Journal of Business Economics (2020) 90:679–711
https://doi.org/10.1007/s11573-020-00993-z

ORIGINAL PAPER

Simultaneous structuring and scheduling of multiple projects with flexible project structures

Luise-Sophie Hoffmann[1] · Carolin Kellenbrink[1] · Stefan Helber[1]

Published online: 17 June 2020

Abstract
We study the problem to simultaneously decide on the structures and the schedules for an entire portfolio of flexible projects. The projects are flexible as alternative technologies and procedures can be used to achieve the respective project task. The choice between different technologies and procedures affects the activities to be implemented and thus the precedence relations, i.e., the structure of the project. The different projects have given due dates with specific delay payments and compete for scarce resources. In this situation, project structure decisions and scheduling decisions are highly intertwined and have to be made simultaneously in order to achieve the assumed objective of minimizing the delay payments for the entire project portfolio. The problem is formally stated and solved via novel and problem-specific genetic algorithms. The performance of the new algorithms is evaluated with respect to speed and accuracy in a systematic and comprehensive numerical study.

Keywords Multi-project scheduling · Flexible projects · Genetic algorithms · RCPSP · RCMPSP

JEL Classification C61 · M11

1 Introduction

In this paper, we study the problem to structure and schedule multiple projects. The different projects have to be considered simultaneously as they compete for scarce resources such as human workers or technical equipment with exogenously limited capacities. We assume that for each of the projects, a due date is given and that an agreed-upon penalty payment per time unit of delay has to be paid if the due date

✉ Stefan Helber
stefan.helber@prod.uni-hannover.de

[1] Institute of Production Management, Leibniz University Hannover, Königsworther Platz 1, 30167 Hannover, Germany

should not be met. However, as opposed to previous works on multi-project scheduling, we consider the case of *flexible projects*.

In a flexible project as introduced in Kellenbrink and Helber (2015), there is some room for decision-making with respect to the structure of the project. The set of activities is *not* assumed to be exogenously given as alternative activities or entire sets of activities can serve the same purpose. In such a situation, the project structure decision determines the activities that are actually implemented. Deciding about the structure of the project also affects precedence relations between activities that have to be respected by any project schedule. Flexible project scheduling problems arise when different technologies and procedures can be used to achieve a certain objective. The choice between those different technologies, for example using either human labor or, to some degree, automated equipment, can require or eliminate the need to perform some activities.

Note that the problem of structuring and scheduling a flexible project is substantially more general and flexible than the multi-mode project scheduling problem (MRCPSP), cf., e.g., Talbot (1982) or the review in Węglarz et al. (2011). In the MRCPSP a set of modes with different durations and capacity consumptions is given for each activity. One out of those modes has to be selected, while both the set of implemented activities and the set of precedence relations are not affected by this mode-selection.

When multiple flexible projects have to be scheduled simultaneously, the project structure decision for one such project can, via the common resource constraints, affect the schedules and penalty payments of other projects. Thus, it can be numerically challenging to make optimal decisions on project structures and schedules for a portfolio of projects.

Since genetic algorithms have frequently been proposed to determine high-quality solutions for project scheduling problems quickly, we have also developed genetic algorithms to make the two intertwined decisions on the project structures and schedules. This requires new and problem-specific representations and decoding procedures like those presented in this paper.

The remainder of this paper is structured as follows: In Sect. 2, we formally describe the problem via a discrete-time mixed-integer programming model. Section 3 describes how individual solutions of the problem can be represented and decoded into workable schedules such that the genetic algorithm developed in Sect. 4 can operate efficiently. The performance of the genetic algorithms is studied in Sect. 5 in a large-scale numerical study before conclusions and suggestions for further research are given in Sect. 6.

2 The resource-constrained multi-project scheduling problem with flexible project structures

2.1 Problem setting

It is quite common that a decision-making authority of an organization is confronted with the task to schedule multiple flexible projects that are executed at least partially in parallel on shared resources. Examples can be found in the construction industry where a firm can possess different types of equipment that can, at least to some extent, perform the same tasks. Assume that for a certain work at a certain construction site, both a lift and a crane could be used to transport construction material vertically to where it is needed. Both types of equipment require specific transportation, installation, and operation resources and procedures. Those activities become part of the respective project schedule, induce precedence relations, and lead to specific capacity requirements.

The turn-around processes of aircraft between landing and starting at commercial airports are another example of a portfolio of flexible projects. At many airports, an aircraft can be served either at a terminal gate or at an apron parking position. In the latter case, passenger buses are often used to transport passengers between the terminal and the aircraft. The choice between those two procedures affects the structure of the network of activities describing the turn-around process. Delays in processing the aircraft can lead to monetary fines. When congested airports face delays of arriving aircraft, they require the capability to quickly schedule multiple turn-around processes involving different resources such as service teams, passenger buses, aircraft tractors etc.

A further example stems from maintenance activities at wind turbines. After some time of usage, the blades of these turbines typically exhibit damages due to material fatigue, bird collisions etc. These damages are repaired during maintenance operations. The firms that perform those maintenance operations can use movable platforms that are installed at the respective wind turbine such that repair crews can access and repair the damaged part of the turbine blade. As an alternative, a worker can be let down on a rope from the wind turbine's gondola and perform certain tasks while hanging at this rope. The owner of the wind turbines is typically interested in a short down time of the turbine during maintenance. However, the firm that is sending repair crews to the different wind turbines may have a limited number of platforms so that the maintenance processes have to be coordinated. The maintenance process for each wind turbine can be interpreted as a flexible project and the service provider has to decide on the structures and schedules for an entire portfolio of those projects.

In a problem setting with multiple (flexible) projects competing for the same set of renewable resources such as personnel, machinery, or vehicles, it may be inevitable to occasionally violate some project due dates. It is not uncommon that some kind of penalty payment per time unit of delay has been agreed upon. As these penalties can be project-specific, they reflect the relative importance of the different projects' due dates. Note that this includes the case of a cost-weighted makespan minimization for a portfolio of projects, i.e., due dates of zero for all projects.

2.2 Literature review

The deterministic resource-constrained project scheduling problem (RCPSP) is a well-discussed topic. For the basic principles and approaches we refer to Demeulemeester and Herroelen (2002). Extensive literature reviews are given by Habibi et al.

(2018), Hartmann and Briskorn (2010), Kolisch and Padman (2001), Brucker et al. (1999), Herroelen et al. (1998) as well as Özdamar and Ulusoy (1995).

To the best of our knowledge, only Hauder et al. (2019) analyze the combination of project portfolios and flexible projects which we treat in this paper. In their work, the production of a steel lot is interpreted as a project with alternative routes. Each route contains a sequence of different activities that have to be implemented if the route is chosen. For each steel lot exactly one route has to be selected. Thereby, the duration of an activity may be affected. Since several different steel lots are planned simultaneously, a flexible multi-project scheduling problem with activity selection and time flexibility results.

The problem setting described in this paper combines the two areas of scheduling (single) flexible projects on the one hand and scheduling project portfolios on the other hand. Therefore, in the broad field of the project scheduling literature, our review focuses on those two aspects: Firstly the definition of flexibility for single projects and secondly the evaluation of schedules for entire portfolios of projects.

The consideration of projects with flexible project structures has attracted increasing attention in recent years. Tiwari et al. (2009) examine an extension of the multi-mode resource-constrained project scheduling problem (MRCPSP) with single rework activities that become necessary if a specific mode is chosen for one activity. Other publications focus on the implementation of alternative branches in the project network. Belhe and Kusiak (1995) consider “design activity networks”, Čapek et al. (2012) operate with alternative process plans. Kuster et al. (2009) and Kuster et al. (2010) address disruption management problems with alternative process implementation paths.

Tao and Dong (2017) present alternative activity chains that are substitutable. In this problem setting, logical depencencies between two alternative subgraphs are only possible if there is a precedence relation between them. Tao and Dong (2018) extend this approach by adding different execution modes for the activities. Tao et al. (2018) consider projects with hierarchical alternatives and stochastic activity durations. In this problem, a project contains choices on alternative methods, where each method can consist of different activities. One method can trigger another choice about further sub-methods.

Servranckx and Vanhoucke (2019a) consider flexible work packages modeled as alternative subgraphs. These alternatives can depend on each other. For example, one alternative work package can be nested in another alternative work package. Furthermore, the optional subgraphs can be linked by precedence relations. Servranckx and Vanhoucke (2019b) extend this problem by taking into account the uncertainty of activity duration as well as resource efficiency and resource breakdowns.

