J Sched (2016) 19:43–60
DOI 10.1007/s10951-015-0457-6

A genetic algorithm for the robust resource leveling problem

Hongbo Li[1,2,3] · **Erik Demeulemeester**[2]

Published online: 16 October 2015

Abstract The resource leveling problem (RLP) involves the determination of a project baseline schedule that specifies the planned activity starting times while satisfying both the precedence constraints and the project deadline constraint under the objective of minimizing the variation in the resource utilization. However, uncertainty is inevitable during project execution. The baseline schedule generated by the deterministic RLP model tends to fail to achieve the desired objective when durations are uncertain. We study the robust resource leveling problem in which the activity durations are stochastic and the objective is to obtain a robust baseline schedule that minimizes the expected positive deviation of both resource utilizations and activity starting times. We present a genetic algorithm for the robust RLP. In order to demonstrate the effectiveness of our genetic algorithm, we conduct extensive computational experiments on a large number of randomly generated test instances and investigate the impact of different factors (the marginal cost of resource usage deviations, the marginal cost of activity starting time deviations, the activity duration variability, the due date, the order strength, the resource factor and the resource constrainedness).

✉ Erik Demeulemeester
Erik.Demeulemeester@kuleuven.be

Hongbo Li
ishongboli@gmail.com

[1] School of Management, Shanghai University, Shanghai, China

[2] Research Center for Operations Management, Faculty of Business and Economics, KU Leuven, Hogenheuvelcollege, Naamsestraat 69, 3000 Leuven, Belgium

[3] School of Economics and Management, Beihang University, Beijing, China

Keywords Project scheduling · Robust resource leveling · Stochastic activity durations · Genetic algorithm

1 Introduction

In many project management situations, reducing the fluctuations in the pattern of expensive renewable resource usage over time is of crucial importance. This results in the resource leveling problem (RLP) which involves the determination of a project baseline schedule that specifies the planned activity starting times while satisfying both the precedence constraints and the project deadline constraint under the objective of minimizing the variation in the resource utilizations.

Many exact and heuristic procedures have been devised to solve the deterministic RLP. Exact approaches are mainly based on dynamic programming (Bandelloni et al. 1994), integer programming (Easa 1989; Hariga and El-Sayegh 2011), or branch-and-bound procedures (Ahuja 1976). Most of the existing heuristics rely on shifting activities within their time slacks or priority-rule methods (Burgess and Killebrew 1962; Wiest and Levy 1977; Harris 1990). In addition, Chan et al. (1996), Leu et al. (2000), and El-Rayes and Jun (2009) adopt a genetic algorithm to deal with the RLP. However, these authors do not present a computational performance analysis and only use small examples to test their algorithms. For studies with computational performance analysis, exact approaches based on a branch-and-bound procedure (Neumann and Zimmermann 2000; Gather et al. 2011) and mixed-integer programming (Rieck et al. 2012; Kreter et al. 2014) have been proposed. Rieck et al. (2012) solve instances with up to 50 activities to optimality for the first time. For heuristics, Neumann and Zimmermann (1999, 2000) devise polynomial heuristics and priority-rule-based heuristics for

the RLP. Metaheuristics based on tabu search (Neumann and Zimmermann 2000), genetic algorithm (Ponz-Tienda et al. 2013), an iterated greedy method (Ballestín et al. 2007), and path-relinking (Ranjbar 2013) are also studied. The path-relinking developed by Ranjbar (2013) outperforms the best available metaheuristic algorithms so far.

Most of the existing literature concentrates on the deterministic RLP. However, uncertainty is inevitable during project execution. Activity durations may be shorter or longer than planned, resources may break down, and due dates may change, etc. As a result, the baseline schedule generated by the deterministic RLP model tends to fail to achieve the desired objective. Very little work has been done to study the RLP under activity duration uncertainties. As early researchers, both Leu et al. (1999) and Leu and Hung (2002) minimize the sum of the absolute differences between the actual resource usage and the average resource usage under activity duration uncertainty. The difference of the two papers is that Leu et al. (1999) model uncertain activity durations as fuzzy numbers, while Leu and Hung (2002) treat uncertain activity durations as stochastic variables. Masmoudi and Haït (2013) investigate the fuzzy RLP with fuzzy activity durations and workload. The above-mentioned research all uses genetic algorithms (GAs) to solve their problems. These GAs adopt the same encoding type, in which a chromosome is represented as an activity starting time vector (the activity starting time is coded as a gene value). This type of encoding is not widely adopted in the project scheduling literature as it suffers from solution space explosion as the project duration increases. Resource leveling has also been studied as one objective in some multi-objective optimization literature. Zahraie and Tavakolan (2009) and Ashuri and Tavakolan (2012) study the multi-objective time-cost-resource optimization (TCRO) problem under fuzzy environments where they optimize project time, cost, and resource usage simultaneously. They also use the fuzzy set theory to model uncertainty parameters, such as time, cost, etc. To obtain a baseline schedule, a nondominated sorting genetic algorithm (NSGA-II) and a hybrid genetic algorithm–particle swarm procedure are used, respectively. These studies, however, do not provide any computational performance analysis, and only small example instances are considered. Li et al. (2015) present heuristics for producing scheduling policies for the resource leveling problem with stochastic activity durations. They conduct a computational performance analysis for projects with up to 90 activities.

In addition, a related research topic to the RLP is the resource loading problem. However, the resource loading problem focuses on the manufacturing environments and it concerns planning more than scheduling. Wullink et al. (2004) study the flexible resource loading problem under uncertainty in which one tries to minimize the expected total costs consisting of a non-regular resource cost and a tardiness penalty cost by assigning jobs to a number of resource groups in manufacturing environments. They consider uncertainties such as work content, resource capacity levels, resource requirements and the occurrence of an activity. Wullink (2005) addresses the robust resource loading problem under work content uncertainty with the objective of obtaining a robust feasible resource loading schedule and minimizing the non-regular resource cost.

The above-mentioned resource loading problem and our paper all tackle resource leveling and robustness. However, the robustness in Wullink et al. (2004) belongs to the *quality robustness* type (Herroelen and Leus 2004a; Demeulemeester and Herroelen 2011), where they make their model meet the customer order due dates as much as possible. On the other hand, the robustness achieved in Wullink (2005) is by means of splitting and shifting activities within predefined time windows in order to cope with a work content increase and such a kind of robustness is more related to *flexibility* (Herroelen and Leus 2004a). However, the robustness in our problem refers to the *solution robustness* type (Herroelen and Leus 2004a; Demeulemeester and Herroelen 2011) where we try to obtain stable activity start times in the baseline schedule that are minimally disrupted during project execution. Clearly, a baseline schedule with stable activity start times will reduce the schedule risks, offer degrees of freedom for rescheduling, improve the feasibility of executing the given activities, to name a few (Lambrechts et al. 2008). Besides, our model also protects the project due date by assigning the dummy end activity a rather high start time deviation cost which means that our model will also have a certain *quality robustness*.

The fundamental approaches for project scheduling in an uncertain environment mainly include stochastic project scheduling, fuzzy project scheduling, and robust project scheduling (Herroelen and Leus 2004a, 2005). Particularly, robust project scheduling (Demeulemeester and Herroelen 2011), as a recent research direction, aims at obtaining a stable baseline schedule that can absorb anticipated disruptions as much as possible during project execution. If the schedule would still be disrupted by unexpected events during project execution, reactive scheduling or rescheduling will be needed (Lambrechts et al. 2008; Tang et al. 2014; Van de Vonder et al. 2007a, b). It is worth noting that the vast majority of the research efforts devoted hitherto to robust project scheduling have been almost exclusively focused on the stability in the activity starting times. However, ensuring stable activity starting times is often not the only issue. In many cases where the pattern of resource usage is also our concern, it is desirable that the actually realized resource usage during project execution deviates a little from the planned resource usage level. In this paper, we consider reducing the positive deviation of the resource usage. In many cases, we want to keep the excessive usage of resources as low as possible in

order to avoid some unexpected situations such as overtime work and hiring inexperienced or expensive external manpower. At the same time, when uncertainties come into play, the stable predictive activity start times will make it easier to achieve the resource leveling objective for the resulting schedule.

To the best of our knowledge, there has been no research that considers both resource leveling and activity start time stability under activity duration uncertainty. The objective of this paper is to develop and validate a GA procedure for the robust RLP with the objective of obtaining robust baseline schedules that minimize the expected positive deviation of both resource utilizations and activity starting times under activity duration uncertainty.

The remainder of the paper is organized as follows. In Sect. 2, we describe the robust resource leveling problem under activity duration uncertainty. To solve this problem, we propose a dedicated GA in Sect. 3. In Sect. 4, we provide a description of our experimental set-up. In Sect. 5, the effectiveness of our genetic algorithm is demonstrated on a large number of randomly generated test instances and the impact of different factors is investigated. In the final section, we present our conclusions.

2 Problem statement

2.1 Robust resource leveling problem

The robust resource leveling problem can be described as follows. A project network $G = (N, A)$ is represented in the activity-on-node format, where the set of nodes N denotes the activities $N = \{1, \ldots, n\}$, and the set of directed arcs A represents the finish–start, zero-lag precedence relations $A \subseteq N \times N$. The nodes are topologically numbered from the single start node 1 to the single terminal node n, $n = |N|$, where nodes 1 and n represent two dummy activities. The activities are executed without preemption. The duration of each non-dummy activity i is stochastic and is denoted as the random variable $\boldsymbol{d}_i$. We assume that $\boldsymbol{d}_i$ follows a known distribution (the computational results presented in Sect. 5 will be based on a Beta distribution). Every non-dummy activity i requires r_{ik} renewable resources for resource type k per time period during execution ($k = 1, 2, \ldots, K$). The resource usage for resource type k during time period t is denoted as $\boldsymbol{u}_{kt} : \boldsymbol{u}_{kt} = \sum_{i \in A_t} r_{ik}$, where A_t is the set of activities that are in progress during time period t. Because $\boldsymbol{u}_{kt}$ is dependent on the start times of the activities and because the activity durations are stochastic variables, $\boldsymbol{u}_{kt}$ also becomes stochastic. Every resource type k has a weight c_k that denotes the penalty cost per unit of resource type k in the situation where the actual resource usage $\boldsymbol{u}_{kt}$ during execution exceeds the given resource usage $\bar{u}_k$. Every non-dummy activity i has a weight w_i that denotes the marginal cost of the positive deviation between the actual starting time $\boldsymbol{s}_i$ during execution and the planned starting time s_i in the baseline schedule. In both robust project scheduling problems and resource leveling problems, the project duration is usually predefined. Therefore, we also consider penalizing the project tardiness by introducing the weight of the dummy end activity w_n which denotes the cost of delaying the project completion beyond a predefined deterministic project due date δ_n. This also means that the planned starting time s_n in the baseline schedule is not allowed to exceed δ_n. In addition, w_n is usually set at a higher value compared to all other w_i in order to address the importance of a timely project completion.

