i\}$ represent the set of divisible subactivities, as defined in Definition 2.

Algorithm 1. Pioneering: $Robu' = LSP\_1\ (L, DL, BL)$

$FS_{(i,v)'_j} = FSP\ (L, DL, BL)$, obtain V_i $(i \in N)$
FOR $i = 2$ TO $n-1$ DO
  IF $\eta_i > 0$ AND $d_i \geq 2\varepsilon_i$ THEN
    Obtain the sets C_1 and C_2
    WHILE $C_1 \neq \emptyset$ AND $C_2 \neq \emptyset$ AND $V_i \leq \eta_i$ DO
      Choose the period T_1 from C_1 and one subactivity (i,v) from C_2
      Generate dd from $[\varepsilon_i, \min\{dur_{i,v} - \varepsilon_i - \theta_i,\ \xi_T - \theta_i\}]$
      Update the sets C_1 and C_2, $V_i = V_i + 1$
      $dur_{i,v} = dur_{i,v} - dd, dur_{i,V_i} = dd + \theta_i, FS_{i,v} = FS_{i,v} + dd, FS_{i,V_i} = \xi_{T_1} - dd - \theta_i$
    END WHILE
  END IF
END FOR
Calculate the objective function value $Robu'$ of the improved solution

The operator LSP_2, which is based on Property 2 and described in Algorithm 2, is used to balance the length of durations between two subactivities of one activity. Let $C_3 = \{(i,v) \mid dur_{i,v} - \theta_i \geq 2\varepsilon_i \text{ and } FS_{i,v} \leq 1 + \theta_i\}$ denote the set of subactivities that are divisible but not abundant in free slacks and let $C_4 = \{(i,v) \mid dur_{i,v} - \theta_i < 2\varepsilon_i \text{ and } FS_{i,v} \geq 2 + \theta_i\}$ represent the ones that are just the reverse.

Algorithm 2. Balancing: $Robu' = LSP\_2\ (L, DL, BL)$

$FS_{(i,v)'_j} = FSP\ (L, DL, BL)$
FOR $i = 2$ TO $n-1$ DO
  IF $\eta_i > 0$ AND $d_i \geq 2\varepsilon_i$ THEN
    Obtain the sets C_3 and C_4, $NS = \min\{|C_3|, |C_4|\}$
    FOR $q = 1$ TO NS DO
      Choose one subactivity (i,v) from C_3 and another one (i,p) from C_4
      $dur_{i,v} = dur_{i,v} - 1,\ dur_{i,p} = dur_{i,p} + 1$
      $FS_{i,v} = FS_{i,v} + 1,\ FS_{i,p} = FS_{i,p} - 1$
      Update C_3 and C_4
    END FOR
  END IF
END FOR
Calculate the objective function value $Robu'$ of the improved solution

The operator LSP_3, which is based on Lemma 1 and described in Algorithm 3, is used to divide one subactivity into subactivities that specifically share the buffer of the original subactivity as equally as possible. For the sake of description, let $C_5 = \{(i,v) \mid dur_{i,v} \geq 2\varepsilon_i \text{ and } FS_{i,v} \geq 2\}$ denote the set of subactivities that are divisible and abundant in free slacks.

Algorithm 3. $Robu' = LSP\_3\ (L, DL, BL)$

$FS_{(i,v)'_j} = FSP\ (L, DL, BL)$, obtain V_i $(i \in N)$
FOR $i = 2$ TO $n-1$ DO
  IF $\eta_i > 0$ AND $d_i \geq 2\varepsilon_i$ THEN
    Obtain the set C_5
    WHILE $C_5 \neq \emptyset$ AND $V_i \leq \eta_i$ DO
      Choose one subactivity (i,v) from C_5, $num_{i,v} = \min\left\{\left[\frac{dur_{i,v} - \theta_i}{\varepsilon_i}\right], \frac{FS_{i,v} + \theta_i}{1 + \theta_i},\ \eta_i - V_i + 2\right\}$
      Divide the subactivity (i,v) into $num_{i,v}$ parts whose free slacks are as equal as possible and durations are $[dur_{i,v} - (num_{i,v} - 1) \cdot \varepsilon_i], \varepsilon_i + \theta_i, \cdots, \varepsilon_i + \theta_i$ respectively
      Update V_i and the set C_5
    END WHILE
  END IF
END FOR
Calculate the objective function value $Robu'$ of the improved solution

IV. Computational Results

A. *Experimental Design*

Based on the three developed local search operators, four different versions of the genetic algorithm are presented. For the sake of description, we represent the genetic algorithm without any local search operator as GA, the genetic algorithm with the operator LSP_1 as GA-LSP1, the genetic algorithm with operators LSP_1 and LSP_2 as GA-LSP12, and the genetic algorithm with all the three operators as GA-LSP123, respectively, in the remainder of the paper. To evaluate the effectiveness of the proposed genetic algorithms, we propose the use of CPLEX as a benchmark to optimally solve the established model. Referring to the methods that are proposed to reduce zero-one polynomial formulations to zero-one linear formulations [37], the proposed non-linear model can be linearized, just as shown in Appendix E. As many variables and constraints are introduced into the model, it may take much time to solve the problem. However, there is no loss of the quality of the solutions for the problem and therefore it is enough for the sake of comparison of effectiveness. Note that we can use CPLEX to directly represent the algorithm that is conducted by the software. The aim of our experiment is not only to test the effectiveness of the three local search operators by comparing the performance of different versions of the genetic algorithm, but also to validate the performance of the genetic algorithm developed in this paper against CPLEX. Besides, it is expected to draw conclusions based on an analysis of the results.

The five algorithms are tested on five instance sets that are constructed by the ProGen project generator [38], [39], which is classified by three parameters, i.e., network complexity (NC), resource factor (RF), and resource strength (RS). Specifically, the instances with 6 or 8 or 10 non-dummy activities, denoted as J6, J8, and J10, are generated by ourselves using the ProGen generator while the instances with 30 or 60 non-dummy activities, denoted as J30 and J60, are randomly (the first and the sixth instances out of the ten provided instances) chosen from the Project Scheduling Problem Library (PSPLIB), which is also generated by the ProGen generator [39]. The five sets consist of 48×2×5=480 instances, and the parameter setting that is used to generate instances is described in Table II. Note that in consideration of the feasibility of the instance generation, we choose a different level setting of network complexity for J6, J8, and J10 from the one for J30 and J60. As for the other parameters in our problem, such as w_i, θ_i, η_i, and ε_i, we generate them randomly to cover all the cases in practice. The parameter settings of these parameters can be found in Table II and Table III.