Birjandi and Mousavi (2019) as well as Birjandi et al. (2019) deal with the resource-constrained project scheduling problem with multiple routes, taking into account uncertain durations and uncertain non-renewable resources. A project contains so-called flexible sections with alternative routes. Each route can contain different activities and precedence relations. Logical dependencies between flexible sections are not considered.

Cajzek and Klanšek (2019) assume alternative production processes with different project structures. There is always exactly one decision concerning the project structure, i.e., after selecting one process, no further structure decisions are triggered. However, there are different execution modes for the activities of a process alternative.

To our best knowledge, there is no other work except Kellenbrink and Helber (2015) that considers logical and precedence relations separately. In that modeling approach, there can be a logical dependency between two activities that are not directly linked to each other by a precedence constraint. The relationship between profit and quality outcome of such a flexible project has been studied in Kellenbrink and Helber (2016).

According to Lova et al. (2000), 84% of companies dealing with project scheduling have a portfolio of different projects that need to be planned simultaneously. When solving such resource-constrained multi-project scheduling problems (RCMPSP), special aspects such as project-specific due dates may occur. As shown below, delay-related objectives can hence be used to evaluate schedules for portfolios of projects. Comprehensive overviews of literature in the field of the RCMPSP are included in the surveys from Hartmann and Briskorn (2010) and Özdamar and Ulusoy (1995).

Gonçalves et al. (2008) consider release and due dates. They present a new performance measure minimizing the weighted sum of tardiness, earliness and flow time of all projects.

Browning and Yassine (2016) focus on portfolios of product development projects which are cyclical. The authors compare the performance of different priority rules for two different objectives, the minimization of the average across the relative delays of all projects and the minimization of the overall relative delay of the portfolio.

Kurtulus (1985) discusses different priority rules for scheduling multiple projects with unequal delay penalties. The performance is evaluated by the total delay penalties of all projects. In addition, he presents different functions to define delay penalties for the problem instances.

Chiu and Tsai (2002) deal with the RCMPSP with discounted cash flows. The objective of this problem is to maximize the net present value of the project portfolio. The net present value includes the net cash flow for each activity, the project final receipt, delay penalties as well as early completion bonuses.

Our idea to jointly consider a portfolio of flexible projects of the type introduced in Kellenbrink and Helber (2015) has originally been presented in Hoffmann et al. (2017) and Hoffmann and Kellenbrink (2018). The contribution of this current paper is to present and evaluate the model formulation and novel algorithms to simultaneously schedule entire portfolios of flexible projects.

2.3 Notation, minimal example and explicit problem statement

In order to formally describe the problem setting of the resource-constrained multi-project scheduling problem with flexible project structures (RCMPSP-PS), we now use the two-project example in Fig. 1 to introduce our notion of a flexible project

Table 1 Notation

Indices and index sets	
$l \in \mathcal{L}$	Projects
$j \in \mathcal{J}_l$	Activities of project l
$i \in \mathcal{P}_{lj}$	Activities i that are immediate predecessors of activity j of project l
$j \in \mathcal{V}_l$	Mandatory activities of project l
$r \in \mathcal{R}$	Renewable resources $r = 1, \ldots, R$
$t, \tau \in \mathcal{T}$	Periods $t, \tau = 1, \ldots, T$
$e \in \mathcal{E}_l$	Choices $e = 1, \ldots, E_l$ in project l
$a(l, e)$	Triggering activity of choice e from project l
$j \in \mathcal{W}_{le}$	Activities j included in the decision set of choice e of project l
$i \in \mathcal{B}_{lj}$	Activities i caused by activity j of project l
Parameters	
d_{lj}	Duration of activity j of project l
k_{ljr}	Resource demand of activity j of project l on resource r
K_r	Capacity of resource r
EFT_{lj}	Earliest finish time of activity j of project l
LFT_{lj}	Latest finish time of activity j of project l
δ_l	Due date of project l
c_l	Tardiness cost for each delayed period of project l
Variables	
$x_{ljt} \in \{0, 1\}$	1, if activity j of project l is finished at the end of period t, 0 else
$v_l \geq 0$	Tardiness of project l

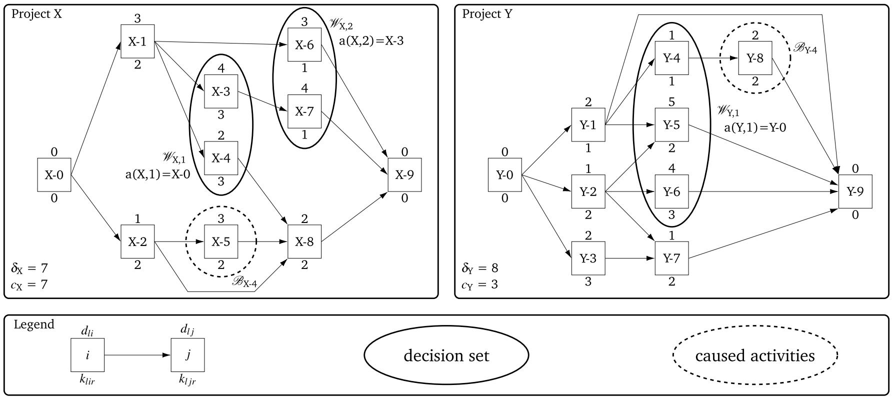


Fig. 1 Example of two different projects

formally. The required notation is summed up in Table 1, see also Kellenbrink and Helber (2015).

According to the legend in Fig. 1, the square nodes of the graph depict the activities $j \in \mathcal{J}_l$ of each project $l \in \mathcal{L}$ with their duration d_{lj} and their resource requirement k_{ljr} for renewable resources $r \in \mathcal{R}$. Without loss of generality and in order to ease the exposition of this example, we include in the name of each activity the name of the project to which the activity belongs. So in the left part of Fig. 1 related to project X, the (dummy) start activity is denoted as "X-0" (which is *not* to be read as "X minus 0"). In this example, we consider only one single resource. So activity X-1 has a duration of three time units and during its execution two units of the single renewable resource are required. The activities 0 and $J_l + 1$ denote dummy-activities with a duration of 0 periods and without any resource consumption. An arrow between activities represents an end-to-start precedence relation in project l between an activity j and its immediate predecessor $i \in \mathcal{P}_{lj}$.

Consider the network of project X in the upper left part of Fig. 1. The ovals represent the so-called decision sets $\mathcal{W}_{le}$ for structure choice $e \in \mathcal{E}_l$ of project l. Such a decision set $\mathcal{W}_{le}$ indicates which activities are included in the decision set of choice $e \in \mathcal{E}_l$ of project l. For example, in *structure decision* $e = 1$ of project X with decision set $\mathcal{W}_{X,1} = \{\text{X-3}, \text{X-4}\}$, either activity X-3 or X-4 has to be selected to be part of the project. The further information $a(X,1) = \text{X-0}$ indicates that this first decision of project X is *triggered* by this project's activity X-0. This particular activity X-0 happens to be a *mandatory activity*, i.e., X-0 $\in \mathcal{V}_X$. Being mandatory, it has to be implemented. For this reason, the structure decision $e = 1$ to either execute activity X-3 or activity X-4 is always triggered, i.e., it can be interpreted as a *mandatory decision*. Note that *any* mandatory activity could be used to trigger a mandatory structure decision, in the case of project X any activity in the set $\mathcal{V}_X = \{\text{X-0}, \text{X-1}, \text{X-2}, \text{X-8}, \text{X-9}\}$. However, we always trigger mandatory decisions via the mandatory (dummy) start activity. If in this first decision activity X-4 should be selected, then the *caused activity* X-5 $\in \mathcal{B}_{\text{X-4}}$ is included in the project network, together with the precedence relations leading to or originating from activity X-5.

The second structure decision $e = 2$ concerns the choice of one activity out of the set $\mathcal{W}_{X,2} = \{\text{X-6}, \text{X-7}\}$, i.e., between activities X-6 and X-7. Note that the second structure decision is a *conditional decision*. This choice is triggered by activity X-3 as we have $a(X,2) = \text{X-3}$. If in the first structure decision between activity X-3 and activity X-4 the triggering activity X-3 should *not* be selected, then neither activity X-6 nor activity X-7 is implemented.