Our objective is to obtain a project baseline schedule $(s_1, s_2, \ldots, s_n)$ that minimizes the expected positive deviations of both resource utilizations and activity starting times subject to the precedence constraints and the project due date constraint. Our problem can be formulated as follows:

$$\text{minimize} \sum_{k=1}^{K} \sum_{t=1}^{s_n} c_k E\left[(\boldsymbol{u}_{kt} - \bar{u}_k)^+\right] + \sum_{i=1}^{n} w_i E\left[(\boldsymbol{s}_i - s_i)^+\right] \tag{1}$$

subject to

$$s_j \geq s_i + \boldsymbol{d}_i \qquad \forall (i, j) \in A \tag{2}$$

$$\boldsymbol{s}_j = \max\left(s_j, \max_{i \in Pre_j} (\boldsymbol{s}_i + \boldsymbol{d}_i)\right) \quad \forall j \in N \tag{3}$$

$$\boldsymbol{u}_{kt} = \sum\nolimits_{i \in A_t} r_{ik} \quad k = 1, \ldots, K, \quad t = 1, \ldots, \delta_n \tag{4}$$

$$s_n \leq \delta_n, \tag{5}$$

where $E(\cdot)$ denotes the expectation operator and s_i are the decision variables. We call objective function (1) the *expected total schedule execution cost* (ETSEC). We call the first part of objective function (1) ($\sum_{k=1}^{K} \sum_{t=1}^{s_n} c_k E\left[(\boldsymbol{u}_{kt} - \bar{u}_k)^+\right]$) the *resource excessive usage cost* (REUC) and the second part ($\sum_{i=1}^{n} w_i E\left[(\boldsymbol{s}_i - s_i)^+\right]$) the *schedule instability cost* (SIC).

Constraints (2) enforce the precedence constraints. Since constraints (2) are stochastic constraints that do not define a deterministic feasible set, in stochastic programming such constraints are usually modeled as chance constraints: $Pr\left(s_i + \boldsymbol{d}_i \leq s_j\right) \geq \alpha, \forall (i, j) \in A$, where Pr is the probability measure and $\alpha \in (0, 1)$ (Lamas and Demeulemeester 2015; Liu 2009). This means that the probability that all the precedence constraints hold is larger than or equal to the confidence level α (α is usually chosen as 0.9, 0.95 or 0.99). However, even if we have a high confidence level, during project execution it is still possible that some precedence constraints are not respected subject to uncertainty (i.e., the realized schedule $(\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_n)$ is not exactly the same as

the planned baseline schedule $(s_1, s_2, \ldots, s_n)$). In this case, the railway scheduling reactive policy that is specified by constraints (3) is applied to repair the disrupted schedule. In constraints (3), Pre_j denotes the set of activity j's direct predecessors. Constraints (3) ensure that each activity will not start earlier than its planned starting time specified by the baseline schedule. Constraints (4) are used to calculate the realized resource usage. Constraint (5) is the project due date restriction.

Although we do not explicitly consider resource constraints, our objective function will achieve a similar effect to that of the resource constraints. Because in objective function (1) REUC tries to reduce the exceeded resource usages as much as possible, in addition to resulting in a smooth resource usage, this part of the objective function can also result in a relatively small resource usage (if the weight c_k is set fairly high).

In the literature on the resource leveling problem, the most often used objective function is the total weighted sum of the squared resource usage which means that both positive and negative deviations of the resource utilizations are minimized (Demeulemeester and Herroelen 2002). However, in the first part of our objective function (i.e., REUC), we consider only penalties for the excessive use of resources. A similar objective function has been discussed by Neumann and Zimmermann (1999, 2000). Besides, in project management, contractors usually reserve a certain amount of resources for the regular execution of a project. Excessive changes in the requirements of resources are very undesirable because this may result in frequently hiring and firing employees on a short-term basis, working overtime, and financial difficulties (Bandelloni et al. 1994). Note that other resource leveling objectives can be easily adapted and used in the first part of objective function (1).

According to the classification scheme of Herroelen et al. (2000), the robust resource leveling problem can be classified as $m, 1\,|cpm, \boldsymbol{d}_i, \delta_n|\, ETSTC$. The first field indicates that the number of renewable resource types is arbitrary. The second field specifies the existence of finish–start zero-lag precedence relationships, stochastic activity durations, and a deterministic project deadline. The last field gives the objective function, here the expected total schedule execution cost.

The deterministic resource leveling problem is NP-hard (Neumann et al. 2003). In our robust project scheduling problem, incorporating stochastic durations further complicates the problem. Therefore, the analytic evaluation of our objective function is very cumbersome. In this paper, we use simulation to evaluate the objective function.

2.2 Example

We provide an example to illustrate the problem under consideration. Figure 1 shows a project network. The activity

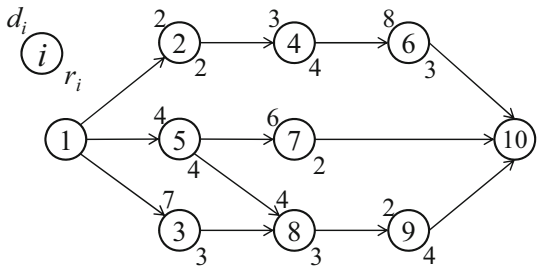


Fig. 1 Project network

number is shown inside the node. Activities 1 and 10 are two dummy activities. For each non-dummy activity, the duration is shown above the node and the single renewable resource requirement is shown below the node. In Fig. 1, the duration of each activity i is deterministic and d_i is therefore not written in bold face. The project due date is 18.

Figure 2 gives two baseline schedules for the example project network. Figure 2a is a leveled schedule (Demeulemeester and Herroelen 2002) obtained by the Burgess and Killebrew leveling procedure (B&K procedure) (Burgess and Killebrew 1962) with the objective of minimizing $\sum_k \sum_t c_k\,(u_{kt} - \bar{u}_k)^+$, where we set $\bar{u}_k$ at 6 in this example. Ever since the B&K procedure was proposed, many subsequent heuristics for the RLP are based on this procedure. The basic idea of the B&K procedure is to readjust the start time of each activity until the variability of the resource usage has been reduced to a near optimum. The order in which an activity is adjusted is indicated by a priority list. Therefore, given more than one priority list, the B&K procedure will iterate until every priority list is applied.

The only difference between Fig. 2a, b is that the start time of activity 6 in (b) is delayed by one time unit. In Fig. 2, the X-axis denotes time and the Y-axis represents resource utilization. The desired resource utilization is 6 which has been indicated by the dashed line in both figures. The bold line shows the resource utilization for each time period. The amount that the resource usage is exceeding 6 is indicated in dark color.

For the deterministic RLP with the objective of minimizing $\sum_k \sum_t c_k\,(u_{kt} - \bar{u}_k)^+$, it is obvious that baseline schedule 1 (with an objective function value equal to 6) is better than baseline schedule 2 (with an objective function value equal to 8).

Next, we will show that the leveled schedule for the deterministic RLP may not be good enough under activity duration uncertainty. Now, we assume that during execution the duration of activity 5 may be prolonged by 2 time units with a probability of 60%, which means that the duration of activity 5 is a stochastic variable with a discrete distribution $P\,(\boldsymbol{d}_5 = 6) = 0.6$ and $P\,(\boldsymbol{d}_5 = 4) = 0.4$. In this case, we need to consider not only resource leveling but also activ-

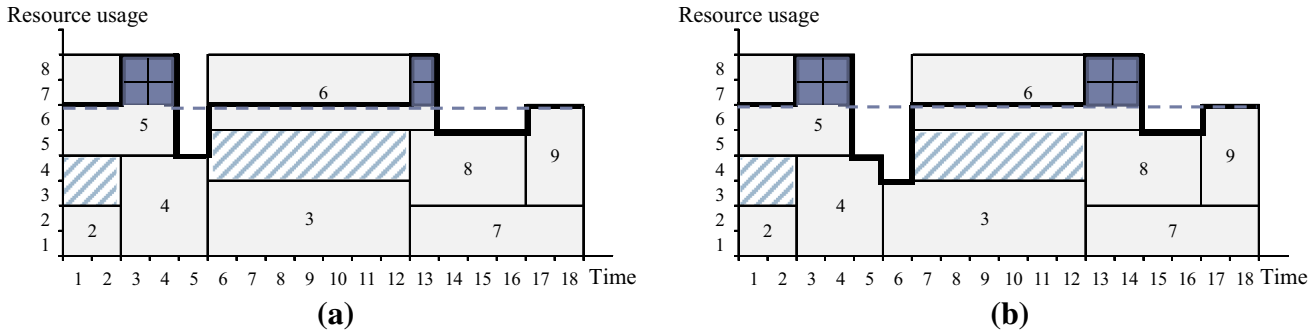


Fig. 2 Two baseline schedules. **a** Baseline schedule 1. **b** Baseline schedule 2

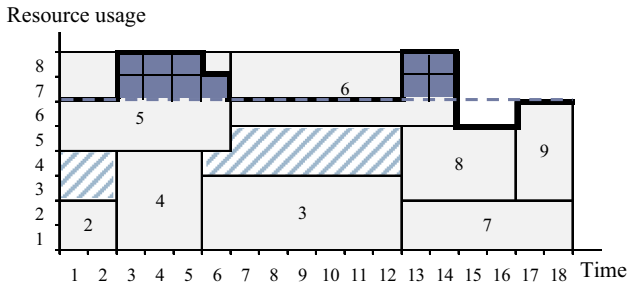


Fig. 3 The realized schedule ($\boldsymbol{d}_5 = 6$)

Table 1 Calculation of the expected total schedule execution cost

Interruption	Probability	Objective function	Objective function value	
			Schedule 1	Schedule 2
No	0.4	REUC	6	8
($\boldsymbol{d}_5 = 4$)		SIC	0	0
Yes	0.6	REUC	11	11
($\boldsymbol{d}_5 = 6$)		SIC	3×1	0
Expected total schedule execution cost			10.8	9.8

ity start time stability. Therefore, we use objective function (1) (the expected total schedule execution cost, see previous section) to compare the two schedules.

When activity 5 is subject to duration uncertainty, the expected total schedule execution cost for baseline schedules 1 and 2 are calculated as shown in Table 1. In this example, we assume $c_1 = 1$ and $w_5 = 3$. When $\boldsymbol{d}_5 = 4$, the realized schedules for both baseline schedules are identical to the corresponding baseline schedules (Fig. 2). When $\boldsymbol{d}_5 = 6$, the realized schedule for both baseline schedules becomes exactly the same which is shown in Fig. 3. The result indicates (see Table 1) that baseline schedule 2 has a lower expected total schedule execution cost (9.8) than baseline schedule 1 (10.8).

This example shows that for the robust RLP the leveled solution for the deterministic RLP may not be suitable anymore under activity duration uncertainty. Therefore, it is reasonable to develop effective solution procedures to obtain robust schedules with as low an expected total schedule execution cost as possible.