In our experiment, the project due date D of each instance is set at $C_{\max}^{\mathrm{RCPSP}}(1+\alpha)$ where $C_{\max}^{\mathrm{RCPSP}}$ represents the minimum makespan that is optimally solved by CPLEX under a deterministic, indivisible, and non-setup-time environment, and the due date factor α is a parameter that is chosen by the project

1

TABLE II
PARAMETER SETTINGS THAT ARE USED TO GENERATE THE DATA SET

Parameter	Setting
Number of non-dummy activities	6, 8, 10, 30, 60
Network complexity, NC	1.2, 1.5, 1.8 for the sets J6, J8, and J10 1.5, 1.8, 2.1 for the sets J30 and J60
Resource factor, RF	0.25, 0.50, 0.75, 1.00
Resource strength, RS	0.2, 0.5, 0.7, 1.0
Number of instances for each combination of parameters under a given number of non-dummy activities	2
Number of initial or terminal activities	3
Maximal number of successors or predecessors	3
Activity duration, d_i	Randomly selected from interval [1, 10]
Number of resource types, K	4
Resource amounts required by activities, $r_{i,k}^{\rho}$	Randomly selected from interval [1, 10]
Weights of non-dummy activities, w_i	Randomly selected from interval [1, 10]

TABLE III
LEVELS OF THE KEY PARAMETERS

Parameter	Level	Value	
Project due date factor α	1-3	20%, 30%, 40%	
Setup times of non-dummy activities θ_i	1	$\theta_i = 0.8 * c * d_i$	
	2	$\theta_i = 1.0 * c * d_i$	
	3	$\theta_i = 1.2 * c * d_i$	
Combination of η_i and ε_i	1	$\eta_i = \min\{d_i - 1, a\}$	$\varepsilon_i = 1$
	2	$\eta_i = \min\{d_i - 1, 1.5a\}$	$\varepsilon_i = 1$
	3	$\eta_i = \min\{d_i - 1, 2a\}$	$\varepsilon_i = 1$
	4	$\eta_i = [d_i/\varepsilon_i] - 1$	$\varepsilon_i = \max\{1, b\}$
	5	$\eta_i = [d_i/\varepsilon_i] - 1$	$\varepsilon_i = \max\{1, 0.7b\}$
	6	$\eta_i = [d_i/\varepsilon_i] - 1$	$\varepsilon_i = \max\{1, 0.4b\}$
	7	$\eta_i = d_i - 1$	$\varepsilon_i = 1$

TABLE V
PERFORMANCE OF THE FOUR VERSIONS OF THE GENETIC ALGORITHM

Set	Version	AOV	APB (%)	API (%)	ARI (%)	ACT (s)	ACT' (s)	AOG' (%)
J6	GA	30.21	73.31	0.00	0.00	0.12	0.24	2.79
	GA-LSP1	30.23	74.11	0.23	15.94	0.12	0.24	2.76
	GA-LSP12	30.28	76.57	3.04	4.89	0.12	0.24	2.70
	GA-LSP123	30.49	96.36	10.07	9.40	0.12	0.20	1.89
J8	GA	40.00	44.71	0.00	0.00	0.16	0.39	2.98
	GA-LSP1	40.07	46.15	0.48	17.31	0.16	0.39	2.92
	GA-LSP12	40.21	50.15	3.80	5.44	0.17	0.38	2.88
	GA-LSP123	40.92	91.34	11.28	11.47	0.17	0.29	1.70
J10	GA	47.62	26.74	0.00	0.00	0.19	0.49	3.39
	GA-LSP1	47.77	28.04	1.02	9.41	0.20	0.49	3.01
	GA-LSP12	48.05	31.40	4.69	5.16	0.19	0.48	3.13
	GA-LSP123	49.38	87.12	12.68	8.45	0.20	0.34	2.00
J30	GA	158.83	3.90	0.00	0.00	3.12	9.44	6.88
	GA-LSP1	162.52	6.37	3.44	5.67	3.21	9.13	5.54
	GA-LSP12	162.83	6.66	8.55	3.60	3.13	9.10	5.69
	GA-LSP123	175.05	83.50	17.98	6.33	3.32	5.22	2.41
J60	GA	300.98	1.82	0.00	0.00	9.35	28.80	9.57
	GA-LSP1	318.10	6.30	10.85	4.23	9.73	26.32	6.40
	GA-LSP12	318.58	6.07	17.12	3.60	9.47	25.51	6.63
	GA-LSP123	349.83	85.81	30.16	7.01	10.04	13.69	4.24
Avg	GA	*115.53*	*30.10*	*0.00*	*0.00*	*2.59*	*7.87*	*5.12*
	GA-LSP1	*119.74*	*32.19*	*3.20*	*10.51*	*2.68*	*7.31*	*4.13*
	GA-LSP12	*119.99*	*34.17*	*7.44*	*4.54*	*2.62*	*7.14*	*4.21*
	GA-LSP123	*129.13*	*88.83*	*16.43*	*8.53*	*2.77*	*3.95*	*2.45*

manager and constitutes the trade-off between project stability and project duration [14]. The value of the four key parameters, i.e., α, θ_i, η_i, and ε_i, is set at certain levels, as shown in Table III, where parameter c denotes a decimal that is randomly selected from [1/10, 1/8] and parameters a and b respectively denote random numbers that are selected from [0, $d_i - 1$] and [1, d_i]. Consequently, a full factorial experiment of the four parameters results in 3×3×7=63 replicates for each instance and 480×63 = 30240 ones overall.

The following ten indices are defined to evaluate the performance of the algorithms. Specifically, the first seven indices are used to compare the performance of the four different versions of the genetic algorithm, and the last three indices are additionally designed to make a comparison of the performance between the genetic algorithm and CPLEX.

- AOV: Average objective function value.
- APB: The percentage of instances for which the algorithm finds a solution that is equal to the best solution known, i.e., the best one among the solutions that are found by the four developed versions of the genetic algorithm – GA, GA-LSP1, GA-LSP12, and GA-LSP123.
- API: The percentage of solutions that are improved after using the local search procedure.
- ARI: Average rate of improvement in terms of the objective function value after using the local search procedure.
- ACT: Average computing time.
- ACT': Average computing time to solve the problems to obtain the best solutions known.
- AOG': Average gap in terms of the objective function values of the worse solutions that are obtained by a specific version of the genetic algorithm compared with those of the best solutions known.
- AOG: Average gap in terms of the objective function values of the worse solutions that are obtained by the genetic algorithm compared with those of the corresponding solutions that are obtained by CPLEX.
- APN: The percentage of instances that cannot be solved by CPLEX within a predefined time limit.
- AWS: The percentage of instances in which worse solutions are obtained by the developed genetic algorithm than by CPLEX.