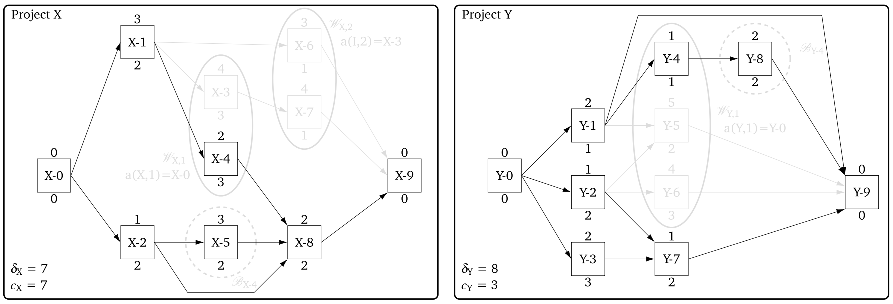


Fig. 2 Optimal project structures

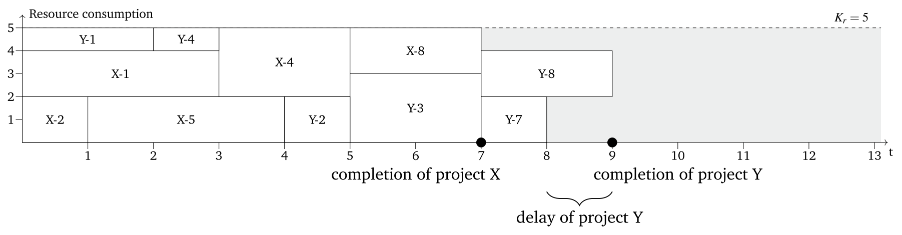


Fig. 3 Optimal schedule

According to Fig. 1, for project X we have the due date $\delta_X = 7$ and the tardiness cost per time unit $c_X = 7$. The respective values for project Y are $\delta_Y = 8$ and $c_Y = 3$. In this example, we assume a given constant capacity K_r of the single renewable resource of five capacity units. The project structures in Fig. 2 and the corresponding schedule in Fig. 3 lead to the minimal total delay cost of three monetary units, as the completion of project Y at the end of period 9 exceeds the due date $\delta_Y = 8$ by one time unit.

In this example, only one renewable resource with a time-constant capacity is considered to facilitate the description of the problem. In the general case, we consider a set of different renewable resources that are required for implementing activities.

Using those ideas we can now state the problem as follows:

- For an entire portfolio of projects, we have to schedule the respective activities.
- All projects have specific due dates and tardiness cost rates. The objective is to find a schedule which minimizes the overall tardiness cost of the entire portfolio of projects.
- The activities of the projects have a given duration and may require renewable resources of limited capacity. In order to be feasible, a project schedule has to respect the capacity constraints of the resources.
- Activities are either mandatory or optional. Mandatory activities have to be included in a schedule. Optional activities, however, are subject to one out of possibly multiple project structure decisions for the respective project.
- In a project structure decision, one out of a set of optional candidate activities has to be selected. Each project structure decision is triggered by a specific activity. If the triggering activity itself is mandatory, e.g., the dummy start activity, the triggered decision is also mandatory. Likewise, if the triggering activity itself is optional, the project structure decision itself is also conditional or optional.
- If an activity is selected in a decision, one or more further activities may be caused.
- The predefined precedence relations have to be respected if the activities between which they are established are included in the schedule, whether those activities have an optional or a mandatory character.

2.4 Model formulation

To formally define the resource-constrained multi-project scheduling problem with flexible project structures (RCMPSP-PS), we use the notation already introduced in the example and summarized in Table 1. The main decision variable is the binary variable x_{ljt} which assumes a value of 1 if activity j of project l is completed at the end of period t and 0 otherwise. The derived variable $v_l \geq 0$ reflects the tardiness of project l. Note that the calculation of earliest finish times EFT_{lj} and latest finish times LFT_{lj} for flexible projects differs from the calculation of time windows for projects with a fixed project structure. We only consider mandatory activities for our flexible projects and refer to Kellenbrink and Helber (2015) for details of the calculation.

Given this notation, the problem can be defined via the following objective function and constraints:

$$\min \sum_{l \in \mathcal{L}} v_l \cdot c_l \tag{1}$$

subject to

$$\sum_{t=EFT_{lj}}^{LFT_{lj}} x_{ljt} = 1 \qquad l \in \mathcal{L}; j \in \mathcal{V}_l \tag{2}$$

$$\sum_{j \in \mathcal{W}_{le}} \sum_{t=EFT_{lj}}^{LFT_{lj}} x_{ljt} = \sum_{t=EFT_{l,a(l,e)}}^{LFT_{l,a(l,e)}} x_{l,a(l,e),t} \qquad l \in \mathcal{L}; e \in \mathcal{E}_l \tag{3}$$

$$\sum_{t=EFT_{li}}^{LFT_{li}} x_{lit} = \sum_{t=EFT_{lj}}^{LFT_{lj}} x_{ljt} \qquad l \in \mathcal{L}; e \in \mathcal{E}_l; j \in \mathcal{W}_{le}; i \in \mathcal{B}_{lj} \tag{4}$$

$$\sum_{t=EFT_{li}}^{LFT_{li}} t \cdot x_{lit} \leq \sum_{t=EFT_{lj}}^{LFT_{lj}} (t - d_{lj}) \cdot x_{ljt} + |\mathcal{T}| \cdot \left(1 - \sum_{t=EFT_{lj}}^{LFT_{lj}} x_{ljt}\right) \qquad l \in \mathcal{L}; j \in \mathcal{J}_l; i \in \mathcal{P}_{lj} \tag{5}$$

$$\sum_{l \in \mathcal{L}} \sum_{j \in \mathcal{J}_l} k_{ljr} \sum_{\tau=t}^{t+d_{lj}-1} x_{lj\tau} \leq K_r \qquad r \in \mathcal{R}; t \in \mathcal{T} \tag{6}$$

$$v_l \geq \sum_{t=EFT_{l,J_l+1}}^{LFT_{l,J_l+1}} t \cdot x_{l,J_l+1,t} - \delta_l \qquad l \in \mathcal{L} \tag{7}$$

$$v_l \geq 0 \qquad l \in \mathcal{L} \tag{8}$$

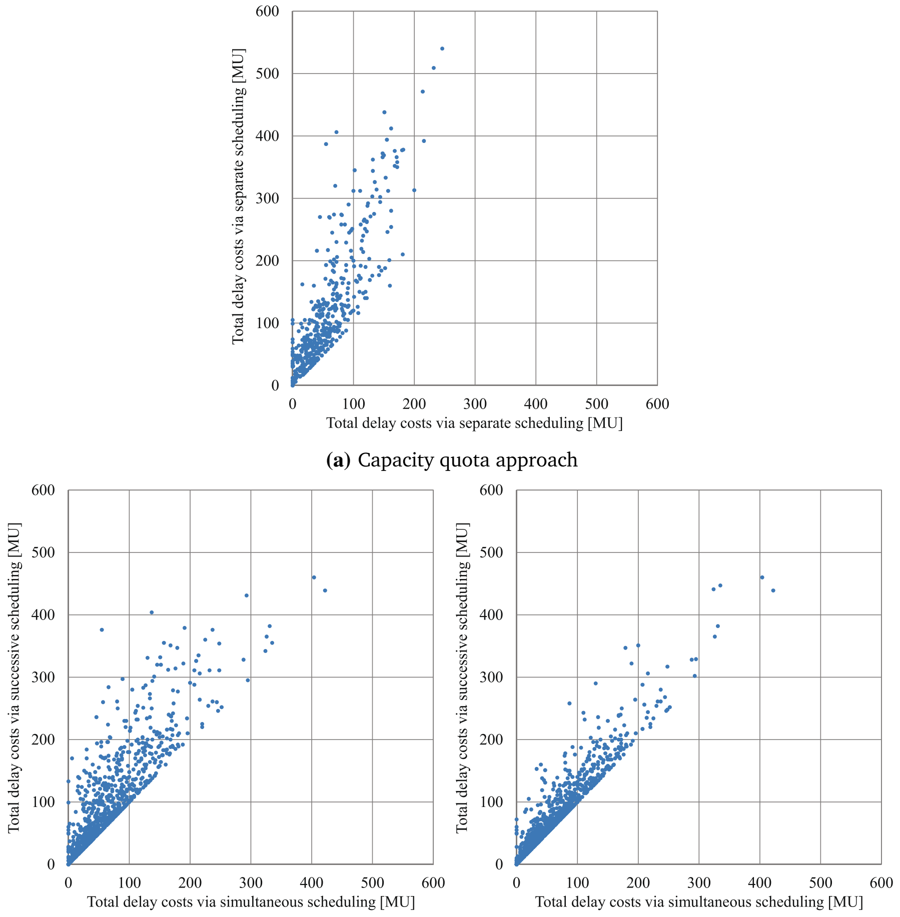
$$x_{ljt} \in \{0,1\} \qquad l \in \mathcal{L}; j \in \mathcal{J}_l; t \in \mathcal{T} \tag{9}$$