3 A genetic algorithm for the robust RLP

Genetic algorithm is a metaheuristic that is inspired by the idea of survival of the fittest in biological evolution. For an overview on the GAs, we refer to Goldberg et al. (1989). The reasons that we choose GA as the basis for our solution approach are as follows: (a) As an NP-hard problem, even the deterministic RLP has only been solved to optimality on instances with no more than 50 activities by Rieck et al. (2012) which seems to be the most powerful exact procedure thus far. Since tackling the RLP under uncertainty becomes much more difficult, this justifies developing heuristics to obtain near-optimal schedules for larger instances. As mentioned in the introduction, GA has already been adopted to solve the uncertain RLP by some authors. (b) Our model presented in Eqs. (1–5) is not a standard linear programming model or a stochastic programming model. Combining GA with simulation renders us the ability to efficiently deal with complex constraints and stochastic activity durations. (c) GA has been applied with success for solving both deterministic (Hartmann 2002; Valls et al. 2008; Debels and Vanhoucke 2007) and uncertain (Ke and Liu 2005; Ballestín 2007) project scheduling problems. Particularly, the decomposition-based GA presented by Debels and Vanhoucke (2007) obtained the currently best known results for the resource-constrained project scheduling problem (RCPSP).

In order to exploit the knowledge of the robust RLP, we introduce several changes in the GA. Our considerations behind these changes are as follows.

The robust RLP can be tackled in a two-stage way (as demonstrated by the example in Sect. 2.2) within a GA framework. (a) In the first stage, we concentrate on resource leveling and generate some schedules with varied resource leveling performance. This task is mainly done by the decod-

ing procedure. We also want to make the decoding procedure fast because this procedure is frequently called by the GA. Therefore, we design a special decoding procedure without relying on simulation to obtain a leveled schedule in as short a time as possible. (b) In the second stage, we choose and/or produce new schedules that aim for more stable starting times and/or for more leveled resource usage. This task is primarily accomplished by the crossover and mutation operators. The crossover operator we use is a hybrid one that combines a resource-based crossover and a two-point crossover. In doing so, we hope to generate new chromosomes with robust starting times and/or leveled resource usages.

In addition, the fitness value for each chromosome is computed by simulation. Also the railway scheduling reactive policy is embedded in the simulation process. Note that this is the only place that uses the time-consuming simulation in our GA. In this way, we hope to achieve a good trade-off between computation speed and quality.

With the above changes, we hope to obtain a satisfactory baseline schedule for the robust RLP effectively and efficiently.

3.1 Schedule representation and the GA framework

For the robust RLP, a solution is a schedule $S = (s_1, s_2, \ldots, s_n)$ that specifies the planned start times for each activity. Like most GAs for project scheduling problems, we do not operate directly on the schedule. Instead, we use the random key encoding in our GA (Debels and Vanhoucke 2007).

In our random key representation, a schedule is encoded as a chromosome that is given by a vector $\mathbf{X} \in \mathbb{R}^n$. Every element of $\mathbf{X}$ is called a gene. Each gene in the chromosome is a random value x_i which denotes the priority value of activity i, where $0 \leq x_i \leq \mathrm{M}, \mathrm{M} \gg n$. Unlike the classic random key representation, we do not consider precedence constraints in our random key representation and the precedence constraints will be considered in the decoding procedure. This mechanism ensures that any chromosome applied by initialization/crossover/mutation procedures will always correspond to a feasible solution. This characteristic is not possessed by the activity starting time-based encoding scheme that has been used in some resource leveling literature (Leu et al. 1999; Leu and Hung 2002; Masmoudi and Haït 2013).

Our GA framework is summarized as shown in Algorithm 1 (more details on our GA are given in the following subsections). Our GA starts with the generation of an initial population (see Sect. 3.3) whose size (i.e., the number of chromosomes in the population) is POP. The GA then transforms the initial chromosomes into schedules by means of the procedure **Decoding**() (see Sect. 3.2) and the corresponding fitness values are computed by means of the procedure **Evaluate**() (see Sect. 3.3). After that, the GA iterates until the maximum number of generations GEN is reached. In each iteration, parent chromosomes are first selected from the population by the procedure **Select_Parents**() (see Sect. 3.4). Then the crossover operator is applied to the resulting parent chromosomes in order to generate children chromosomes by the procedure **Crossover**() (see Sect. 3.5). Subsequently, the GA applies the procedure **Mutation**() (see Sect. 3.6) to the children chromosomes with the purpose of avoiding to fall into a local optimum. At the end of every iteration, we call the procedure **Select_Offspring**() (see Sect. 3.7) to replace the current population by selecting POP best individuals from the union set of parent and children chromosomes. After all of the iterations finish, the GA returns the best schedule found.

Algorithm 1. The GA framework

Input: the project network data
Output: a robust project schedule

Generate the initial population *chromosome*[POP] randomly;
Decoding(*chromosome*[POP], *schedule*[POP]);
fitness_chromosome[POP] ← Evaluate(*schedule*[POP]);
for *generation* = 1 to GEN
    Select_Parents(*chromosome*[POP], *parents*[POP]);
    Crossover(*parents*[POP], *children*[POP/2]);
    Mutation(*children*[POP/2]);
    Decoding(*children*[POP/2], *schedule*[POP/2]);
    fitness_children[POP/2] ← Evaluate(*schedule*[POP/2]);
    Select_Offspring(*parents*[*POP*] ∪ *children*[POP/2], *chromosome*[POP]);
    Update *fitness_chromosome*[POP];
end for
Decoding(*chromosome*[POP], *schedule*[POP]);
Return the best schedule;

It is worth noting that when solving the robust RLP, we do not consider SIC and REUC separately. Instead, in many parts of our GA (such as **Decoding**() and **Evaluate**() procedures), we deal with resource leveling and activity start time stability simultaneously.

3.2 The decoding procedure

In order to transform a chromosome into a schedule, a decoding procedure is needed. In the project scheduling literature, there are mainly two kinds of decoding procedures: the serial schedule generation scheme (SGS) and the parallel SGS. Both of the SGSs aim at generating a schedule with a makespan that is as short as possible. However, in our robust RLP, our objective is to obtain a leveled resource usage and stable activity starting times instead of a short makespan. Therefore, the serial or parallel SGS is no longer suitable for our problem and we design a specific decoding procedure for the robust RLP.

The basic idea of our decoding procedure is inspired by Neumann and Zimmermann (1999) and Ballestín et al.

(2007). The main differences between their procedure and ours are as follows. First of all, our procedure only focuses on the finish–start, zero-lag precedence relations, while their procedure considers more general relations (i.e., minimum and maximum time lags). Secondly, the tentative start time for a given activity is selected between its possible earliest start time and latest start time instead of relying on the so-called decision set as used by Neumann and Zimmermann (1999) and Ballestín et al. (2007). Finally, on the one hand, since our baseline schedule will be executed in an uncertain environment, it is desired that the decoding procedure could handle activity duration uncertainty to some extent. On the other hand, since the decoding procedure will be frequently invoked in the GA iterations, it is important for the decoding procedure to have a polynomial time complexity. Therefore, we use the mean duration for each activity as a predictive duration in our decoding procedure. In doing so, we avoid using simulation to handle the complex chance constraints mentioned in Sect. 2.1. This kind of treatment is quite common in practice when dealing with stochastic project scheduling problems (Ballestín 2007; Li et al. 2015) and also makes the decoding procedure much less time consuming. The pseudo-code of our decoding procedure is shown in Algorithm 2.

Algorithm 2. Pseudo-code of the decoding procedure

Input: a chromosome

Output: a schedule

$s_1 = 0; N' = \emptyset; CB = +\infty; \boldsymbol{u}_{kt}(N') = \vec{0};$
for $i = 1$ to n
  $es_i \leftarrow$ Earliest start time of activity i;
  $ls_i \leftarrow$ Latest start time of activity i given project due date δ_n;
  $s_i \leftarrow 0$;
end for
while $N \backslash N' \neq \emptyset$ do
  Select an eligible (precedence feasible) activity i from $N \backslash N'$ with minimum random key;
  for $j = es_i$ to ls_i
    $PM \leftarrow 0$;
    for $k = 1$ to K
      for $t = j$ to $j + d_i - 1$
        $PM \leftarrow PM + c_k \times (r_{ik} + \boldsymbol{u}_{kt}(N'))^2$;
      end for
    end for
    if $(PM < CB)$ $CB \leftarrow PM$; $s_i \leftarrow j$;
  end for
  $N' \leftarrow N' \cup \{i\}$;
  $CB \leftarrow +\infty$;
  // Update $\boldsymbol{u}_{kt}(N')$ according to N';
  for $k = 1$ to K
    for $t = 1$ to δ_n
      $\boldsymbol{u}_{kt}(N') \leftarrow \boldsymbol{u}_{kt}(N' \backslash \{i\})$
      if $t \in [s_i, s_i + d_i - 1]$ $\boldsymbol{u}_{kt}(N') \leftarrow \boldsymbol{u}_{kt}(N') + r_{ik}$;
    end for
  end for
  // Update es_j and ls_j;
  for each $j \in N \backslash N'$
    $es_j \leftarrow \max(es_j, \max_{i \in \{N' \cap Pred_j\}}(s_i + lpd_{ij}))$;
    $ls_j \leftarrow \min(ls_j, \min_{i \in \{N' \cap Succ_j\}}(s_i - lpd_{ji}))$;
  end for
end while

In Algorithm 2, N' denotes the set of scheduled activities and $u_{kt}(N')$ denotes the resource usage for resource type k during time period t given the scheduled activities in N'. d_i is the mean duration of activity i. Note that decision makers can also let d_i take value that are on some other representative estimators in terms of the project environment (e.g., 80% percentile, 50% percentile, and so on). At the beginning of the decoding procedure, the start time of each activity i is set at zero and its earliest start time es_i and latest start time ls_i is calculated according to the critical path method (CPM) (Kelley and Walker 1959; Demeulemeester and Herroelen 2002, Chapter 4.1.1). Then the algorithm repeats until all activities are scheduled. In each iteration, we select an eligible (precedence feasible) activity i with minimum random key value and choose the start time of activity i s_i between es_i and ls_i, such that the performance measure PM is minimized when activity i is scheduled to start at s_i. The performance measure PM is used to evaluate whether a resource profile is level or not and is defined as the total weighted sum of the squared resource usage: $PM = \sum_{k=1}^{K} \sum_{t=s_i}^{s_i+d_i-1} c_k \left(r_{ik} + u_{kt}(N')\right)^2$. We do not use our objective function (1) as the performance measure, because at the early stages of the iteration, the number of scheduled activities is so limited that the objective function (1) will always equal zero. However, the total weighted sum of the squared resource usage can avoid such a problem. Note that there may exist more than one s_i that leads to the same minimum PM value. In this case, we always choose the smallest s_i and this principle has been proven to be better than choosing the largest s_i based on preliminary experiments results. CB equals the current best performance measure value. After the start time of activity i is allocated, we need to update the current resource profile $u_{kt}(N')$ based on the scheduled activities in N'.