In our experiment, the developed algorithms are programmed in the C++ language, implemented in Microsoft Visual Studio 2013 and executed on a DELL OptiPlex 3040MT with 3.20 GHz clock-pulse and 8G RAM.

B. Parameter Selection

TABLE IV
VALUES OF PARAMETERS FOR INSTANCE SETS

Set	μ	φ	Z	$maxlife$	np	$itrpt$	$pmut$	δ
J6, J8, and J10	64	15	2	7	25	0.4	5%	80
J30	64	11	2	7	30	0.4	5%	350
J60	64	15	2	9	50	0.4	4%	450

Our developed genetic algorithm allows for different choices of eight parameters. With a focus on the value of AOV, we performed a preliminary experiment to choose the best combination of parameters. This experiment tests the instances whose project due date factor α is set at 30%, setup time is set at level 1, and the combination of η_i and ε_i is set at level 4. According to the results of the preliminary test, the parameters are set at different values to solve different instance sets, as shown in Table IV.

C. Performance of the Developed Genetic Algorithm

1) Comparison of the Four Different Versions of the Genetic Algorithm

The results of the performance of the four developed genetic algorithms on the five instance sets are presented in Table V, where the italic numbers in the four bottom rows represent the average values of the five instance sets. It is noteworthy that the five left indices are used to measure the performance of the genetic algorithms that stop after a predefined number of iterations while the two right ones are used to measure the performance of the genetic algorithms that stop once obtaining the best solutions known. From the table, we observe that for different instance sets the conclusion is almost the same in terms of the performance of the four versions of the genetic algorithm. The indices AOV and APB of GA-LSP1 are higher than those of GA, which verifies a better performance of GA-LSP1 compared with GA. This is not surprising because the operator LSP_1 is added in GA-LSP1, which on average improves the objective function values of 3.20 percent of the solutions by 10.51%. Similarly, the effectiveness of the operator LSP_2 can be analyzed by comparing the versions GA-LSP12 and GA-LSP1. On average, GA-LSP12 performs better than GA-LSP1 in terms of AOV, APB, and ACT. Furthermore, we find that GA-LSP123, followed by GA-LSP12, GA-LSP1, and GA, performs the best with the highest average objective function value (AOV) and the highest average percentage of the best solutions (APB). Corresponding with the highest value of APB, GA-LSP123 takes the least time to solve the problems again, reaching a smallest average gap of the objective function values compared with those of the best solutions known. Most of the success is due to the application of the operator LSP_3, which on average improves the objective function values of 16.43 percent of the solutions by 8.53%. Compared with the operators LSP_1 and LSP_2, LSP_3 is much more effective as there is a sharp increase of AOV and APB once it is included in the genetic algorithm. In summary, the three developed local search operators improve the solution robustness of the baseline schedules, although it takes a somewhat longer computing time to solve the problems. Thus, GA-LSP123 is the most promising version for the problem among the four presented genetic algorithms, which can be used to compete with a commercial mathematical programming solver next.

2) Comparison of the Performance between the Genetic Algorithm and Commercial Software

To test the effectiveness of the algorithm that is developed in this paper, we conduct an experiment to compare the performance between GA-LSP123 and a commercial mathematical programming solver (CPLEX). In this experiment, we predefine a maximum period of one hour for CPLEX to solve each instance. This means that even though one instance is not solved optimally by that time, we end the algorithm and save the outcome that has been obtained thus far, which includes the best solution, the objective function value, and the computing time. Because it is difficult for CPLEX to solve the problems with a lot of non-dummy activities, we only choose to test the three instance sets J6, J8, and J10.

TABLE VI
PERFORMANCE OF GA-LSP123 AND CPLEX

Set	ACT (s)		APN	AWS	AOG
	GA-LSP123	CPLEX	(%)	(%)	(%)
J6	0.12	1592.04	42.21	2.63	2.30
J8	0.17	1766.92	46.93	8.09	1.37
J10	0.20	2022.72	54.17	16.77	1.33
Avg	*0.16*	*1793.89*	*47.77*	*9.16*	*1.67*

The results of the experiment can be found in Table VI. From the table, we can see that the number of instances that cannot be solved by CPLEX in the predefined period is very high and increases quickly with an increasing number of non-dummy activities. Simply put, CPLEX requires a great deal of time to solve the problem. This is not surprising because many variables and constraints are introduced during the linearization process of the proposed scheduling model, which results in the difficulty of computing problems for CPLEX. By contrast, GA-LSP123 is much more efficient, with a very small computing time. Although GA-LSP123 cannot solve some instances as optimally as CPLEX, the percentage of these instances is very small, and it is acceptable of the average gap between the objective function values of the solutions for these instances that are solved by GA-LSP123 and the corresponding ones that are solved by CPLEX.

D. Sensitivity Analysis of the Key Parameters

Firstly, we investigate the effect of different levels of the combination of η_i and ε_i on solution robustness for the five instance sets. In addition to the seven levels of the divisible case, we take level 8, which represents the indivisible case, into account. The results are described in Table VII where for each instance set the italic numbers in the second row from the bottom represent the average values of the divisible case. From the table, two main phenomena can be observed. The first one is that the average objective function value under the divisible case is significantly higher than the corresponding values under the indivisible case. This indicates that activity splitting is beneficial for generating more robust baseline schedules that are likely to have lower adjustment costs during project execution. Compared with the classic proactive scheduling models where activity splitting is not allowed, this paper offers a new method to improve schedule robustness when activity splitting is allowed and generates better solutions to project management. This phenomenon can be explained as follows. When activities can be split, it will be more flexible for project managers to schedule activities at the design stage of the baseline schedules, which may help to obtain higher solution robustness. Essentially, the solution space of the divisible case is extended because of the constraint relaxation. The second phenomenon is that the average objective function value increases with the growth of η_i or the decline of ε_i. Activities can be split more frequently with a higher value of η_i or with a lower value of ε_i, which improves the scheduling feasibility, and thus this is beneficial for obtaining a higher objective function value.

Secondly, we investigate the influence of the key parameter θ_i on the index AOV for the five instance sets. The results are shown in Fig. 3, from which we can see that the growth of θ_i has a negative effect on the average objective function value. This is because there will be less space for inserting time buffers when taking more setup times into account.