In the objective function (1), the total tardiness cost over the entire project portfolio is minimized. Equations (2) ensure that each mandatory activity of each project is terminated exactly once. Equations (3) serve to trigger choices between activities. It is possible that a choice e of a project l is *not* triggered if the triggering activity $j = a(l,e)$ itself is an optional activity that is not chosen to be implemented. The right-hand side of Eq. (3) assumes a value of 1 if and only if choice e of project l is triggered. In this case, exactly one activity $j \in \mathcal{W}_{le}$ out of the set of alternative activities of this choice e is scheduled. The caused activities $i \in \mathcal{B}_{lj}$ are implemented if and only if the triggering activity j of project l is chosen to be executed through Eq. (4). The precedence relations between those activities that are actually implemented are enforced via Constraints (5). The capacity restrictions for renewable resources are modeled via Constraints (6). Constraints (7) serve, together with the optimization direction of the objective function, to determine the project-specific delay v_l. According to Constraints (8) and (9), the tardiness cannot be negative and the discrete-time scheduling variables x_{ljt} are binary.

2.5 Economic benefit of simultaneous coordination of project portfolios

An organization confronted with the need to find project structures and schedules for an entire portfolio of flexible projects can use different coordination approaches. One option is to make all decisions simultaneously as suggested in this paper, thereby fully considering all their interdependencies. However, other coordination mechanisms such as decomposition into (or hierarchization between) different subproblems of the original planning problem, see Küpper (1997, p. 101), are also popular in practice and seem to suggest themselves. In order to show the benefit of our simultaneous approach compared to other coordination mechanisms in a clear and systematic manner, we now refer in an aggregated form to 864 different portfolio planning instances consisting of two projects each. Those instances are described in detail in Sect. 5.1 as they are also used to evaluate the numerical performance of our algorithms to be presented in the second half of this paper. For each of these project instances, we were able to solve the model presented in Sect. 2.4 to proven optimality using the Gurobi solver.

One very simple coordination approach is to split the capacity of each resource and to give all projects an equal capacity quota. Within this capacity quota, each project can then be scheduled separately again using the model from Sect. 2.4, but now solved for each single project. However, both in reality and in our test bed one can see situations in which a project needs for at least one resource-intensive activity temporarily *more* that an equal share of the capacities. Therefore, the capacity quota approach found feasible solutions for only 452 out of the 864 problem instances. Figure 4(a) presents a scatter plot of the numerical results for those feasible



(a) Capacity quota approach

(b) Random sequential approach

(c) Prioritized sequential approach

Fig. 4 Comparison of simultaneous optimization with other coordination approaches

instances. Each dot shows horizontally the total delay cost in monetary units (MU) of the optimal (simultaneous) solution and vertically the total delay cost of the plans achieved via capacity quotas. Each point above the 45° line indicates that operating with quotas is more costly than the simultaneous solution. The figure shows clearly that, even when the fixed capacity quota approach led to feasible schedules, the total delay costs were often substantially higher than those resulting from the simultaneous optimization.

Operating with fixed quotas imposes a lot of inflexibility and potentially leads to an inefficient usage of the scarce resources. An alternative coordination mechanism between the projects in a given portfolio is to schedule the projects sequentially. The project considered firstly is allowed to use the whole capacity of all resources. The actually required resources based on the resulting schedule are then "booked" and no longer available for those projects that are treated later in this sequential approach so that the next project to be scheduled can only use the (complete) *remaining* resources and so on. Figure 4(b) shows the results when the projects within a portfolio are considered in a random sequence. Now all instances can be solved. As the dots in Fig. 4(b) tend to be lower than in Fig. 4(a), this mechanism leads to slightly better solutions as resource capacities can be used more efficiently. However, it is shown clearly that our simultaneous approach still uses them much better.

A more advanced approach is to operate with a hierarchy or priority between the projects in a given portfolio. To study this coordination approach, we sorted in each of the 864 problem instances the projects l by decreasing cost rates c_l for project delays, interpreting high cost rates for project delays as indicators of a high project priority. The project with the highest cost rate was then scheduled first, again having access to all resource capacities which were then partially booked based on the resulting schedule. Then the project with the second-highest priority was treated in the same way, having access to the remaining capacities. Compared to Fig. 4(a) and (b), the results in Fig. 4(c) indicate a smaller cost disadvantage of this sequential approach with project priorities. However, as for the two other coordination mechanisms, we again see the clear superiority of our simultaneous approach. It can therefore be truly beneficial to solve the entire problem simultaneously as suggested by our model in Sect. 2.4 and not to decompose it by establishing fixed capacity quotas or by scheduling the projects within a portfolio sequentially.

2.6 Computational complexity and algorithmic approach

If only a single project is considered in which all activities are mandatory (so that the choice set is empty) and the due date of this project is set to 0, then the RCMPSP-PS is reduced to the standard resource-constrained project scheduling problem (RCPSP). According to Kolisch (1996), the RCPSP is an $\mathcal{NP}$-hard optimization problem as it constitutes a generalization of the static job shop scheduling problem which has been shown to be $\mathcal{NP}$-hard in Blazewicz et al. (1983). Since the RCMPSP-PS contains the RCPSP as a special case, the RCMPSP-PS is therefore also $\mathcal{NP}$-hard. We would therefore be surprised to discover an exact algorithm for the RCMPSP-PS with a computational effort that is a polynomial function of the problem size (in terms of activities, periods etc.). Given the $\mathcal{NP}$-hardness of many well-established variants of the RCPSP, many researchers have resorted to heuristic approaches. Out of these, genetic algorithms have often been shown to be extremely powerful so that even relatively large problem instances could be solved with a high solution quality and with a negligible computation time, cf., e.g., Hartmann and Kolisch (2000).

For this reason, we also propose to use a tailored genetic algorithm to solve the problem presented above. A genetic algorithm is a search algorithm that operates with a population of solutions to the underlying problem. This population is modified in an iterative way, resembling the evolution of living species. New and hopefully better solutions are created by combining elements of one particular solution with other elements of a second solution, just as living creatures inherit traits from both mother and father, creating new and potentially better adapted creatures.

In order for such a genetic algorithm to operate successfully, a suitable representation of solutions to the problem is required. A direct representation in terms of the variables x_{ljt} and v_l as introduced in this section is not suitable. A genetic algorithm uses operators to construct new feasible solutions by exchanging and combining elements of promising old solutions. When it is applied to a direct representation in terms of the variables x_{ljt}, it will almost inevitably fail to create new feasible solutions. In order to overcome this problem, established solution representations typically decide on the sequence or priority in which different activities are treated in a so-called schedule generation scheme (SGS) which then, finally, leads to a workable schedule.

In the context of our RCMPSP-PS, a solution representation must not only consider a sequence or set of priorities in which a schedule is constructed. In addition, it has to take care of the project structure decisions for the different projects as well.

In order to describe the algorithm to solve the RCMPSP-PS, in Sect. 3 we explain the representation of individual solutions and their decoding into a corresponding schedule, before we illustrate in Sect. 4 the operators of the genetic algorithm that modify the above-mentioned populations of encoded problem solutions. Please note that the representation is not necessarily limited to an application in a genetic algorithm. It could be used in other algorithmic approaches as well, e.g., a simulated annealing algorithm.

3 Representing a solution as a basis for schedule generation

3.1 Elements of a solution representation and interaction with a decoding procedure

As mentioned above, many powerful heuristic scheduling algorithms operate with a compact and problem-specific solution representation in combination with a suitable decoding procedure. As this decoding procedure derives a schedule from the representation, it is called a schedule generation scheme (SGS).