After we determine a new start time s_i for activity i, the es_j and ls_j for the unscheduled activities j need to be updated according to the scheduled activities and are calculated as follows:

$$es_j = \max\left(es_j, \max_{i \in \{N' \cap Pred_j\}} \left(s_i + lpd_{ij}\right)\right) \tag{2}$$

$$ls_j = \min\left(ls_j, \min_{i \in \{N' \cap Succ_j\}} \left(s_i + lpd_{ij}\right)\right). \tag{3}$$

Here, $Pred_j$ denotes the set of activities that are direct and indirect predecessors to activity j, and $Succ_j$ denotes the set of activities that are direct and indirect successors to activity

$j.lpd_{ij}$ is the longest path distance between activities i and j. By doing so, the precedence constraints are satisfied.

3.3 Initial population and fitness

The number of elements in the initial population is POP. We generate the initial population randomly. To compute the fitness value of a chromosome, the chromosome is first transformed into a schedule; then the fitness value is returned by the procedure **Evaluate**() which is based on simulation and the railway scheduling policy which means that the simulated activity start times are never sooner than the planned start times indicated by the baseline schedule (Herroelen and Leus 2004b; Van de Vonder et al. 2005; Demeulemeester and Herroelen 2011).

Algorithm 3 gives the pseudo-code of the **Evaluate**() procedure. REP denotes the number of replications of the simulation. Activity durations are sampled from specified probability distributions. Then we get simulated activity start times by applying a railway scheduling policy. After every activity is allocated a simulated start time, we get a simulated schedule. Based on the simulated schedule, we can calculate $\boldsymbol{u}_{kt}$ and then the fitness value that equals our objective function value ($\sum_{k=1}^{K}\sum_{t=1}^{s_n} c_k E\left[(\boldsymbol{u}_{kt}-\bar{u}_k)^+\right] + \sum_{i=1}^{n} w_i E\left[(\boldsymbol{s}_i - s_i)^+\right]$). The final fitness value is obtained by averaging the fitness values returned by each simulation run.

Algorithm 3. Pseudo-code of the **Evaluate**() procedure

Input: baseline schedule $S=(s_1,s_2,\ldots,s_n)$

Output: fitness value

```
fitness = 0;
for each resource type k, ū_k = ⌊(Σ_{i=1}^{n} d_i × r_ik)/δ_n⌋;
for rep = 1 to REP
    for each activity i, generate simulated duration d_i;
    // Get simulated activity start time by applying railway scheduling policy
    s_1 ← 0;
    for i = 1 to n
        for each direct successor j of activity i
            s_j ← max(s_i + d_i, s_j);
        end for
    end for
    // Get simulated resource usage
    for each k and t, u_kt = 0;
    for i = 1 to n
        for k = 1 to K
            for t = s_i to s_i + d_i − 1
                u_kt ← u_kt + r_ik;
            end for
        end for
    end for
    fitness ← fitness + Σ_{k=1}^{K} Σ_{t=1}^{s_n} c_k max(0, u_kt − ū_k) + Σ_{i=1}^{n} w_i(s_i − s_i);
end for
fitness ← fitness/REP;
```

3.4 Parent selection

The **Select_Parents()** procedure selects POP/2 pairs of parent chromosomes for the crossover operator. The top POP/2 best chromosomes are selected as the father chromosomes. The mother chromosomes are selected in the following way: two chromosomes are randomly selected from all of the chromosomes (except the father) and the one with a better fitness value is chosen as the mother chromosome. This process is repeated until POP/2 mother chromosomes are selected.

3.5 Crossover

POP/2 children chromosomes are generated by the **Crossover()** procedure. We hope to generate new chromosomes with such characteristics: some chromosomes have good resource leveling performance, some have robust starting times, and some have both. This would lead to a wide search of the solution space and result in some promising solutions. Therefore, the crossover operator we designed combines a resource-based crossover and a two-point crossover. POP/2 father chromosomes are divided into two sets on a fifty-fifty basis: set 1 (the top POP/4 father chromosomes) and set 2 (the remaining father chromosomes). The resource-based crossover operator is applied on set 1 and the two-point crossover operator is applied on set 2.

Our resource-based crossover is inspired by Debels and Vanhoucke (2007) and Valls et al. (2008). For the father chromosome and its corresponding schedule, we first select a sub-schedule length l which is randomly chosen between 1/4 $\times\delta$ and 3/4 $\times\delta$. Then we determine the crossover point t, such that $\sum_k \sum_{t=\{1,2,\ldots,\delta-l\}}^{t+l} c_k\,|u_{kt}-\bar{u}_k|^+$ is minimized. For activities with start times between t and $t+l$, the corresponding random key is added by a big enough number BIGN (equals 5000 in our GA) and the resulting random key is given to the child chromosome. The other gene positions of the child chromosome are derived from the corresponding positions of the mother chromosome.

In the two-point crossover, for the father chromosome, we randomly select two crossover points t_1 and t_2: the genes between t_1 and t_2 are given to the child chromosome. The other gene positions of the child chromosome are derived from the corresponding positions of the mother chromosome.

3.6 Mutation

For each newly generated child chromosome, each gene in the chromosome is changed to a new random priority value with a probability of *p_mutation*. The **Mutation()** procedure works as follows. Scan the chromosome from the first gene to

the last one. For each scanned gene, draw a random number from the interval [0, 1]. If this number is less than *p_mutation*, the corresponding gene value will be replaced by a newly generated random priority value.

3.7 Offspring selection

We call the procedure **Select_Offspring**() to select the POP best chromosomes from the union set of the parent and children chromosomes. The selected chromosomes will replace the current population and enter into the next iteration.

4 Experimental set-up

In our experiment, we primarily investigate the impact of different factors on the total schedule execution cost. Our GA has been coded in Visual C++ 2012. The GA has been tested on a large number of randomly generated problem instances. The computational experiment has been conducted on an Intel Core i5 2.40 GHz portable computer under Windows 7 64-bit version.

There has been no standard test set for the RLP under uncertainty. Therefore, we used *RanGen* (Demeulemeester et al. 2003) to construct 810 test instances using the parameter settings in Table 2. The order strength (OS) describes the network density and is computed as the number of precedence relations divided by the theoretical maximum number of precedence relations in the network. The larger the OS value, the higher is the network density. Herroelen and De Reyck (1999) demonstrated that OS is better than other commonly used measures [for example, network complexity, which is adopted in PSPLIB (Kolisch and Sprecher 1997)] when describing the network topology. The resource factor (RF) reflects the average number of resource types used by an activity. The resource constrainedness (RC) defines the average portion of the resource availability that is used by an activity. For each instance, the number of resource types is fixed at 4.

Specifying 3 settings for the number of activities, 3 settings for OS, 3 settings for RF, and 3 settings for RC, we generated 10 problem instances for each of the $3 \times 3 \times 3 \times 3$ parameter combinations, resulting in 810 instances in total.

Table 2 Parameter settings for the dataset

Parameter	Value
Number of activities (n)	30; 60; 90
Order strength (OS)	0.3; 0.5; 0.7
Resource factor (RF)	0.5; 0.75; 1
Resource constrainedness (RC)	0.3; 0.5; 0.7
Number of resource types	4

Table 3 Average results for different numbers of simulation replications

Number of replications	CPU time (in seconds)	Total schedule execution cost	REUC	SIC
50	0.74	4479.13	1183.98	3295.15
100	1.46	4482.18	1178.33	3303.85
1000	14.04	4492.64	1181.31	3311.33

We have fine-tuned the parameters of our GA and decided to choose the following settings for our GA based on the results of our preliminary experiments. The maximum number of schedules is set equal to 1000 (with GEN = 100, POP = 10). The mutation probability *p_mutation* is set equal to 0.05.

We have carried out the following preliminary experiment to determine the value of REP which is the number of replications of the simulation used by the procedure **Evaluate**() when evaluating the objective function. We have run our GA (GEN = 100, POP = 10 and *p_mutation* = 0.05) with REP being 50, 100, and 1000. In this preliminary experiment, we only used 10 % of the problem instances selected from dataset $n = 30$ and these problem instances are selected based on the following order: 1, 11, 21 …

Table 3 shows the average values for the CPU time, the total schedule execution cost, REUC, and SIC. As shown in this table, when REP = 50, it already provides a reasonable trade-off between the computational time and the estimation of the expected objective value. Therefore, we set REP = 50 in our GA.

The factors investigated in the computational experiments include: the marginal cost of the resource usage deviation (c_k), the marginal cost of the activity start time deviation (w_i), the activity duration variability (DUR_VAR), the due date (DUEDATE), the number of activities (n), the order strength (OS), the resource factor (RF), and the resource constrainedness (RC). The levels of these factors are summarized in Table 4.

Table 4 Factor levels for the experiment

Factor	Level
c_k	L (Low); N (Normal)
w_i	L (Low); M (Medium); H (High)
DUR_VAR	L (Low); M (Medium); H (High)
DUEDATE	L (Low); H (High)
n	30; 60; 90
OS	0.3; 0.5; 0.7
RF	0.5; 0.75; 1
RC	0.3; 0.5; 0.7

For c_k, its low level is drawn from a uniform distribution U[1, 5] (with average value 3) and its normal level is drawn from a uniform distribution U[1, 11] (with average value 6). For w_i, its low level is drawn from a discrete distribution with $P(w_i = 2 \times q) = (44 - 8q)\,\%, q \in \{1, 2, \ldots, 5\}$, its medium level is drawn from $P(w_i = 2 \times q) = (21 - 2q)\,\%, q \in \{1, 2, \ldots, 10\}$, and its high level is drawn from $P(w_i = 2 \times q) = (84 - 8q)\,\%, q \in \{6, 7, \ldots, 10\}$. The average values ($w_{avg}$) for these distributions are 4.2, 7.7, and 14.4. These distributions result in a lower probability for high weights. The weight of the dummy end activity is set at $w_n = \lfloor 10 \times w_{avg} \rfloor$ and represents the marginal cost of not finishing the project within the due date.

The realized activity durations $\mathbf{d}_i$ are drawn from a right-skewed beta distribution with parameters 2 and 5 with mean duration $E(\boldsymbol{d}_i) = d_i$ (d_i is the deterministic duration given in the dataset). We distinguish the activity duration variability by restricting the range of $\mathbf{d}_i$. For the low duration variability, the minimum duration $\mathbf{d}_{min} = 0.75 \times E(\boldsymbol{d}_i)$ and the maximum duration $\mathbf{d}_{max} = 1.625 \times E(\boldsymbol{d}_i)$. For the medium duration variability, the minimum duration $\mathbf{d}_{min} = 0.5 \times E(\boldsymbol{d}_i)$ and the maximum duration $\mathbf{d}_{max} = 2.225 \times E(\boldsymbol{d}_i)$. For the high duration variability, the minimum duration $\mathbf{d}_{min} = 0.25 \times E(\boldsymbol{d}_i)$ and the maximum duration $\mathbf{d}_{max} = 2.875 \times E(\boldsymbol{d}_i)$. These parameter settings have been widely used in the robust project scheduling literature (Van de Vonder et al. 2007a, b; Vonder et al. 2008; Deblaere et al. 2011).