TABLE VII
EFFECT OF DIFFERENT LEVELS OF THE COMBINATION OF η_i AND ε_i

Set	Case	Level	η_i	ε_i	AOV	ACT
J6	Divisible	1	2.38	1.00	28.91	0.12
		2	2.92	1.00	29.86	0.12
		3	3.38	1.00	31.26	0.13
		4	1.17	3.26	24.49	0.11
		5	2.17	2.06	29.45	0.12
		6	3.43	1.33	33.94	0.13
		7	4.66	1.00	35.54	0.14
		Avg	*2.87*	*1.52*	*30.49*	*0.12*
	Indivisible	8	0.00	5.66	16.43	0.09
J8	Divisible	1	2.31	1.00	38.83	0.17
		2	2.84	1.00	40.81	0.17
		3	3.26	1.00	42.62	0.17
		4	1.03	3.26	31.93	0.15
		5	2.03	2.06	38.73	0.16
		6	3.25	1.34	45.17	0.18
		7	4.49	1.00	48.33	0.19
		Avg	*2.74*	*1.52*	*40.92*	*0.17*
	Indivisible	8	0.00	5.49	21.56	0.12
J10	Divisible	1	2.15	1.00	47.01	0.20
		2	2.65	1.00	48.94	0.20
		3	3.03	1.00	51.12	0.21
		4	1.07	3.04	38.93	0.18
		5	2.02	1.93	47.05	0.20
		6	3.22	1.28	54.50	0.22
		7	4.25	1.00	58.08	0.22
		Avg	*2.63*	*1.46*	*49.38*	*0.20*
	Indivisible	8	0.00	5.25	26.92	0.14
J30	Divisible	1	2.17	1.00	167.22	3.21
		2	2.63	1.00	176.05	3.30
		3	3.04	1.00	184.71	3.40
		4	1.02	3.22	132.16	2.82
		5	1.97	2.05	163.02	3.18
		6	3.20	1.34	192.29	3.54
		7	4.41	1.00	209.88	3.77
		Avg	*2.63*	*1.52*	*175.05*	*3.32*
	Indivisible	8	0.00	5.41	84.88	2.45
J60	Divisible	1	2.27	1.00	334.81	9.37
		2	2.76	1.00	353.98	9.91
		3	3.19	1.00	370.93	10.39
		4	1.07	3.25	267.95	8.24
		5	2.03	2.06	328.29	9.37
		6	3.29	1.34	379.33	10.73
		7	4.52	1.00	413.52	12.23
		Avg	*2.73*	*1.52*	*349.83*	*10.03*
	Indivisible	8	0.00	5.52	169.18	6.95

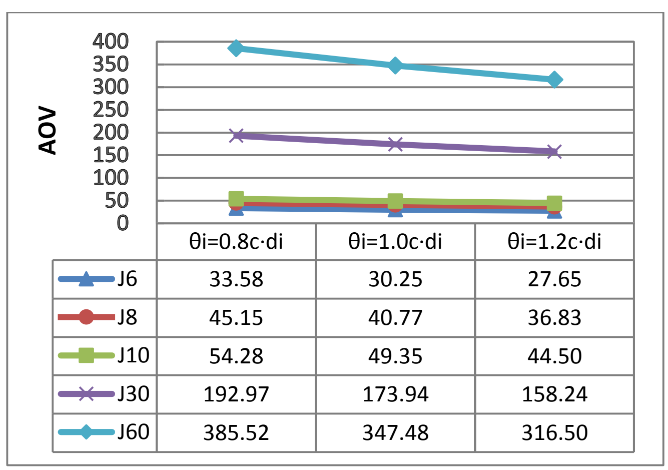
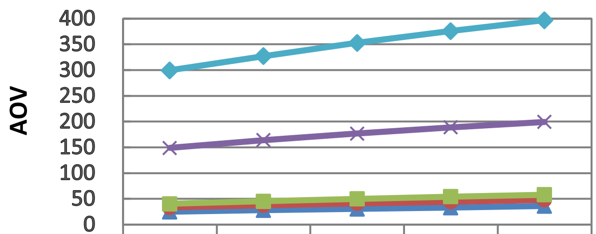


Fig. 3. The influence of the key parameter θ_i on AOV.

In addition, we investigate the influence of the key parameter α on the index AOV for the five instance sets, and we take two more levels of α, 25% and 35%, into account. The results are shown in Fig. 4, from which we can see that the growth of α has a positive effect on the average objective function value. This is reasonable because there will be more inserted buffers in the schedule as the project due date constraint becomes less strict.



	α=20%	α=25%	α=30%	α=35%	α=40%
J6	24.84	27.92	30.58	33.00	36.07
J8	33.50	37.55	41.25	44.55	48.00
J10	40.26	45.44	49.95	54.26	57.92
J30	148.85	164.22	176.79	188.93	199.51
J60	299.68	327.29	352.77	375.96	397.05

Fig. 4. The influence of the key parameter α on AOV.

V. CONCLUSIONS

This paper presents a proactive resource- constrained project scheduling problem with activity splitting where each activity can be split at discrete time instants under the constraints of a maximum number of splitting and a minimum period of continuous execution. Besides, in this problem setup times are considered. Based on the analysis of the established model, two properties and one lemma are proposed and applied in our developed genetic algorithm to improve the local search efficiency. In addition, we linearize the proposed model, making it solvable for commercial software. A computational experiment that is performed on data sets generated by the ProGen is designed and executed, from which the following conclusions are drawn:

1) The two developed properties and the proposed lemma can be used to maximize the objective function, and the genetic algorithm with a combination of the three local search operators performs the best.
2) Compared with commercial software, the developed genetic algorithm is much more efficient to solve the proposed scheduling problem, and the gap in terms of the objective function value is acceptable.
3) Due to the increase in flexibility of executing activities, activity splitting enhances the robustness of baseline schedules that are likely to have lower adjustment costs during project execution. Compared with the classic proactive scheduling models where activity splitting is not allowed, this paper offers a new method to improve schedule robustness when activity splitting is allowed and generates better solutions to project management.
4) With the growth of the maximum number of splitting, the decline in the minimum execution time, the decrease in the setup times, and the extension of the project due date, schedule robustness increases.

Note that the research in this paper is based on specific assumptions of activity splitting, so further research can provide support for quantitative decisions on project management under more complex and realistic conditions of activity splitting, such as cases in which activity splitting is allowed at arbitrary rational times. In addition, more effective and efficient algorithms can be developed to solve the proposed scheduling problem, and other efficient methods can be proposed to solve the zero-one polynomial formulations.