In a *serial* schedule generation scheme (SSGS), an activity is eligible for scheduling as soon as all predecessors are scheduled. For the eligible activity with the highest priority, the earliest feasible time with respect to capacity and precedence constraints is selected as starting time for this activity. In this approach it is possible that an activity which is considered late in the scheduling process may be located early in the final schedule due to the remaining capacity.

By contrast, in a *parallel* schedule generation scheme (PSGS), a schedule is built during a single pass along the time axis. An activity is eligible at the currently considered point in time if all predecessors are finished and if the remaining capacity is sufficient for the whole duration of this activity. The eligible activity with the highest priority is selected and scheduled. If no schedulable activity exists, the time-based procedure moves on to the next point in time when an activity finishes and/or free resources become available.

Both the SSGS and the PSGS are well-established methods in the project scheduling literature. The details for both procedures are given, e.g., in Hartmann and Kolisch (2000).

In a standard single-project RCPSP, the input to the SSGS or the PSGS will often be either an *activity list* (cf., e.g., Hartmann 1998) or a *random key vector* (cf., e.g., Bean 1994). In an activity list, the prioritization of the activities refers to the activity's position in the list, i.e., the earlier the activity is included in the list, the higher its priority. The random keys provide continuous priority values between 0 and 1 for each activity, i.e., the higher the activity's random key, the higher its priority.

Scheduling an entire portfolio of flexible projects subject to individual due dates requires a richer solution representation than a standard RCPSP. In particular, a solution has to address the following aspects of the underlying problem:

- Project structures
- Priorities of activities
- Priorities of projects
- Selection of the schedule generation scheme

The first two of these four aspects have already been addressed and documented in great detail in Kellenbrink and Helber (2015) and Kellenbrink (2014) for an activity-list based approach inspired by Hartmann (1998) in the single project case. As an additional approach we present a solution representation for those components based on random key vectors following the ideas in Bean (1994).

3.2 Representing a project structure

Inspired by the effectiveness of the random key representation for scheduling projects, cf., e.g., Gonçalves et al. (2008), we developed a *random key based representation* for the decision on the project structure. For each activity $j \in \mathcal{W}_{le}$ of a project l that is part of a decision e, exactly one element ϵ_{lj} with a continuous value between 0 and 1 indicates the priority value of that activity. When the representation is decoded, the activity $j^* \in \mathcal{W}_{le}$ with the highest priority value ϵ_{lj^*} is chosen for each triggered decision $e \in \mathcal{E}_l$. Note that only optional activities which are included in a decision set require priority values.

As an example, we consider the two projects introduced in Fig. 1 and the following vector $\epsilon = (\epsilon_{lj})$ of structure decision priorities:

$$\epsilon = \begin{pmatrix} - & - & \epsilon_{\text{X-3}} & \epsilon_{\text{X-4}} & - & \epsilon_{\text{X-6}} & \epsilon_{\text{X-7}} & - \\ - & - & - & \epsilon_{\text{Y-4}} & \epsilon_{\text{Y-5}} & \epsilon_{\text{Y-6}} & - & - \end{pmatrix}$$
$$= \begin{pmatrix} - & - & 0.6 & 0.8 & - & 0.4 & 0.1 & - \\ - & - & - & 0.7 & 0.2 & 0.6 & - & - \end{pmatrix}$$

We now use Algorithm 1 to decode this vector of random keys into project structure decisions for our portfolio consisting of two projects. In this process, the project structure is determined separately for each project l (Line 1). First of all, the mandatory activities are activated (Line 2). In project X, these are activities X-1, X-2, and X-8. In the next step, the choices are considered successively (Line 3) and it is checked whether the choice is triggered (Line 4). The first decision of project X is triggered by mandatory activity X-0 and thereby one of the optional activities $j \in \mathcal{W}_{X,1} = \{\text{X-3}, \text{X-4}\}$ has to be implemented. The activity j^* with the highest priority value has to be implemented (Line 5), in this case activity X-4 with $\epsilon_{\text{X-4}} = 0.8 > 0.6 = \epsilon_{\text{X-3}}$. All activities caused by activity j^* are implemented too (Line 6), so that activity X-5 $\in \mathcal{B}_{\text{X-4}}$ is activated. The for-loop (Line 3) now considers the next choice of project X. As the second (and last) decision of project X is not triggered, no further activities are implemented. The next project is considered in the same manner (Line 1). The resulting project structures of both projects X and Y for the given priority values are those shown in Fig. 2. As we further develop our example to explain the algorithm, we proceed with this project structure.

Algorithm 1: Determination of the Project Structure

```
1 for all Projects $l$ do
2     Implement all mandatory activities $j \in \mathscr{V}_l$
3     for all choices $e \in \mathscr{E}_l$ do
4         if if triggering activity $a(l,e)$ is implemented then
5             Implement activity $j^*$ with the highest priority value $j^* = \arg\max_{i \in \mathscr{W}_{le}} (\varepsilon_{l,i})$
6             Implement all caused activities $i \in \mathscr{B}_{l,j^*}$
```

3.3 Representing activitiy priorities

Priorities of activities can be presented using activity lists as in Hartmann (1998) for the RCPSP and in Kellenbrink and Helber (2015) for the RCPSP-PS or, again, via random keys as in Gonçalves et al. (2008), for the RCMPSP. One possible approach to transfer those ideas to the RCMPSP-PS is to combine all projects into a super-project consisting of all non-dummy activities as in the following activity list.

$$\lambda^{\text{super}} = (\text{X-2} \quad \text{X-5} \quad \text{Y-1} \quad \text{Y-4} \quad \text{X-1} \quad \text{X-4} \quad \cancel{\text{X-3}} \quad \text{Y-2} \quad \cancel{\text{Y-6}} \quad \cancel{\text{Y-5}} \quad \cancel{\text{X-6}} \quad \text{Y-3} \quad \text{X-8} \quad \text{Y-8} \quad \text{Y-7} \quad \cancel{\text{X-7}})$$

In this list, the activities not implemented in the project structures depicted in Fig. 2 are ignored. If this list is decoded using the standard SSGS as in Hartmann (1998), the schedule shown in Fig. 3 will result. It is also possible to operate with an activity list λ^{single} for single projects combined with a separate approach to establish priorities between those projects. Such activity lists for individual projects could look as follows:

$$\lambda^{\text{single}} = \begin{pmatrix} \text{X-2} & \text{X-5} & \text{X-1} & \text{X-4} & \cancel{\text{X-3}} & \cancel{\text{X-6}} & \text{X-8} & \cancel{\text{X-7}} \\ \text{Y-1} & \text{Y-4} & \text{Y-2} & \cancel{\text{Y-6}} & \cancel{\text{Y-5}} & \text{Y-3} & \text{Y-8} & \text{Y-7} \end{pmatrix}$$

In order to decode activity list λ^{single} into a schedule, additional information is needed to decide which project should be scheduled in which step, see Sect. 3.4.

As an alternative to using an activity list, we introduce a vector $\pi = (\pi_{l,j})$ of random keys which could look as follows in our example:

$$\pi = \begin{pmatrix} \pi_{\text{X-1}} & \pi_{\text{X-2}} & \pi_{\text{X-3}} & \pi_{\text{X-4}} & \pi_{\text{X-5}} & \pi_{\text{X-6}} & \pi_{\text{X-7}} & \pi_{\text{X-8}} \\ \pi_{\text{Y-1}} & \pi_{\text{Y-2}} & \pi_{\text{Y-3}} & \pi_{\text{Y-4}} & \pi_{\text{Y-5}} & \pi_{\text{Y-6}} & \pi_{\text{Y-7}} & \pi_{\text{Y-8}} \end{pmatrix}$$
$$= \begin{pmatrix} 0.5 & \underline{0.7} & \cancel{0.6} & 0.5 & 0.3 & \cancel{0.9} & \cancel{0.1} & 0.9 \\ 0.4 & 0.9 & \underline{0.1} & 0.2 & \cancel{0.7} & \cancel{0.3} & 0.7 & 0.8 \end{pmatrix}$$

Activity X-2 with a priority value of 0.7 is hence preferred to activity Y-3 with a value of only 0.1 and could be given priority when decoding the solution via the SSGS or the PSGS.

Please note that we define random keys for all non-dummy activities even though the random keys of those activities that are not implemented in the specific project structure (ignored in the example) are not used when decoding the priority vector π. However, as the priority values of the project structures and of the activities are determined independently within our representation, it is not possible to determine in advance which random keys are necessary so that they are always part of the solution representation.