For the due date, the low level means that the due date equals 1.2 times the critical path length. The high level means that the due date equals 1.4 times the critical path length.

We use a factorial experiment to investigate the impact of the above-mentioned factors on the total schedule execution cost. In this experiment, we focus on the main effects of the factors by examining two factors every time. For each investigated factor combination, 10 instances are solved using our GA. This results in a total of $2 \times 3 \times 3 \times 2 \times 3 \times 3 \times 3 \times 3 \times 10 = 29160$ executions of the GA.

5 Experimental results

Before we present our main results in Sect. 5.2, we first show in Sect. 5.1 comparison results between other often used resource leveling heuristics and our GA.

5.1 Comparison with other heuristics

5.1.1 Comparison with an existing GA

As mentioned in Sect. 1, many authors use GA to solve the resource leveling problem under uncertainty (Leu et al. 1999; Leu and Hung 2002; Masmoudi and Haït 2013; Zahraie and Tavakolan 2009; Ashuri and Tavakolan 2012). Among these papers, Leu and Hung (2002) is the only one that models the uncertain activity durations as stochastic variables. Therefore, in this subsection we compare our GA with the GA (LHGA) that was developed by Leu and Hung (2002). Unlike our GA that considers both resource leveling and activity start time stability, LHGA is specifically devised for resource leveling. In Leu and Hung (2002), they try to obtain a baseline schedule by minimizing the averaged resource leveling index (RLI) under stochastic activity duration. A project network with seven real activities is used to validate LHGA. In this project, each activity needs two types of resources ($K = 2$) with the same weight ($c_1 = c_2 = 1$). The duration of each activity is assumed to follow a triangular distribution. The project due date $\delta_n = 11$.

Table 5 Unit averaged resource leveling indices with project due date = 11

LHGA	1.17
Our GA	1.45

We run our GA on the same project data and use the same stop criterion (POP = 50, GEN = 100) as LHGA. The number of simulation replications is also the same (REP = 50). In order to obtain a relatively fair comparison, we omit the SIC part of our objective function (1) by setting $w_i = 0$ in the **Evaluate**() procedure. This means that we put all emphasis on resource leveling. After our GA terminates, we further calculate the value of the resource leveling index for the final output solution using simulation.

Table 5 presents the results of the comparison in terms of the unit averaged resource leveling index ($\frac{RLI}{\delta_n \cdot K}$). The results indicate that even though our GA is not solely designed for resource leveling, it is already effective enough compared with a special purpose GA. Our GA is slightly less competitive than LHGA. This would be explained as follows. Although we set the SIC part to be 0, we use the railway scheduling policy in the simulation when evaluating the objective function. Our GA may miss some promising solutions when the railway scheduling policy prohibits some activities from starting earlier.

5.1.2 Comparison with the B&K procedure

In this subsection, we compare our GA with a typical heuristic procedure (the B&K procedure in this experiment) on the basis of minimizing the expected total schedule execution cost. The purpose is to show that our GA is better when solving the robust RLP compared with the B&K procedure which is originally devised for solving the deterministic RLP.

We first use the B&K procedure to solve the robust RLP in the following way. In order to have a fair comparison with

Table 6 Comparison results between the B&K procedure and the GA for the robust RLP

Dataset	Total schedule execution cost		Improvement (%)
	B&K procedure	GA	
$n = 30$	5649.87	5018.54	11.17
$n = 60$	11,049.62	9375.34	15.15
$n = 90$	16,666.95	13,569.32	18.59
All instances	11,122.14	9321.07	16.19

GA, the stopping criterion for the B&K procedure we used is that 1000 schedules (which is the same as in our GA) have been generated. Specifically, we first generate a random priority list and the B&K procedure will iterate until a local optimum is reached. If the number of evaluated schedules is less than 1000, a new priority list will be generated randomly. This process will repeat until 1000 schedules have been generated. We solve the deterministic RLP with the objective of minimizing the total weighted sum of the squared resource usage ($\min \sum_k \sum_t u_{kt}^2$). For each instance, we can find a best schedule which is then evaluated by calling the procedure **Evaluate**() under all levels for c_k, w_i, DUR_VAR, and DUEDATE.

Our GA is then used to solve the same dataset. The comparison results are presented in Table 6. The column labeled with 'Total schedule execution cost' shows the expected total schedule execution cost of the best schedules obtained by the B&K procedure and GA, respectively. The column labeled with 'Improvement' indicates the percentage that the GA outperforms the B&K procedure in terms of expected total schedule execution cost. This is not surprising as our GA is specially designed for the robust RLP: our GA obviously outperforms the B&K procedure when solving the robust RLP.

Since we do not find an objective function that is similar to ours in the literature, we cannot provide a fair comparison with existing algorithms. Therefore, we have decided to focus on analyzing the impacts of different factors on the performance of our GA (see next section), hoping that these results can be used as a benchmark for future similar studies.

5.2 Main results

In this section, the impact of each factor on the total schedule execution cost is illustrated graphically for dataset $n = 30$ (Fig. 4). We will not give the similar figures for dataset $n = 60$ and 90 in order to avoid repetition. Instead, the results for dataset $n = 60$ and 90 will be shown in matrices (see below). For any examined two factors, each sub-figure in Fig. 4 plots the total schedule execution cost against one factor for all levels of the other factor.

In the following, we first take Fig. 4a as an example to explain how our results are presented. Afterwards, the patterns of the impacts for different dataset ($n = 30$, 60 and 90) will be summarized in matrices (see Sects. 5.2.1 and 5.2.2).

Figure 4a shows the total schedule execution cost, the REUC and the SIC against the marginal cost of the resource usage deviation c_k for all levels of the marginal cost of the activity start time deviation w_i. Figure 4a indicates that for a given w_i, a higher c_k results in a higher total schedule execution cost. This is also the case for the REUC, while the impact of c_k on the SIC is weak. Similarly, for a given c_k, a higher w_i also leads to a higher total schedule execution cost. The same pattern appears on the SIC, while the impact of w_i on the REUC is weak (the three lines are close to each other).

Overall, Fig. 4 reveals that c_k has the largest impact on the total schedule execution cost compared with other factors. Both RF and RC have the weakest impact on the SIC. The same conclusions could be drawn regarding the datasets $n = 60$ and $n = 90$.

5.2.1 The results for dataset $n = 30$

We summarize our main results for dataset $n = 30$ in a matrix as shown in Fig. 5. Each cell in the matrix represents the impact of a column element given a row element. Each cell consist of three marks: the mark in the top left corner indicates the impact on the total schedule execution cost, the mark in the top right corner indicates the impact on the REUC, and the mark in the lower right corner indicates the impact on the SIC. We take the cell in row 4 and column 3 for example. The mark '−' means that for a given level of the activity duration variability (DUR_VAR), the due date (DUEDATE) has a *weak* impact on the total schedule execution cost. The mark '↑' means that for a given level of the activity duration variability, the due date has a *positive* impact on the REUC. The mark '↓' means that for a given level of the activity duration variability, the due date has a *negative* impact on the SIC.

In addition, for the cell in row 1 and column 3, the mark in the top left corner is '− ↑.' This means that the due date has a *weak* impact on the total schedule execution cost when c_k is low and this impact becomes a *positive* one as c_k increases.

Based on Fig. 5, we can easily find that there mainly exist five patterns of impact which have been highlighted in different colors.

For factors c_k, RF, and RC, they have consistent patterns of impact. A higher c_k always means a higher REUC. And higher RF and RC mean that more resources are involved in the resource leveling process. Therefore, given any other factor, a higher c_k, RF, or RC results in a higher total schedule execution cost and REUC. However, the impacts of the above three factors on SIC are weak. For c_k, this is because c_k is

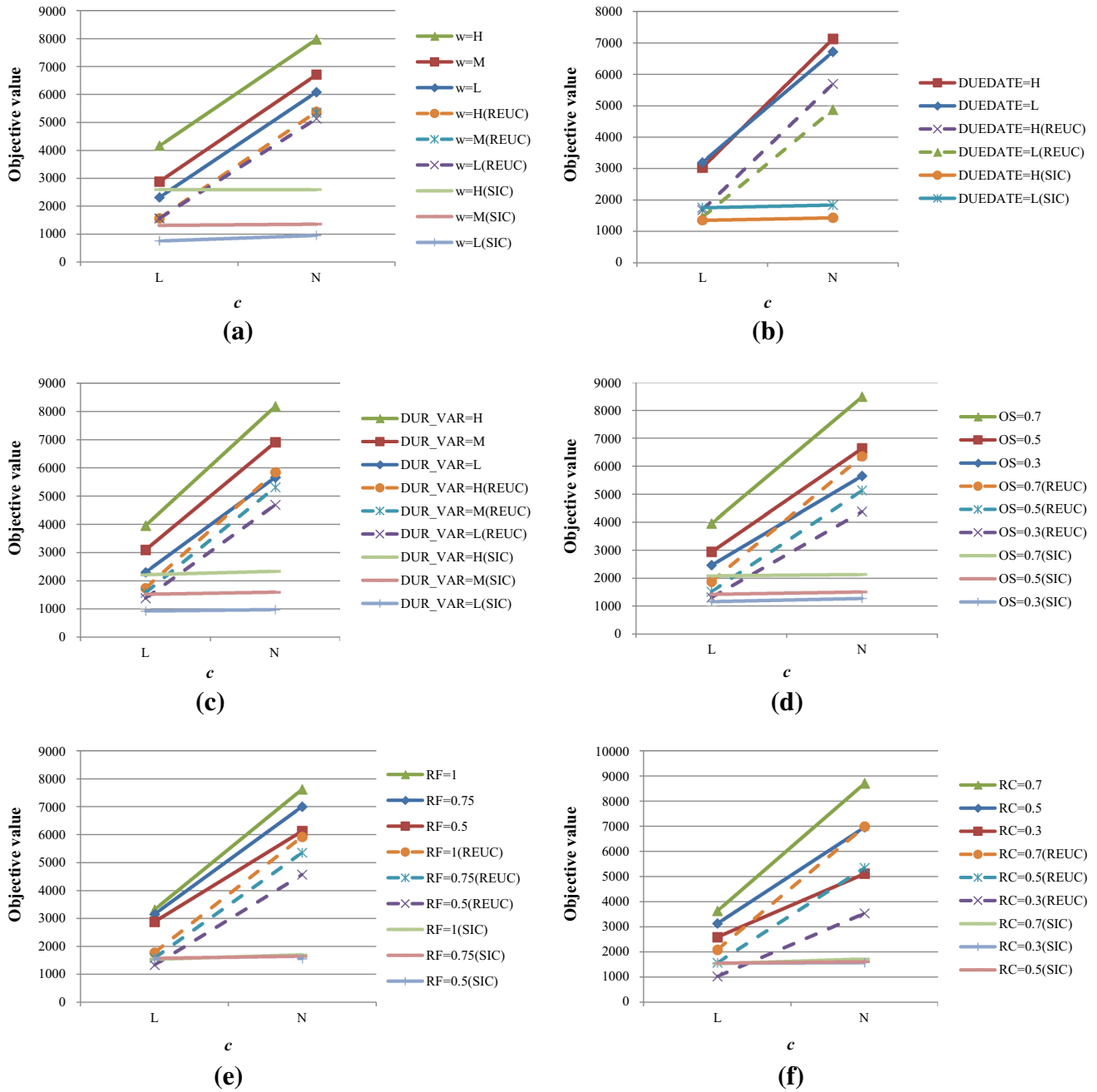


Fig. 4 Impacts of the factors ($n = 30$)

not included in the SIC part of the objective function. For RF and RC, our robust RLP does not explicitly consider resource constraints and this leads to a weak impact on the SIC.