REFERENCES

[1] W. Herroelen and R. Leus, "Robust and reactive project scheduling: A review and classification of procedures," *Int. J. Prod. Res.*, vol. 42, pp. 1599-1620, 2004.
[2] W. Herroelen and R. Leus, "Project scheduling under uncertainty: Survey and research potentials," *Eur. J. Oper. Res.*, vol. 165, pp. 289-306, 2005.
[3] S. Van de Vonder, E. Demeulemeester, W. Herroelen, and R. Leus, "Proactive-reactive project scheduling trade-offs and procedures," in *Perspectives in Modern Project Scheduling*. New York, NY, USA: Springer, 2006, ch. 2, pp. 25-51.
[4] E. Demeulemeester and W. Herroelen, *Robust project scheduling*. Boston, USA: Now Publishers, 2011.
[5] S. Van de Vonder, E. Demeulemeester, W. Herroelen, and R. Leus, "The use of buffers in project management: The trade-off between stability and makespan," *Int. J. Prod. Econ.*, vol. 97, pp. 227-240, 2005.
[6] Ö. Hazır, M. Haouari, and E. Erel, "Robust scheduling and robustness measures for the discrete time/cost trade-off problem," *Eur. J. Oper. Res.*, vol. 207, pp. 633-643, 2010.
[7] Ö. Hazır, E. Erel, and Y. Günalay, "Robust optimization models for the discrete time/cost trade-off problem," *Int. J. Prod. Econ.*, vol. 130, pp. 87-95, 2011.
[8] W. Herroelen and R. Leus, "The construction of stable project baseline schedules," *Eur. J. Oper. Res.*, vol. 156, pp. 550-565, 2004.
[9] R. Leus and W. Herroelen, "Stability and resource allocation in project planning," *IIE Trans.*, vol. 36, pp. 667-682, 2004.
[10] S. Van de Vonder, E. Demeulemeester, and W. Herroelen, "Proactive heuristic procedures for robust project scheduling: An experimental analysis," *Eur. J. Oper. Res.*, vol. 189, pp. 723-733, 2008.
[11] O. Lambrechts, E. Demeulemeester, and W. Herroelen, "Proactive and reactive strategies for resource-constrained project scheduling with uncertain resource availabilities," *J. Scheduling*, vol. 11, pp. 121-136, 2008.
[12] O. Lambrechts, E. Demeulemeester, and W. Herroelen, "Time slack-based techniques for generating robust project schedules subject to resource uncertainty," *Ann. Oper. Res.*, vol. 186, pp. 443-464, 2011.
[13] M. A. Al-Fawzan and M. Haouari, "A bi-objective model for robust resource-constrained project scheduling," *Int. J. Prod. Econ.*, vol. 96, pp. 175-187, 2005.
[14] S. Van De Vonder, E. Demeulemeester, W. Herroelen, and R. Leus, "The trade-off between stability and makespan in resource-constrained project scheduling," *Int. J. Prod. Res.*, vol. 44, pp. 215-236, 2006.
[15] S. Van de Vonder, E. Demeulemeester, and W. Herroelen, "A classification of predictive-reactive project scheduling procedures," *J. Scheduling*, vol. 10, pp. 195-207, 2007.
[16] H. Chtourou and M. Haouari, "A two-stage-priority-rule-based algorithm for robust resource-constrained project scheduling," *Comput. Ind. Eng.*, vol. 55, pp. 183-194, 2008.
[17] F. Deblaere, E. Demeulemeester, and W. Herroelen, "Proactive policies for the stochastic resource-constrained project scheduling problem," *Eur. J. Oper. Res.*, vol. 214, pp. 308-316, 2011.
[18] P. Lamas and E. Demeulemeester, "A purely proactive scheduling procedure for the resource-constrained project scheduling problem with stochastic activity durations," *J. Scheduling*, vol. 19, pp. 409-428, 2016.
[19] E. Demeulemeester and W. Herroelen, "An efficient optimal solution procedure for the preemptive resource-constrained project scheduling problem," *Eur. J. Oper. Res.*, vol. 90, pp. 334-348, 1996.
[20] J. Buddhakulsomsiri and D. S. Kim, "Properties of multi-mode resource-constrained project scheduling problems with resource vacations and activity splitting," *Eur. J. Oper. Res.*, vol. 175, pp. 279-295, 2006.
[21] J. Buddhakulsomsiri and D. S. Kim, "Priority rule-based heuristic for multi-mode resource-constrained project scheduling problems with

resource vacations and activity splitting," *Eur. J. Oper. Res.*, vol. 178, pp. 374-390, 2007.
[22] J. Damay, A. Quilliot, and E. Sanlaville, "Linear programming based algorithms for preemptive and non-preemptive RCPSP," *Eur. J. Oper. Res.*, vol. 182, pp. 1012-1022, 2007.
[23] F. Ballestin, V. Valls, and S. Quintanilla, "Pre-emption in resource-constrained project scheduling," *Eur. J. Oper. Res.*, vol. 189, pp. 1136-1152, 2008.
[24] F. Ballestín, V. Valls, and S. Quintanilla, "Scheduling projects with limited number of preemptions," *Comput. Oper. Res.*, vol. 36, pp. 2913-2925, 2009.
[25] V. V. Peteghem and M. Vanhoucke, "A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem," *Eur. J. Oper. Res.*, vol. 201, pp. 409-418, 2010.
[26] M. Haouari, A. Kooli, E. Neron, and J. Carlier, "A preemptive bound for the Resource Constrained Project Scheduling Problem," *J. Scheduling*, vol. 17, pp. 237-248, 2014.
[27] A. Moukrim, A. Quilliot, and H. Toussaint, "An effective branch-and-price algorithm for the Preemptive Resource Constrained Project Scheduling Problem based on minimal Interval Order Enumeration," *Eur. J. Oper. Res.*, vol. 244, pp. 360-368, 2015.
[28] M. Vanhoucke and D. Debels, "The impact of various activity assumptions on the lead time and resource utilization of resource-constrained projects," *Comput. Ind. Eng.*, vol. 54, pp. 140-154, 2008.
[29] S. Quintanilla, Á. Pérez, P. Lino, and V. Valls, "Time and work generalised precedence relationships in project scheduling with pre-emption: An application to the management of Service Centres," *Eur. J. Oper. Res.*, vol. 219, pp. 59-72, 2012.
[30] M. Tavana, A. R. Abtahi, and K. Khalili-Damghani, "A new multi-objective multi-mode model for solving preemptive time-cost-quality trade-off project scheduling problems," *Expert Syst. Appl.*, vol. 41, pp. 1830-1846, 2014.
[31] O. Lambrechts, E. Demeulemeester, and W. Herroelen, "A tabu search procedure for developing robust predictive project schedules," *Int. J. Prod. Econ.*, vol. 111, pp. 493-508, 2008.
[32] J. Blazewicz, J. K. Lenstra, and A. H. G. R. Kan, "Scheduling subject to resource constraints: Classification and complexity," *Discrete Appl. Math.*, vol. 5, pp. 11-24, 1983.
[33] R. Leus, "The generation of stable project plans: Complexity and exact algorithms," PhD thesis, Katholieke Universiteit Leuven, Belgium, 2003.
[34] J. Holland, *Adaptation in Natural and Artificial Systems*: *An Introductory Analysis with Applications to Biology*, *Control and Artificial Intelligence*. Michigan, USA: University of Michigan Press, 1975.
[35] W. Huang and L. Ding, "Project-scheduling problem with random time-dependent activity duration times," *IEEE Trans. Eng. Manage.*, vol. 58, pp. 377-387, 2011.
[36] C. Fang, F. Marle, M. Xie, and E. Zio, "An integrated framework for risk response planning under resource constraints in large engineering projects," *IEEE Trans. Eng. Manage.*, vol. 60, pp. 627-639, 2013.
[37] L. J. Watters, "Reduction of integer polynomial problem to zero-one linear programming problems," *Oper. Res.*, vol. 15, pp. 1171-1174, 1967.
[38] R. Kolisch, A. Sprecher, and A. Drexl, "Characterization and generation of a general class of resource-constrained project scheduling problems," *Manage. Sci.*, vol. 41, pp. 1693-1703, 1995.
[39] R. Kolisch and A. Sprecher, "PSPLIB - A project scheduling problem library," *Eur. J. Oper. Res.*, vol. 96, pp. 205-216, 1997.