3.4 Representing project priorities

In a multi-project situation, it is promising to consider the specific characteristics of the different projects, e.g., concerning the due date or tardiness costs, cf. Hoffmann et al. (2017). In our approach, if we decide to consider specific project priorities, which is *necessary* when we operate with project-specific activity lists λ^{single} and *possible* when we use the project-specific random keys π, we select the next activity to be scheduled out of the current set of eligible activities in two steps. Firstly, the project l^* with the highest project priority is chosen. In this step, all projects with at least one currently eligible activity are considered. Secondly, the eligible activity $j^* \in \mathcal{J}_{l^*}^{impl}$ with the highest priority of all eligible activities in project l^* is scheduled. To this end, we operate with a matrix μ of continuous random keys between 0 and

1. Each element of the matrix is multiplied by the corresponding project-dependent delay costs c_l. Thereby, we create *priority values for the project-specific scheduling steps* f_l. Consider the following example of priority values:

$$\mu \cdot c = \left(\mu_{l,f_l}\right) \cdot (c_l) = \begin{pmatrix} 0.2 & 0.2 & 0.3 & 0.4 & 0.1 & \cancel{0.8} & \cancel{0.5} & \cancel{0.9} \\ 0.3 & 0.9 & 0.1 & 0.3 & 0.8 & 0.7 & \cancel{0.7} & \cancel{0.5} \end{pmatrix} \begin{matrix} \cdot 7 \\ \cdot 3 \end{matrix}$$

$$= \begin{pmatrix} 1.4 & 1.4 & \underline{2.1} & 2.8 & 0.7 & \cancel{5.6} & \cancel{3.5} & \cancel{6.3} \\ 0.9 & \underline{2.7} & 0.3 & 0.9 & 2.4 & 2.1 & \cancel{2.1} & \cancel{1.5} \end{pmatrix}$$

We assume that for the introduced example two activities of project X (X-2 and X-5) and one activity of project Y (Y-3) have already been scheduled. In addition, we suppose that both projects are eligible as they possess an eligible activity. Therefore, in the next step it is intended to either schedule a third activity of project X ($f_X = 3$) **or** a second activity of project Y ($f_Y = 2$). As the priority for the second scheduling step of project Y is higher than the third step of project X ($\mu_{Y,2} \cdot c_Y = 2.7 > 2.1 = \mu_{X,3} \cdot c_X$), project $l^* = Y$ is chosen to be scheduled next.

The decision which activity out of the eligible activities in project Y is selected is then made based on either the activity list λ^{single} or the priorities π introduced in Sect. 3.3. The complete scheduling procedure is explained in Sect. 3.5.

With this approach, only as many random keys are necessary for each project as activities are implemented for the selected project structure. However, as this number varies with different project structures, always J_l random keys are defined for each project l. The remaining values for a specific project structure (ignored in our exemplary project structure) are omitted in the procedure. Please note that only the random keys μ but not the multiplied priority values $\mu \cdot c$ are used in the solution representation.

3.5 Flexible serial-parallel schedule generation scheme

As mentioned above, both activity lists and random key priorities can be decoded into a schedule using the SSGS or the PSGS. Extensive studies in literature show that for the classic RCPSP the results are often better for the SSGS, our *first* decoding variant. However, for some instances the PSGS provides benefits. Therefore, following Hartmann (2002), we study as a *second* approach a so-called self-adapting solution representation with an endogenous decision between the SSGS and PSGS to create the schedule. To refine this idea and gain further flexibility, we developed as a *third* alternative a flexible approach in which both SGS can be applied stepwise. Here the optimization decides endogenously which of them to use *in each step* of building the schedule. Each candidate solution contains the explicit information which SGS is used in the next step z to further construct the schedule. To this end we use a vector $\gamma = (\gamma_z)$ with potential entries "S" and "P" referring to the SSGS and the PSGS, respectively, as shown below for our previously established example:

$$\gamma = \begin{pmatrix} \mathrm{S} & \mathrm{P} & \mathrm{P} & \mathrm{S} & \mathrm{P} & \mathrm{S} & \mathrm{P} & \mathrm{S} & \mathrm{S} & \mathrm{P} & \mathrm{P} & \cancel{\mathrm{S}} & \cancel{\mathrm{S}} & \cancel{\mathrm{S}} & \cancel{\mathrm{S}} & \cancel{\mathrm{P}} \end{pmatrix}$$

It indicates that the first scheduling step is done using the SSGS, the second and third using the PSGS, and so on. As many elements as non-dummy activities ($\sum_{l\in\mathcal{L}} J_l$) are defined. For the project structure in Fig. 2, five activities will not be scheduled so that the last five entries of γ are ignored.

The procedure of the flexible serial-parallel SGS is shown in Algorithm 2 for the exemplary combination with project selection priorities and activity priorities. Note that it can operate both with activity lists and with random keys representing activity priorities.

Algorithm 2: Flexible Serial-Parallel Schedule Generation Scheme

```
1  Set period $\tau := 1$
2  Set scheduling step $z := 1$
3  Set project step $f_l := 1 \; \forall l$
4  while not all activities $j \in \mathcal{J}_l^{impl} \; \forall l$ are scheduled do
5      for all projects $l$ do
6          for all unscheduled activities $j \in \mathcal{J}_l^{impl}$ do
7              case $\gamma_z = S$ do
8                  if activity $j$ has no unscheduled predecessors then
9                      Define activity $j$ as eligible
10             case $\gamma_z = P$ do
11                 if activity $j$ has no unfinished predecessors in period $\tau$ then
12                     if there is enough remaining capacity on all resources $r$ for the whole
                         duration of activity $j$ when starting in period $\tau$ then
13                         Define activity $j$ as eligible
14     if there are projects with eligible jobs then
15         Choose eligible project $l^*$ with highest priority $\mu_{l^*,f_{l^*}} \cdot c_{l^*}$
16         Choose eligible activity $j^* \in \mathcal{J}_{l^*}^{\text{impl}}$ with highest priority $\pi_{l^*,j^*}$
17         case $\gamma_z = S$ do
18             Find earliest period $t$ that is feasible concerning the remaining capacity on
                 all resources $r$ for the whole duration of activity $j^*$ and concerning the
                 precedence relations
19             $ST_{l^*,j^*} = t$
20         case $\gamma_z = P$ do
21             $ST_{l^*,j^*} = \tau$
22         $z := z + 1$
23         $f_{l^*} := f_{l^*} + 1$
24         Adjust the remaining capacity
25     else
26         Set $\tau$ to the next period where at least one currently running activity ends and
             where capacities remains on at least one resource $r$
```

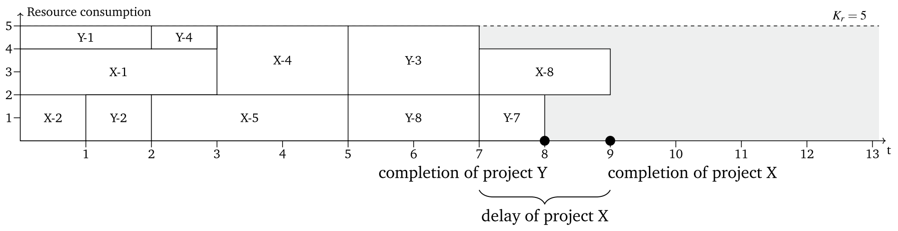


Fig. 5 Schedule resulting from the exemplary representation

As an example, we assume that the project structure in Fig. 2 has already been selected so that the sets of activities to be implemented $\mathcal{J}_l^{\text{impl}}, l \in \mathcal{L}$ have already been determined. We assume furthermore the following combination of schedule selection parameters γ, project selection priorities $\mu \cdot c$, and activity priorities π:

$$\gamma = \begin{pmatrix} S & P & P & S & P & S & P & S & S & P & P & \cancel{S} & \cancel{S} & \cancel{S} & \cancel{S} & \cancel{P} \end{pmatrix}$$

$$\mu \cdot c = \begin{pmatrix} 1.4 & 1.4 & 2.1 & 2.8 & 0.7 & \cancel{5.6} & \cancel{3.5} & \cancel{6.3} \\ 0.9 & 2.7 & 0.3 & 0.9 & 2.4 & 2.1 & \cancel{2.1} & \cancel{1.5} \end{pmatrix}$$

$$\pi = \begin{pmatrix} 0.5 & 0.7 & \cancel{0.6} & 0.5 & 0.3 & \cancel{0.9} & \cancel{0.1} & 0.9 \\ 0.4 & 0.9 & 0.1 & 0.2 & \cancel{0.7} & \cancel{0.3} & 0.7 & 0.8 \end{pmatrix}$$

The resulting schedule is shown in Fig. 5. Project X has a delay of two periods so that the objective function equals $Z = \sum_{l \in \mathcal{L}} v_l \cdot c_l = 2 \cdot 7 + 0 \cdot 3 = 14$. A detailed step-by-step explanation of this procedure is provided in Appendix A.