Note that for the low level of c_k, the impact pattern of RF is slightly different (see Fig. 4e). When RF changes from 0.75 to 1, the corresponding changes in the value of the total schedule execution cost, REUC, and SIC are small. This means that for the circumstance that c_k is low and RF is high, the impact of RF is weak.

For w_i, it has a positive impact on the total schedule execution cost and SIC. And its impact on REUC is weak. This is obvious because w_i is the marginal cost of the activity start time deviation and mainly affects activity start time deviation related costs.

For factors DUR_VAR and OS, higher level of both factors result in a higher total schedule execution cost, REUC, and SIC. We may conclude that the larger the activity duration variability and the more activity precedence relations, the more costs will be incurred when we execute a project schedule.

Last, for the factor DUEDATE, it has two kinds of impact patterns that are caused by the differences in extent and direc-

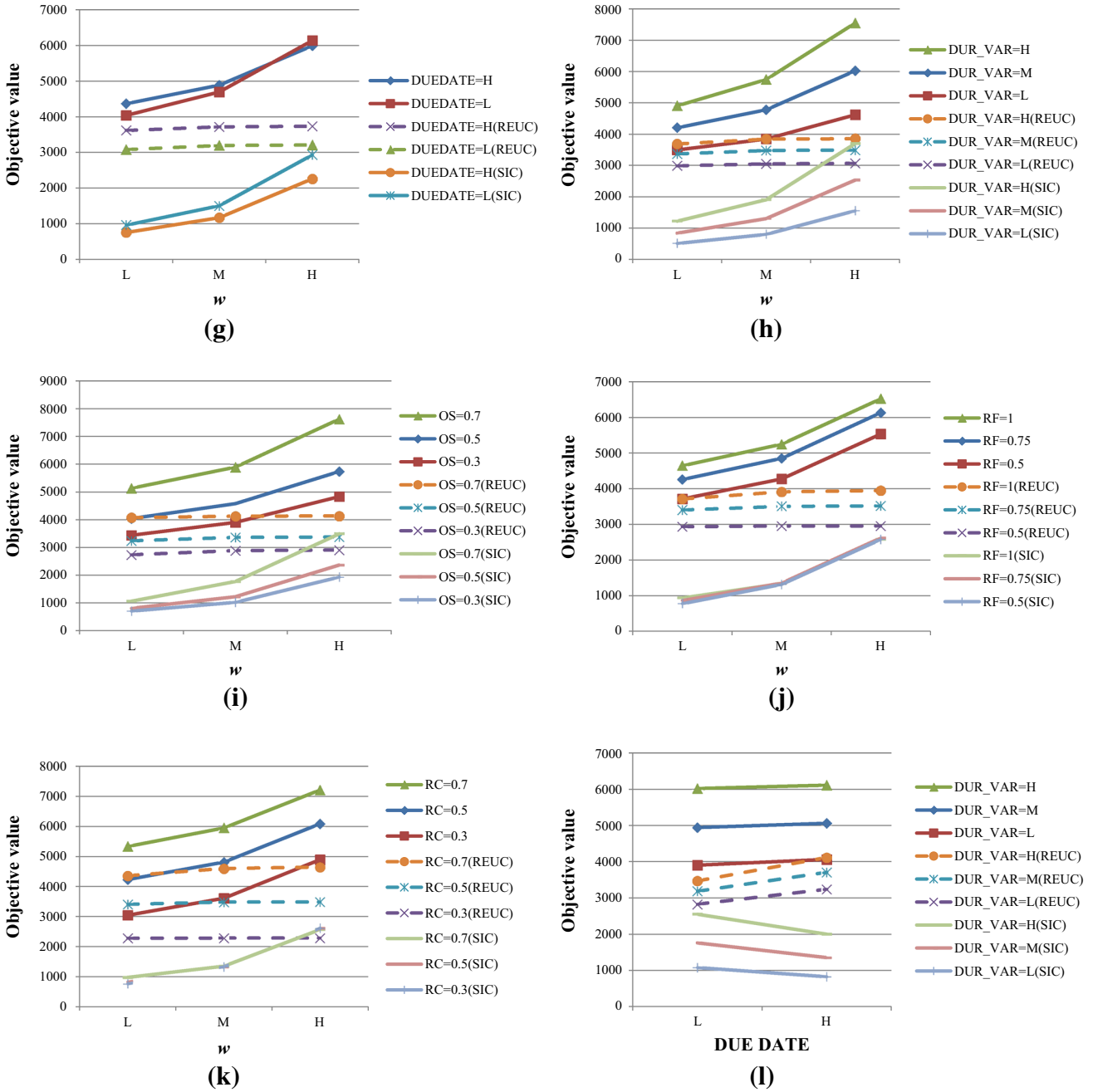


Fig. 4 continued

tion of the impacts of DUEDATE on REUC and SIC. A higher due date leads to a higher REUC which is surprising. This may be caused by the trade-off between the REUC and SIC when optimizing the total schedule execution cost. On the other hand, a higher due date results in a lower SIC because in this case more time buffers are inserted into the schedule, strengthening its ability to absorb more uncertainties. The combined results are that the impact of the due date on the total schedule execution cost becomes weak when considering DUR_VAR, OS, RF, or RC. However, when c_k is high or w_i is low, the due date has a positive impact on the total schedule execution cost. We may conclude that the impact of the due date will not be weak any more under the condition that the resource excessive usage unit cost is high enough (i.e., the resource excessive usage unit cost is very important) or the marginal cost of the activity start time deviation is low enough (i.e., the marginal cost of the activity start time deviation is not important).

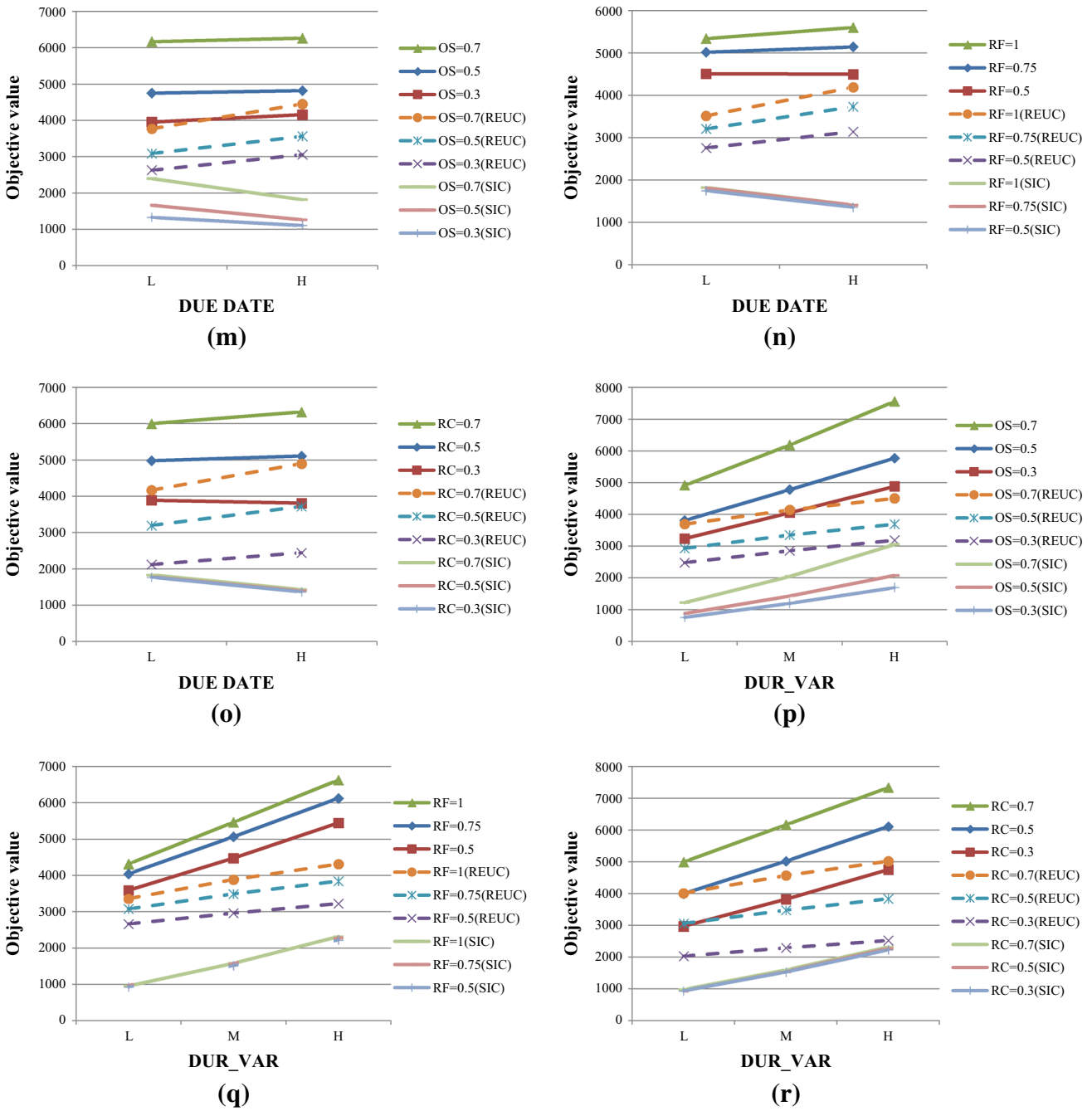


Fig. 4 continued

5.2.2 The results for dataset $n = 60$ and $n = 90$

Compared with dataset $n = 30$, it is obvious that an increasing number of activities results in a higher total schedule execution cost for datasets $n = 60$ and $n = 90$. The patterns of impact for dataset $n = 60$ (Fig. 6) and $n = 90$ (Fig. 7) are similar.