1 APPENDIX

A. More Details about the Literature on Proactive Scheduling

TABLE VIII
DETAILS ABOUT THE LITERATURE ON PROACTIVE SCHEDULING

Reference	Mode		Objective		Surrogate robustness measure	Slacks		Algorithm	
	Single	Multi-	Robustness	Others		Time	Resource	Exact	Heuristic
[5]	√			Trade-off		√			CC/BM & ADFF
[6]		√		Trade-off	√	√		A two-phase approach	
[7]		√		Trade-off	√	√		Benders decomposition	Tabu search
[8]	√		√		√	√			EWD1
[9]	√		√			Resource allocation		Brand & bound	
[10]	√		√			√			Multiple procedures
[11]	√		√			√	√		Priority rule based
[12]	√		√			√			Steepest descent
[13]	√			Trade-off	√	√			Tabu search
[14]	√			Trade-off		√			RFDFF
[16]	√			Trade-off	√	√			Priority rule based
[17]	√			Cost		√			STC + D heuristic
[18]	√		√		√	Confidence level		CCP method	
[31]	√		√		√	√			Tabu search
Mine	√		√		√	√			Genetic algorithm

B. More Details about the Literature on the Resource-constrained Project Scheduling Problem with Activity Splitting

TABLE IX
DETAILS ABOUT THE LITERATURE ON THE RESOURCE-CONSTRAINED PROJECT SCHEDULING PROBLEM WITH ACTIVITY SPLITTING

Reference	Mode		Minimum execution time	Maximum splitting times	Setup times	Objective		Algorithm	
	Single	Multi-				Makespan	Others	Exact	Heuristic
[19]	√					√		Branch & bound	
[20]		√				√		Branch & bound	
[21]		√				√			Priority-based
[22]	√					√			Local search
[23]	√					√			Improved RCPSP
[24]	√		√	√		√			Evolutionary
[25]		√				√			Genetic algorithm
[26]	√					√		Lower bound	
[27]	√					√		Branch-and-price	
[28]		√				√		Branch & bound	
[29]	√						Cost		Genetic algorithm
[30]		√	√	√			Trade-off		Evolutionary
Mine	√		√	√	√		Robustness		Genetic algorithm

C. Simplification of the Proposed Problem

In the proposed problem, activities can be split into certain parts, which is decided by two parameters, the maximum splitting times η_i and the minimum continuous execution time ε_i. If we set $\eta_i = 0$ or $\varepsilon_i = d_i$ for each activity i, then the problem will be simplified into the proactive scheduling problem without activity splitting. As the dummy end activity is assumed to start and end at the project deadline, we can add an extra dummy activity $(n-1)$ in the project network $G' = (N, A')$ where $(n-1, n) \in A'$ and $(i, n-1) \in A', \forall (i, n) \in A$. Then we can set the weight of activity $(n-1)$ as 1 and the weights of all other activities as 0. In this way, the objective function (maximize $\sum_{i=1}^{n}[w_i(\sum_{v=1}^{V_i}\sum_{b=1}^{FS_{i,v}} e^{-b})]$) will be simplified to maximize $\sum_{b=1}^{FS_{n-1,1}} e^{-b}$, which is equivalent to minimizing the project makespan C_{max}. As there are also precedence and resource constraints in the model and there won't be a deadline constraint if we set the project deadline much bigger than C_{max}, the proposed problem can be simplified into the resource-constrained project scheduling problem (RCPSP) with the objective of makespan minimization.

D. More Details of the Procedures in the Genetic Algorithms

Algorithm 1. Decoding procedure: $s_{(i,v)_j} = DCP\ (L, DL, BL)$

$s_{(i,v)_1} = 0$
FOR $j = 2$ TO $nsub$ DO
  $s_{(i,v)_j} = \max_{(i,v)_h \in P_{(i,v)_j}}(s_{(i,v)_h} + dur_{(i,v)_h} + buf_{(i,v)_h})$
  WHILE $\exists k, t$: $\sum_{h \in S(t)} r_{h,k}^{\rho} > R_k^{\rho}$ $(k = 1, \cdots, K$ and $t = s_{(i,v)_j}, \cdots, s_{(i,v)_j} + dur_{(i,v)_j} + buf_{(i,v)_j} - 1)$ DO
    $s_{(i,v)_j} = s_{(i,v)_j} + 1$
  END WHILE
END FOR
$s_{(i,v)_{nsub}} = \max(s_{(i,v)_{nsub}}, D)$

Note: $P_{(i,v)_j}$ represents the set of predecessors of subactivity $(i, v)_j$.