3.6 Improvement step

Once a schedule has been generated, an improvement step can be applied, cf. Li and Willis (1992). The basic idea is to first move activities forward in time without extending the project's makespan MS_l and then to move them backwards again. Thereby, it may be possible to exploit slack times in a way that leads to a reduced project makespan. For the case of multiple projects combined with due dates, our approach distinguishes between delayed and non-delayed projects. The method is presented in Algorithm 3.

In the first step, all delayed projects l are considered (Line 1). For those projects with positive delay $v_l = MS_l - \delta_l > 0$, a reduction of the makespan would lead to an improvement of the objective function. The implemented activities are considered in a descending order of end times ET_{lj}, so that the activity with the largest end time is considered first (Line 2). Each currently considered activity j is then postponed (i.e., right-shifted on the time axis) as much as possible without a *further* violation of the current project's due date ($ET_{lj} \leq MS_l$, Line 3). During this, the resource constraints as well as the precedence constraints are regarded. As the activities are considered in descending order of their end times, it is always assured that all implemented successors of a currently considered activity have already been right-shifted as far as possible. This ensures that the precedence constraints are always respected. Afterwards, all implemented activities are considered again, but now sorted by ascending start times ST_{lj} (Line 4). Each activity is moved backwards in time as much as possible considering the resource capacities and precedence restrictions (Line 5). This may lead to a reduction of the makespan so that the delay has to be updated (Line 6).

In a second step, all non-delayed projects are considered (Line 7). It is worth noting that projects accelerated in the first step so that they meet their due date may now be considered again. For these non-delayed projects, only the forward-improvement is applied. The corresponding activities are, beginning with the latest ending activity (Line 8), postponed as much as possible without violating the project's due date δ_l (Line 9). This approach is an attempt to actually make use of the time budget δ_l for non-delayed projects, thereby potentially releasing capacities especially in earlier periods that can then be used to reduce the makespan of currently delayed projects. Therefore, if there have been any changes in the overall schedule up to this step (Line 10), the forward-backward-improvement for delayed projects is applied again in the third step (Line 11).

Algorithm 3: Improvement Step

```
1  for all delayed projects $l$ ($v_l > 0$) do
2      for all activities $j \in \mathscr{J}_l^{impl}$ sorted by descending end times $ET_{lj}$ do
3          Schedule activity $j$ as late as possible considering resource and precedence
             restrictions with $ET_{lj} \leq MS_l$
4      for all activities $j \in \mathscr{J}_l^{impl}$ sorted by ascending start times $ST_{lj}$ do
5          Schedule activity $j$ as early as possible considering resource and precedence
             restrictions
6      $v_l = max(MS_l - \delta_l, 0)$
7  for all non-delayed projects $l$ ($v_l = 0$) do
8      for all activities $j \in \mathscr{J}_l^{impl}$ sorted by descending end times $ET_{lj}$ do
9          Schedule activity $j$ as late as possible considering resource and precedence
             restrictions with $ET_{lj} \leq \delta_l$
10 if there was any change in the overall schedule due to Lines 1 to 9 then
11     Repeat Lines 1 to 6
```

4 Genetic algorithms for solving the RCMPSP-PS

4.1 Different combinations of solution representation elements and decoding procedures

In the previous section, we addressed multiple elements of designing a compact solution representation for the RCMPSP-PS and a suitable decoding scheme. These

Table 2 Components describing a solution representation and decoding variant

Component	Characteristic
Project Structures	Random Keys ϵ
Priorities of Activities	Activity List λ^{super}, Activity List λ^{single}, Random Keys π
Priorities of Projects	No consideration, Random Keys $\mu \cdot c$
SGS	SSGS, Self-Adapting SGS, Flexible Serial-Parallel SGS γ
Improvement Step	No, yes

different aspects can be combined in numerous ways to create an entire family of closely related algorithms, each characterized by a specific combination of those previously presented components shown in Table 2.

However, not all combinations of the components are possible as, e.g., always information on the project priorities is needed if the activity list for single projects λ^{single} is used. One particular variant of a solution representation and decoding approach could, e.g., use decision priorities ϵ (Sect. 3.2), project-specific activity lists λ^{single} (Sect. 3.3) to model activity priorities in an approach that does explicitly consider project priorities $\mu \cdot c$ (Sect. 3.4), and the flexible serial-parallel schedule generation scheme (Sect. 3.5), and try to improve each generated schedule via the forward-backward improvement step (Sect. 3.6).

The combination of the elements describes how a *single* solution is represented and decoded into an actual schedule for which an objective function value can be computed. In the next sections we explain how our genetic algorithm generates the initial *population* of encoded solutions and how this population is changed from iteration to iteration.

4.2 General procedure

Genetic algorithms were firstly introduced by Holland (1975) and have been successfully applied to many combinatorial optimization problems, including the classical RCPSP and its extensions, cf., e.g., Hartmann and Kolisch (2000).

Each solution of the underlying problem is interpreted as one individual and therefore described, e.g., by representation components ϵ, λ, μ and γ as introduced in Sect. 3. Each single element of such a component is named a gene. The objective function value of a solution gives this individual's fitness. As a population-based heuristic, a genetic algorithm considers multiple solutions simultaneously. These are recombined to create potentially new individuals of a further generation using a crossover operator. Minor mutations of the newly created solutions should contribute to the investigation of the whole solution space. Afterwards, the expanded set of individuals is reduced down to its original size based on the individuals' fitness in a selection process to form the population for the next generation. In this way, the individuals should adapt to the requirement of the problem in order to create better solutions. The different components of the genetic algorithm are described in Sects. 4.3–4.6.

4.3 Generating the initial population

The *POP* individuals of the initial population are generated randomly. As we use random keys for the prioritization of the project structure (ϵ) and, if applicable, the projects (μ), their individual genes can be generated independently from each other. For each gene, a uniformly distributed value between 0 and 1 is randomly determined. If random keys are also used to prioritize activities, the same procedure is applied to the π values. If otherwise an activity list λ^{super} or λ^{single} is used, it is initialized in a randomized order, cf. Kellenbrink and Helber (2015).

If we operate with the self-adapting SGS or the flexible serial-parallel SGS γ, we randomly determine whether the SSGS or the PSGS should be applied for the whole schedule or for each potentially possible scheduling step, respectively. Afterwards, the fitness of each individual of this initial population is determined using the respective SGS and, if applicable, the improvement step in Algorithm 3.

4.4 Crossover

Within the crossover, a pair of randomly chosen individuals I^M (mother) and I^F (father) are combined to create two new individuals I^D (daughter) and I^S (son). Each individual of the current generation is thereby used exactly once as a mother or as a father. A so-called parameterized uniform crossover (cf., e.g., Spears and de Jong 1991) is used to decide for each gene from which of the two parents the child's gene is taken. With an exogenously given probability q a daughter's gene is inherited from the mother. Otherwise the gene of the father will be passed to the daughter.

The precise implementation of the crossover operator varies slightly on the chosen combination of solution representation and decoding scheme as outlined in Sect. 4.1. In Algorithm 4 we show how to create a new daughter I^D for the case in which we operate with random key priorities π for the activities, consider project priorities μ, and use the flexible serial-parallel SGS γ. This turned out to be the most promising combination as will be seen in our numerical results. The creation of a son individual I^S is equivalent with exchanged roles of the individuals I^M and I^F. Accordingly, *POP* new individuals are created in each generation.

The majority of the daughter's genes are determined project-wise (Line 1). The procedure starts with the crossover for the decision priorities ϵ. For each decision $e \in \mathcal{E}_l$ of a project l (Line 2) and for each activity in the decision set $j \in \mathcal{W}_{le}$ (Line 3) a continuous random value *cross* between 0 and 1 is determined (Line 4). If the determined value is less than or equal to the probability q^ϵ, the daughter's priority value ϵ^D_{lj} is inherited from the mother I^M. Otherwise, the corresponding priority value of the father I^F is selected (Line 5).