Compared with Fig. 5, we have marked the differences in Figs. 6 and 7 with a symbol '◥'. As shown in Figs. 6 and 7, the differences between dataset $n = 30$ and $n = 60/90$ are small and the changes only occur in SIC. We only discuss the changes. For a given RF or RC, the impact pattern of c_k varies between weak impact and positive impact as the levels of RF or RC change. For a given c_k, w_i, OS, and RC/RF, the impact of RF and RC on SIC has the same pattern (except the impact of RF for a given OS on dataset $n = 90$, which is the only difference between dataset $n = 60$ and dataset $n = 90$ and the corresponding symbol '◥' has been put in a different

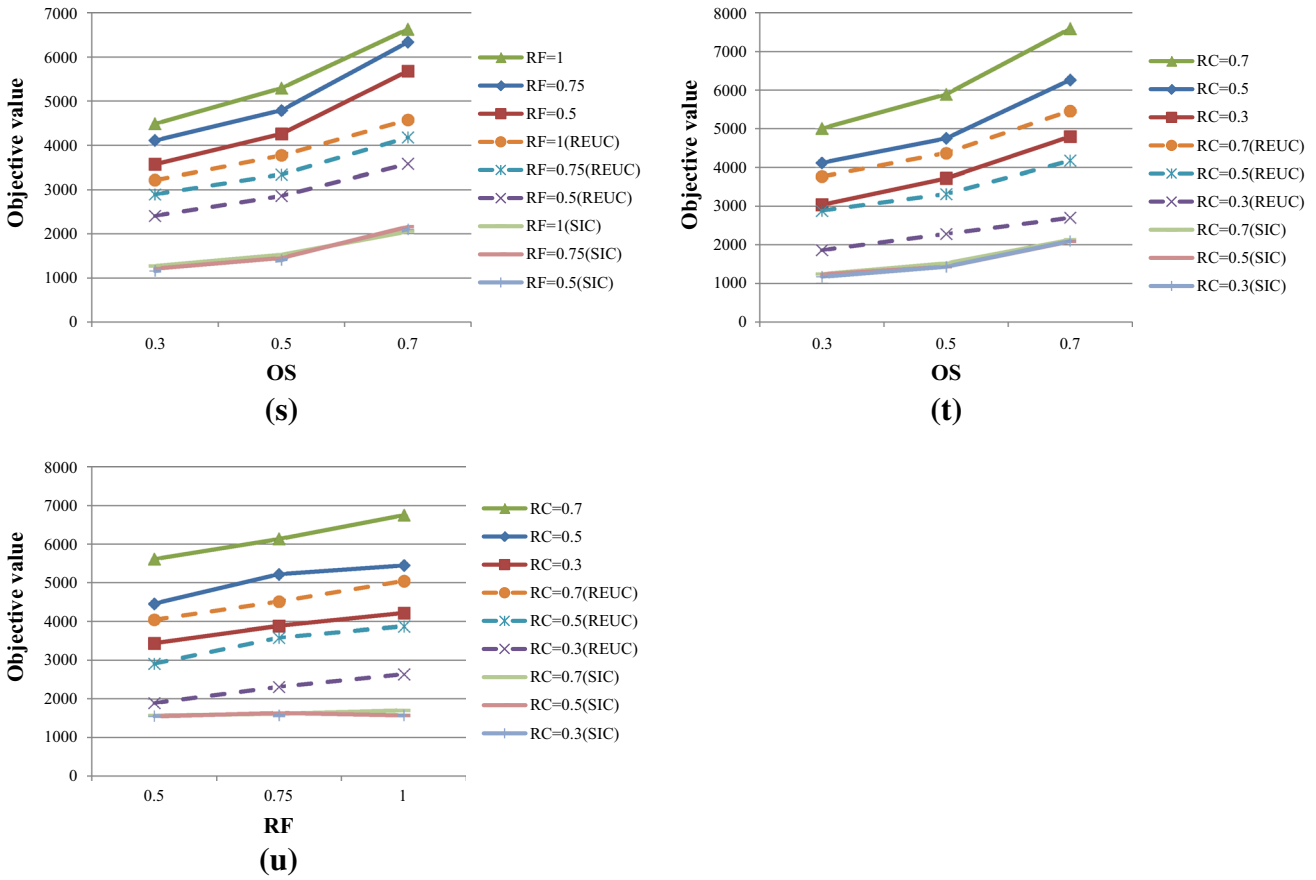
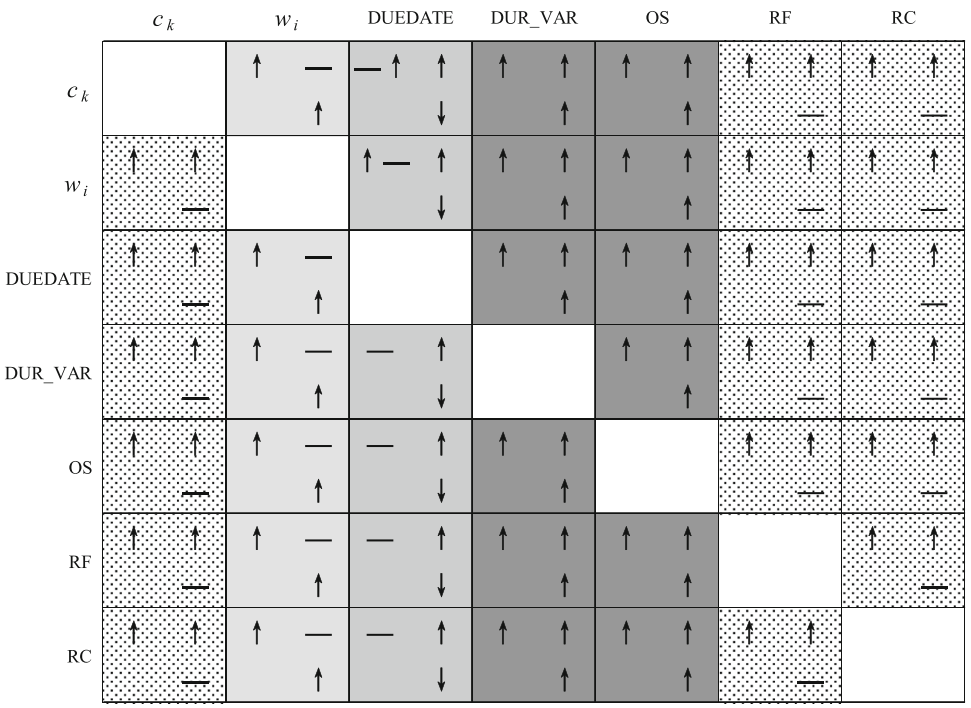


Fig. 4 continued

Fig. 5 Main results illustrated in a matrix ($n = 30$)



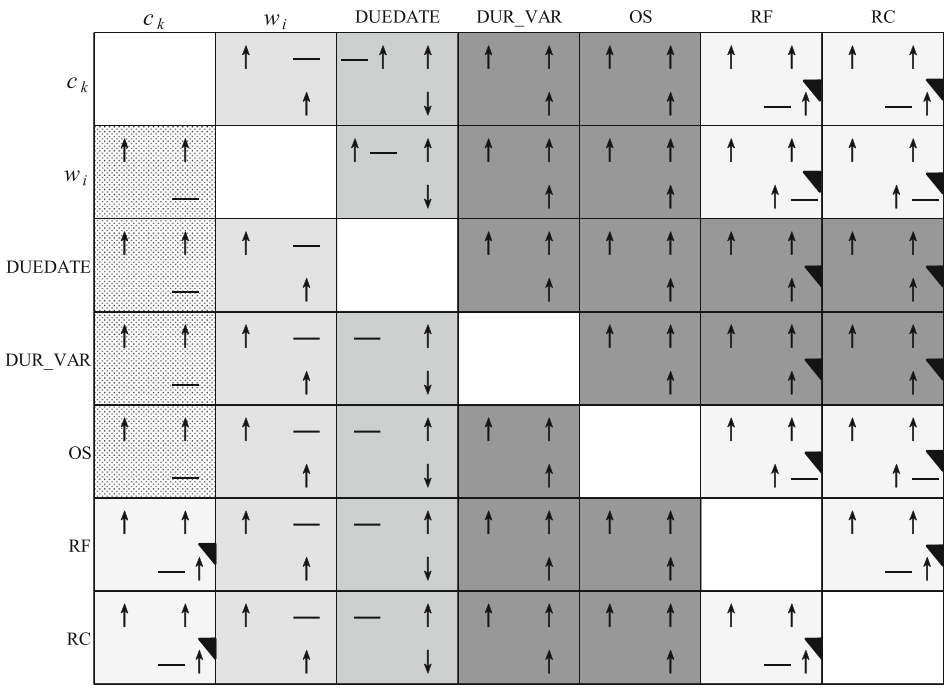


Fig. 6 Main results illustrated in a matrix ($n = 60$)

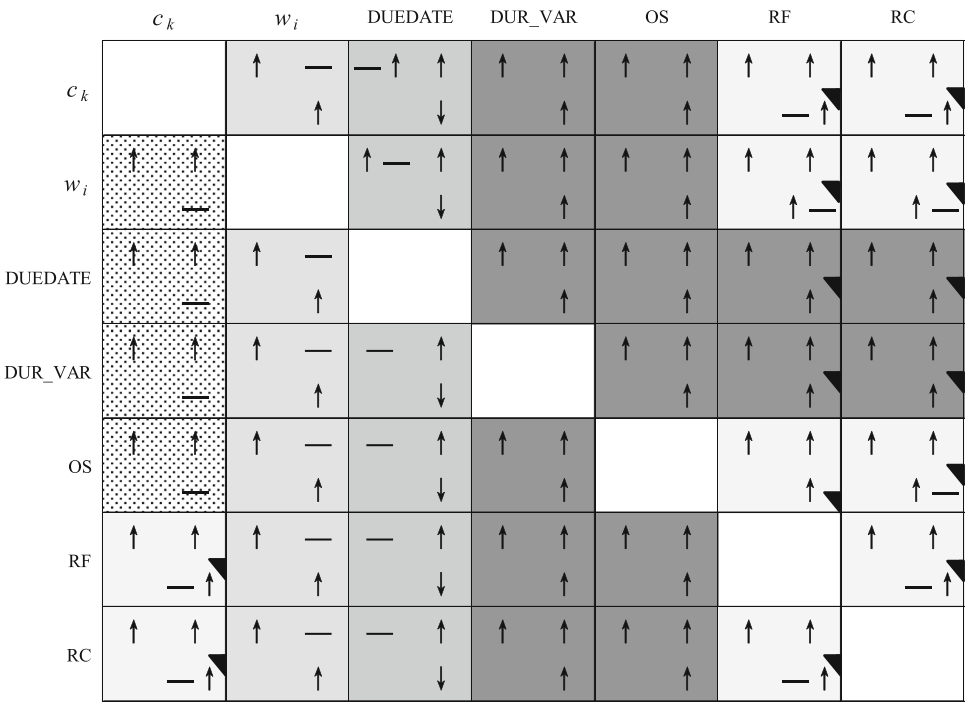


Fig. 7 Main results illustrated in a matrix ($n = 90$)

position compared with others in order to distinguish it). For a given due date or duration variability, both RF and RC have a positive impact on SIC. However, the extents of the above-mentioned positive impacts on SIC are still very small compared with the other factors.