Algorithm 2. Slack calculation: $FS_{(i,v)'_j} = FSP\ (L, DL, BL)$

$s_{(i,v)_j} = DCP\ (L, DL, BL)$
Obtain the list L'
$ES_{(i,v)'_1} = LS_{(i,v)'_1} = s_{(i,v)'_1}$, $FS_{(i,v)'_1} = 0$
FOR $j = 2$ TO $nsub$ DO
  $ES_{(i,v)'_j} = s_{(i,v)'_j}$, $LF_{(i,v)'_j} = \min\{ES_h | h \in S_{(i,v)'_j}\}$, $LS_{(i,v)'_j} = LF_{(i,v)'_j} - dur_{(i,v)'_j}$
  WHILE $\exists k, t$: $\sum_{h \in S(t)} r_{h,k}^{\rho} > R_k^{\rho}$ $(k = 1, \cdots, K$ and $t = ES_{(i,v)'_j}, \cdots, LF_{(i,v)'_j} - 1)$ DO
    $LF_{(i,v)'_j} = LF_{(i,v)'_j} - 1$, $LS_{(i,v)'_j} = LS_{(i,v)'_j} - 1$
  END WHILE
  $FS_{(i,v)'_j} = LS_{(i,v)'_j} - ES_{(i,v)'_j}$
END FOR

Note: L' represents the list of subactivities that are ordered according to their non-increasing completion times (the tiebreaker is the highest subactivity number). For convenience, $(i, v)'_j$ denotes the subactivity in position j of the ordered list L'. Additionally, $S_{(i,v)'_j}$, $ES_{(i,v)'_j}$, $LS_{(i,v)'_j}$, and $LF_{(i,v)'_j}$ respectively denote the set of immediate successors, the earliest starting time, the latest starting time, and the latest completion time of the subactivity $(i, v)'_j$.

Algorithm 3. Buffering: $Robu' = BFP\ (L, DL, BL)$

$FS_{(i,v)'_j} = FSP\ (L, DL, BL)$, $\zeta = 0$
Calculate the objective function value $Robu$
WHILE $\zeta \leq Z$ DO
  Choose one subactivity $(i, v)_j$ from the list L, and then $buf_{(i,v)_j} = buf_{(i,v)_j} + 1$
  $FS_{(i,v)'_j} = FSP\ (L, DL, BL)$, and calculate its new objective function value $Robu'$
  IF $s_{(i,v)_{nsub}} > D$ OR $Robu' \leq Robu$ THEN
    $\zeta = \zeta + 1$, $buf_{(i,v)_j} = buf_{(i,v)_j} - 1$
  ELSE
    $Robu = Robu'$
  END IF
END WHILE

Algorithm 4. Individual generation: $(L, DL, BL) = IGP(g)$

DO
  Initialize $Elig$ and the three lists, $j = 0$, $V_i = 0$ $(\forall i \in N)$, $leftd(i) = d_i$ $(\forall i \in N)$
  WHILE $Elig \neq \emptyset$ DO
    Select an activity i from $Elig$, $V_i = V_i + 1$, $j = j + 1$
    $L_j = (i, V_i)$, $buf_{(i,v)_j} = 0$, $nsub = j$
    Generate a random number m_1 between 0 and 1
    IF $m_1 > itrpt$ AND $V_i < \eta_i$ AND $leftd(i) \geq 2\varepsilon_i$ THEN
      Generate a random number m_2 from $[\varepsilon_i, leftd(i) - \varepsilon_i]$
      $dur_{(i,v)_j} = m_2 + \theta_i$, $leftd(i) = leftd(i) - m_2$
    ELSE
      $dur_{(i,v)_j} = leftd(i) + \theta_i$, update $Elig$
    END IF
  END WHILE
WHILE $(s_{(i,v)_{nsub}} > D)$
$Robu' = BFP(L, DL, BL)$

Note: Let $leftd(i)$ represent the number of duration units of activity i that have not yet been assigned (setup times are not included in $leftd(i)$), and let $Elig$, defined as $Elig = \{i \mid leftd(i) > 0$ and $leftd(j) = 0, (j, i) \in A\}$, be the set of eligible activities.

Algorithm 5. Crossover: $(\mathrm{L_C}, \mathrm{DL_C}, \mathrm{BL_C}) = \mathrm{CRP}(\mathrm{L_F}, \mathrm{DL_F}, \mathrm{BL_F}, \mathrm{L_M}, \mathrm{DL_M}, \mathrm{BL_M})$

Generate a random number m between 1 and $nsub_F$
Copy the first m elements of every list of the father to the child
Obtain V_i and $leftd(i)$ of the child after copy, $j = \sum_{i \in N}(\eta_i + 1)$
FOR $q = nsub_M$ TO 1 DO
  $i = L_q^M$, $d = dur_{(i,v)_q^M}$
  IF $leftd(i) > 0$ THEN
    IF $V_i \geq \eta_i$ OR $leftd(i) < d - \theta_i$ THEN
      $dur_{(i,v)_j^C} = leftd(i) + \theta_i$, $leftd(i) = 0$
    ELSE
      $dur_{(i,v)_j^C} = d$, $leftd(i) = leftd(i) - d + \theta_i$
    END IF
    $L_j^C = L_q^M$, $buf_{(i,v)_j^C} = buf_{(i,v)_q^M} \times [(dur_{(i,v)_j^C} - \theta_i)/(d - \theta_i)]$
    $V_i = V_i + 1$, $j = j - 1$
  END IF
END FOR
Erase the blank cells from L_C, DL_C, and BL_C

Note: The parameters that are labeled with F, M, and C respectively represent the subactivity list, the duration list, and the buffer list of the father, the mother, and the child.

Algorithm 6. Mutation: $(L_Q, DL_Q, BL_Q) = MTP(L, DL, BL)$

FOR $q = 1$ TO $pmut \cdot nsub$ DO
  Randomly generate a number m_3 from $\{0,1\}$, a number j from $[1, nsub]$
  IF $m_3 = 0$ THEN
    Calculate the possible positions $[a, b]$ of subactivity $(i, v)_j$ in the list L without causing the precedence constraint violation
    Generate a random number m_4 ($m_4 \neq j$) from $[a, b]$
    Place subactivity $(i, v)_j$ in position m_4 and update the lists
  ELSE
    Generate a random number m_5 from $\{0,1\}$
    IF $m_5 = 0$ THEN
      $buf_{(i,v)_j} = buf_{(i,v)_j} + 1$
    ELSE
      IF $buf_{(i,v)_j} \geq 1$ THEN
        $buf_{(i,v)_j} = buf_{(i,v)_j} - 1$
      END IF
    END IF
  END IF
END FOR

Note: $pmut$ represents the probability of mutation, and the parameters that are labeled by Q represent the subactivity list, the duration list, and the buffer list of the mutated individual.

E. Linearization of the Model

To conduct the linearization, we redefine V_i as the maximum number of subactivities of activity i, which is a constant value that is known in advance instead of being a decision variable. We use M to denote a large positive number and introduce U_i to represent the maximum number of free slacks of activity i, which is calculated as the length of the time window of activity i without the resource constraints under an indivisible scheduling environment. Then, the free slack $FS_{i,v}$ ranges from 0 to U_i. Additionally, five groups of binary variables are defined as follows.

$$y_{i,v} = \begin{cases} 0 & \text{if the duration of subactivity } (i,v) \text{ is zero} \\ 1 & \text{otherwise} \end{cases}$$

$$x_{i,v,u} = \begin{cases} 0 & \text{if free slack } u \text{ of subactivity } (i,v) \text{ is zero} \\ 1 & \text{otherwise} \end{cases}$$

$$\alpha_{i,v,t} = \begin{cases} 1 & \text{if } s_{i,v} \leq t \\ 0 & \text{otherwise} \end{cases}$$

$$\beta_{i,v,t} = \begin{cases} 1 & \text{if } s_{i,v} + dur_{i,v} + FS_{i,v} > t \\ 0 & \text{otherwise} \end{cases}$$

$$\gamma_{i,v,t} = \begin{cases} 1 & \text{if } \alpha_{i,v,t} = \beta_{i,v,t} = 1 \\ 0 & \text{otherwise} \end{cases}$$

There are seven groups of decision variables in the transformed linear model, i.e., $y_{i,v}$, $dur_{i,v}$, $s_{i,v}$, $x_{i,v,u}$, $\alpha_{i,v,t}$, $\beta_{i,v,t}$, and $\gamma_{i,v,t}$. Compared with those decision variables in the non-linear model, $y_{i,v}$ is used to replace V_i while $dur_{i,v}$ and $s_{i,v}$ stay the same. In addition, $x_{i,v,u}$ is used to take the place of the computation of the free slack, while $\alpha_{i,v,t}$, $\beta_{i,v,t}$, and $\gamma_{i,v,t}$ will decide the set of activities that are in progress at time t. Based on the above definitions, the non-linear scheduling model can be transformed into a linear one, as follows.

$$\text{Maximize } Robu = \sum_{i=1}^{n}\left[w_i\left(\sum_{v=1}^{V_i}\sum_{u=1}^{U_i} e^{-u} x_{i,v,u}\right)\right] \tag{1}$$

$$s_{1,1} = 0 \tag{2}$$

$$s_{i,V_i} + dur_{i,V_i} + FS_{i,V_i} \leq s_{j,1} \quad (i,j) \in A \tag{3}$$

$$s_{i,v} + dur_{i,v} + FS_{i,v} \leq s_{i,v+1} \quad \forall i;\ v = 1, \dots, V_i - 1 \tag{4}$$

$$s_{n,1} \leq D \tag{5}$$

$$\sum_{i=1}^{n}\left(r_{i,k}^{\rho} \sum_{v=1}^{V_i} \gamma_{i,v,t}\right) \leq R_k^{\rho} \quad \forall k, \forall t \tag{6}$$

$$\sum_{v=1}^{V_i} dur_{i,v} = d_i + \theta_i \sum_{v=1}^{V_i} y_{i,v} \quad \forall i \tag{7}$$

$$V_i = \eta_i + 1 \quad \forall i \tag{8}$$

$$dur_{i,v} \geq (\varepsilon_i + \theta_i) \times y_{i,v} \quad \forall i;\ v = 1,2,\dots,V_i \tag{9}$$

$$y_{i,v+1} \leq y_{i,v} \quad \forall i;\ v = 1,2,\dots,V_i - 1 \tag{10}$$

$$dur_{i,v} + FS_{i,v} \leq y_{i,v} \times M \quad \forall i, \forall v \tag{11}$$

$$\sum_{u=1}^{U_i} x_{i,v,u} = FS_{i,v} \quad \forall i, \forall v \tag{12}$$

$$M\left(\alpha_{i,v,t} - 1\right) \leq t - s_{i,v} < M \times \alpha_{i,v,t} \quad \forall i, \forall v, \forall t \tag{13}$$

$$M\left(\beta_{i,v,t} - 1\right) < s_{i,v} + dur_{i,v} + FS_{i,v} - t \leq M \times \beta_{i,v,t} \tag{14}$$

$$2\,\gamma_{i,v,t} \leq \alpha_{i,v,t} + \beta_{i,v,t} \leq \gamma_{i,v,t} + 1 \quad \forall i, \forall v, \forall t \tag{15}$$

$$FS_{i,v},\ dur_{i,v}, \text{ and } s_{i,v} \text{ are nonnegative integers} \tag{16}$$

$$y_{i,v}, x_{i,v,u}, \alpha_{i,v,t}, \beta_{i,v,t}, \gamma_{i,v,t} \in \{0, 1\} \quad \forall i, \forall v, \forall t \tag{17}$$

In the formulation, the objective function is transformed into a new linear one, while two constraints, (2), and (5), stay the same. In addition, six constraints, (3), (4), (6), (7), (8), and (9), are adjusted into new ones, and six constraints, from (10) to (15), are added. Specifically, $FS_{i,v}$ should be included in the precedence constraints (3) and (4). As the decision variable $x_{i,v,u}$ is used to decide the value of $FS_{i,v}$ through (12), it should also replace $FS_{i,v}$ in the objective function. In constraints (7), we now use $\sum_{v=1}^{V_i} y_{i,v}$ to represent the number of non-dummy subactivities. Because V_i now represents the maximum number of subactivities of activity i, which is calculated by constraints (8), there will be dummy subactivities whose durations and free slacks should be zero. Hence, constraints (9) force the duration of each subactivity to be at least its minimum execution time plus its setup time, but only if it is a non-dummy one. Moreover, constraints (10) and (11) ensure that the dummy subactivities are the last ones of each activity and that their duration and free slack are zero. Further, with three added constraints, which are shown in (13), (14), and (15), to describe the set $S(t)$ based on the definition $S(t) = \left\{i \middle| s_{i,v} \leq t < s_{i,v} + dur_{i,v} + FS_{i,v}\right\}$, the resource constraints are transformed into new ones, as stated in constraints (6).