As mentioned before, for dataset $n = 90$, the impact of RF on SIC is positive regardless of the level of OS, while for dataset $n = 60$, this impact depends on the level of OS.

On the whole, we observe that the impact patterns stay stable for most of the factors except RF and RC with the increase of the number of activities. The impact patterns of RF and RC are related with the scale of the project. When the number of activities is relatively big, the impacts of RF and RC on SIC show a positive trend. Unsurprisingly, most factors have a positive impact. However, since we combine leveled resource usage and stable activity starting times in the

Table 7 Average CPU time (in seconds)

Dataset	Avg. CPU time
$n = 30$	0.74
$n = 60$	1.17
$n = 90$	1.68

robust resource leveling problem, some unexpected phenomena are observed: (a) the factor due date actually has a weak impact on the total schedule execution cost in most cases, (b) RF and RC do not show an obvious impact on SIC until the number of activities becomes big, and (c) in the meantime, the impact of c_k also changes from weak to positive with the increase in the scale of the project.

5.2.3 Computation times

Table 7 shows the average CPU time needed to solve a problem instance with different numbers of activities. We do not give the detailed CPU time for other factors, because our results reveal that the number of activities has the only significant impact on the computation times. We find that due to the fact that the maximum number of schedules generated is only 1000, the average CPU time for each dataset is still quite acceptable even though the time-consuming simulation is embedded in the GA.

6 Conclusions and future research

In this paper, we proposed a genetic algorithm for the robust resource leveling problem where one tries to obtain a baseline schedule such that the expected total schedule execution cost is minimized during project execution. The total schedule execution cost consists of the cost related to the resource usage exceeding the desired resource level as well as the cost associated with positive deviations from the planned activity starting times. Our GA differs from a typical GA for extensions of the RCPSP in several aspects, such as a special decoding procedure for obtaining leveled schedules, a hybrid crossover operator that combines resource-based crossovers and two-point crossovers and an evaluation based on simulation for the objective function.

We performed extensive computational experiments on a large number of randomly generated test instances. We summarized the impact patterns of factors into five categories. Experimental results revealed that a higher level of a factor usually means a higher expected total schedule execution cost. Among the factors, c_k has the largest impact on the total schedule execution cost. However, both RF and RC have the weakest impact on the schedule instability cost in our robust resource leveling environment.

The study of proactive scheduling policies (Deblaere et al. 2011) for the stochastic RLP is a topic for future research. At the scheduling decision point during project execution, one has to decide which activities to execute such that the expected deviation of the resource usage is minimized while meeting the project due date as much as possible. The development of reactive scheduling procedures for the robust RLP is also an interesting research issue.

Acknowledgments The authors thank the reviewers for providing valuable suggestions that have improved the quality of this paper. The research of Hongbo Li is supported by the Research Center for Operations Management of the KU Leuven, the National Natural Science Foundation of China under Grant No. 71271019, the China Postdoctoral Science Foundation under Grant No. 2015M571542, the Humanities and Social Sciences Foundation of the Ministry of Education of China under grant 15YJCZH077 and a scholarship from the China Scholarship Council.

References

Ahuja, H. N. (1976). *Construction performance control by networks*. New York: Wiley.

Ashuri, B., & Tavakolan, M. (2012). Fuzzy enabled hybrid genetic algorithm-particle swarm optimization approach to solve TCRO problems in construction project planning. *Journal of Construction Engineering and Management*, *138*(9), 1065–1074.

Ballestín, F. (2007). When it is worthwhile to work with the stochastic RCPSP? *Journal of Scheduling*, *10*(3), 153–166.

Ballestín, F., Schwindt, C., & Zimmermann, J. (2007). Resource leveling in make-to-order production: Modeling and heuristic solution method. *International Journal of Operations Research*, *4*(1), 50–62.

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

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

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

Debels, D., & Vanhoucke, M. (2007). A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem. *Operations Research*, *55*(3), 457–469.

Deblaere, F., Demeulemeester, E., & Herroelen, W. (2011). Proactive policies for the stochastic resource-constrained project scheduling problem. *European Journal of Operational Research*, *214*(2), 308–316.

Demeulemeester, E. L., & Herroelen, W. (2002). *Project scheduling: A research handbook*. Boston: Kluwer Academic Pub.

Demeulemeester, E. L., & Herroelen, W. (2011). *Robust project scheduling (Vol. 3, No. 3–4)*. Delft: Now Publishers Inc.

Demeulemeester, E., Vanhoucke, M., & Herroelen, W. (2003). RanGen: A random network generator for activity-on-the-node networks. *Journal of Scheduling*, *6*(1), 17–38.

Easa, S. M. (1989). Resource leveling in construction by optimization. *Journal of Construction Engineering and Management*, *115*(2), 302–316.

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

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

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

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

Harris, R. B. (1990). Packing method for resource leveling (PACK). *Journal of Construction Engineering and Management*, *116*(2), 331–350.

Hartmann, S. (2002). A self-adapting genetic algorithm for project scheduling under resource constraints. *Naval Research Logistics*, *49*(5), 433–448.

Herroelen, W., & De Reyck, B. (1999). Phase transitions in project scheduling. *Journal of the Operational Research Society*, *50*(2), 148–156.

Herroelen, W., De Reyck, B., & Demeulemeester, E. (2000). On the paper "Resource-constrained project scheduling: Notation, classification, models and methods" by Brucker et al. *European Journal of Operational Research*, *128*(3), 221–230.

Herroelen, W., & Leus, R. (2004a). Robust and reactive project scheduling: A review and classification of procedures. *International Journal of Production Research*, *42*(8), 1599–1620.

Herroelen, W., & Leus, R. (2004b). The construction of stable project baseline schedules. *European Journal of Operational Research*, *156*(3), 550–565.

Herroelen, W., & Leus, R. (2005). Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, *165*(2), 289–306.

Ke, H., & Liu, B. (2005). Project scheduling problem with stochastic activity duration times. *Applied Mathematics and Computation*, *168*(1), 342–353.

Kelley, J. E., & Walker, M. R. (1959). Critical-path planning and scheduling. In *Proceedings of the Eastern Joint Computer Conference* (pp. 160–173).

Kolisch, R., & Sprecher, A. (1997). PSPLIB: A project scheduling problem library. *European Journal of Operational Research*, *96*(1), 205–216.

Kreter, S., Rieck, J., & Zimmermann, J. (2014). The total adjustment cost problem: Applications, models, and solution algorithms. *Journal of Scheduling*, *17*(2), 145–160.

Lamas, P., & Demeulemeester, E. (2015). A purely proactive scheduling procedure for the resource-constrained project scheduling problem with stochastic activity durations. *Journal of Scheduling*, 1–20. doi:10.1007/s10951-015-0423-3.

Lambrechts, O., Demeulemeester, E., & Herroelen, W. (2008). Proactive and reactive strategies for resource-constrained project scheduling with uncertain resource availabilities. *Journal of Scheduling*, *11*(2), 121–136.

Leu, S. S., Chen, A. T., & Yang, C. H. (1999). A fuzzy optimal model for construction resource leveling scheduling. *Canadian Journal of Civil Engineering*, *26*(6), 673–684.

Leu, S. S., Yang, C. H., & Huang, J. C. (2000). Resource leveling in construction by genetic algorithm-based optimization and its decision support system application. *Automation in Construction*, *10*(1), 27–41.

Leu, S. S., & Hung, T. H. (2002). An optimal construction resource leveling scheduling simulation model. *Canadian Journal of Civil Engineering*, *29*(2), 267–275.

Li, H., Xu, Z., & Demeulemeester, E. (2015). Scheduling policies for the stochastic resource leveling problem. *Journal of Construction Engineering and Management*, *141*(2), 04014072.

Liu, B. (2009). Stochastic programming. *Theory and practice of uncertain programming* (2nd ed., pp. 25–56). Berlin, Heidelberg: Springer.

Masmoudi, M., & Haït, A. (2013). Project scheduling under uncertainty using fuzzy modelling and solving techniques. *Engineering Applications of Artificial Intelligence*, *26*(1), 135–149.

Neumann, K., & Zimmermann, J. (1999). Resource levelling for projects with schedule-dependent time windows. *European Journal of Operational Research*, *117*(3), 591–605.

Neumann, K., & Zimmermann, J. (2000). Procedures for resource leveling and net present value problems in project scheduling with general temporal and resource constraints. *European Journal of Operational Research*, *127*(2), 425–443.

Neumann, K., Schwindt, C., & Zimmermann, J. (2003). *Project scheduling with time windows and scarce resources*. Berlin: Springer.

Ponz-Tienda, J. L., Yepes, V., Pellicer, E., & Moreno-Flores, J. (2013). The Resource Leveling Problem with multiple resources using an adaptive genetic algorithm. *Automation in Construction*, *29*, 161–172.

Ranjbar, M. (2013). A path-relinking metaheuristic for the resource levelling problem. *Journal of the Operational Research Society*, *64*(7), 1071–1078.

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

Tang, L., Zhao, Y., & Liu, J. (2014). An improved differential evolution algorithm for practical dynamic scheduling in steelmaking-continuous casting production. *IEEE Transactions on Evolutionary Computation*, *18*(2), 209–225.

Valls, V., Ballestin, F., & Quintanilla, S. (2008). A hybrid genetic algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research*, *185*(2), 495–508.

Van de Vonder, S., Demeulemeester, E., Herroelen, W., & Leus, R. (2005). The use of buffers in project management: The trade-off between stability and makespan. *International Journal of Production Economics*, *97*(2), 227–240.

Van de Vonder, S., Ballestin, F., Demeulemeester, E., & Herroelen, W. (2007a). Heuristic procedures for reactive project scheduling. *Computers & Industrial Engineering*, *52*(1), 11–28.

Van de Vonder, S., Demeulemeester, E., & Herroelen, W. (2007b). A classification of predictive-reactive project scheduling procedures. *Journal of Scheduling*, *10*(3), 195–207.

Van de Vonder, S., Demeulemeester, E., & Herroelen, W. (2008). Proactive heuristic procedures for robust project scheduling: An experimental analysis. *European Journal of Operational Research*, *189*(3), 723–733.

Wiest, J., & Levy, F. (1977). *A management guide to PERT/CPM*. Englewood Cliffs: Prentice Hall.

Wullink, G., Gademann, A. J. R. M., Hans, E. W., & Van Harten, A. (2004). Scenario-based approach for flexible resource loading under uncertainty. *International Journal of Production Research*, *42*(24), 5079–5098.

Wullink, G. (2005). Resource loading under uncertainty. PhD thesis, University of Twente.

Zahraie, B., & Tavakolan, M. (2009). Stochastic time-cost-resource utilization optimization using nondominated sorting genetic algorithm and discrete fuzzy sets. *Journal of Construction Engineering and Management*, *135*(11), 1162–1171.