In the next step, the activity priorities π^D are determined for each non-dummy activity $j = 1, \dots, J_l$ (Line 6). If the random number *cross* (Line 7) is less than or equal to the probability q^π, the activity priority of the daughter $\pi^D_{l,j}$ is taken from the mother I^M and otherwise from the father I^F (Line 8).

For the daughter's prioritization of projects μ^D, for each possible scheduling step f_l (Line 9) a random number *cross* between 0 and 1 is determined again (Line 10). Depending on this either the random key μ^M_{l,f_l} of the mother or the random key μ^F_{l,f_l} of the father is selected (Line 11).

The information on the flexible serial-parallel SGS γ^D is determined independently from the projects. For each possible scheduling step z (Line 12) a continuous random number *cross* between 0 and 1 is determined (Line 13) and the information about the selected SGS γ^D_z is taken from the mother I^M if *cross* is less than or equal to q^γ and from the father otherwise (Line 14).

The crossover for other combinations of solution representations can essentially be created by omitting elements of Algorithm 4. The crossover of the self-adapting SGS resembles those of the flexible serial-parallel SGS with a single element. The parameterized uniform crossover operator applied to activity lists operates in a similar gene-by-gene manner while ensuring that each generated individual contains all the activities, cf. Hartmann (1998).

Algorithm 4: Crossover operator to generate a new daughter individual

```
1  for all Projects $l$ do
2      for all choices $e \in \mathscr{E}_l$ do
3          for all activities $j \in \mathscr{W}_{le}$ do
4              Determine cross $\sim U(0,1)$
5              if cross $\leq q^\varepsilon$ then $\varepsilon^D_{l,j} = \varepsilon^M_{l,j}$ else $\varepsilon^D_{l,j} = \varepsilon^F_{l,j}$
6      for all activities $j = 1,...,J_l$ do
7          Determine cross $\sim U(0,1)$
8          if cross $\leq q^\pi$ then $\pi^D_{l,j} = \pi^M_{l,j}$ else $\pi^D_{l,j} = \pi^F_{l,j}$
9      for all project steps $f_l = 1,...,J_l$ do
10         Determine cross $\sim U(0,1)$
11         if cross $\leq q^\mu$ then $\mu^D_{l,f_l} = \mu^M_{l,f_l}$ else $\mu^D_{l,f_l} = \mu^F_{l,f_l}$
12 for all scheduling steps $z = 1,...,\sum_{l\in\mathscr{L}} J_l$ do
13     Determine cross $\sim U(0,1)$
14     if cross $\leq q^\gamma$ then $\gamma^D_z = \gamma^M_z$ else $\gamma^D_z = \gamma^F_z$
```

4.5 Mutation

The mutation should extend the search to further parts of the solution space. For each newly created individual, each gene is mutated with the (small) probability p. In case of a mutation of a gene belonging to the random key based components (ϵ, μ, π), a new continuous number between 0 and 1 is chosen randomly.

When using the SSGS in combination with activity lists, after mutating the random keys representing the project structure, a final repair step as in Kellenbrink and Helber (2015) ensures that precedence constraints between implemented activities are respected. The repair step is not necessary for the PSGS as an activity is only eligible when the precedence restrictions are fulfilled.

With probability p^λ a gene in the activity list is mutated. In this case we try to exchange the position of the associated activity i with the position of the next implemented activity j. However, if activity j is a successor of activity i, the gene will not be mutated in order to avoid creating invalid activity lists.

In case of a mutation of a gene selecting the SGS for the self-adapting SGS or for the flexible serial-parallel SGS, the SSGS is selected if the PSGS was previously included and vice versa.

4.6 Selection

After adding the newly generated individuals, the population size has temporarily been doubled to $2 \cdot POP$. To reduce it again to a population size of POP individuals, we use the so-called elite selection or ranking method, cf. e.g. Hartmann (1998). In this step, the worst POP individuals which have the highest delay cost are deleted from the population. The remaining individuals then form the next generation from which new individuals are created again.

5 Numerical study

5.1 Test design

Our numerical study addresses two distinct questions. One asks from a managerial point of view for the economic benefit of being able to coordinate entire portfolios of flexible projects via simultaneous optimization, see Sect. 2.5. The other question is related to the speed and accuracy of our proposed algorithmic approach, to be studied below. To answer both questions, we systematically created test instances with different parameter combinations. As a basis, the instance generator for *single* flexible projects from Kellenbrink and Helber (2015) was used, which in turn generalizes the instance generator ProGen by Kolisch et al. (1995). Our generator creates for each instance a portfolio of projects and for each project a due date and a tardiness cost. A project's tardiness cost per time unit is randomly chosen between 1 and 10.

To set the due dates δ_l of the projects $l \in \mathcal{L}$, two contradictory aspects must be balanced. If due dates are too tight, possibly approaching 0, our problem turns into a weighted makespan minimization problem. If the due dates are too loose, schedules without any delay can be found easily and the practical difficulty and relevance of the problem essentially disappears. In order to strike the delicate balance between those two aspects, we first determine for each project l the length of the shortest possible critical path CP_l under the assumption of *infinite* capacity by solving the mathematical model described in Sect. 2.4 without the capacity restrictions and minimizing the end times of all implemented activities. This critical path length CP_l is multiplied by a varying due date factor D to define the due date $\delta_l = D \cdot CP_l$.

The resource capacities K_r are calculated based on those unrestricted schedules. The minimal capacity K_r^{min} is determined by the maximum resource consumption of a single activity which is implemented in any of the corresponding project structures. The maximal resource capacity K_r^{max} is equivalent to the maximum capacity that is needed to execute the unrestricted schedules of all projects simultaneously. The actual capacities K_r are then calculated using Eq. (10) with different values for the resource strength RS_r, cf. Kolisch et al. (1995).

$$K_r = K_r^{min} + round(RS_r \cdot (K_r^{max} - K_r^{min})) \quad \forall r \in \mathcal{R} \tag{10}$$

Two problem classes named P2J15 (2 projects with 15 non-dummy jobs each) and P4J30 (4 projects with 30 non-dummy jobs each) were created, see Table 3. The number of decisions $|\mathcal{E}_l|$ per project is denoted as N_l^E and the number of activities that cause other activities as N_l^C, cf. Kellenbrink and Helber (2015).

Other parameters were varied in the same manner for all problem classes, see Table 4. The first two lines refer to the flexibility of the project structure, namely to the number of optional activities per decision N_l^W and the number of activities that may be caused by one optional activity N_l^B. The number of renewable resources $|\mathcal{R}|$ as well as the resource factor RF_r and the resource strength RS_r define the resource demand and supply. The network complexity NC provides information on the relative number of precedence relations whereas the due date factor D is used to compute due dates $\delta_l = D \cdot CP_l$ as described above. For further details on these key metrics see Kolisch et al. (1995) as well as Kellenbrink and Helber (2015). All different parameter combinations in Tables 3 and 4 result in 864 instances per problem class.

For the computation of the earliest and latest finish times, an upper bound for the planning horizon is needed. However, the sum of the durations of all activities of all projects overestimates the number of periods, especially for large instances with much possibility for parallelization and with a high flexibility. Therefore, a

Table 3 Parameters for the different classes of test instances

Problem class	$\lvert\mathcal{L}\rvert$	J_l	N_l^E	N_l^C
P2J15	2	15	{1; 2}	{1; 2}
P4J30	4	30	{2; 4}	{2; 4}

Table 4 Parameters for each class of test instances

Parameter	#	Values		
N_l^W	2	2	3	
N_l^B	2	1	2	
$\lvert\mathcal{R}\rvert$	1	2		
RF_r	2	0.5	1	
RS_r	3	0.25	0.5	0.75
NC	3	1.5	1.8	2.1
D	3	1.0	1.1	1.2

project-specific schedule is determined heuristically using different basic priority rules. It reflects an upper bound of the minimal makespan of *one* project. The sum of these feasible makespans of all projects is then used as an upper bound for the time horizon. Within a preprocessing procedure, infeasible project structures, i.e., structures in which at least one activity exceeds the capacity of at least one renewable resource, are excluded.

To generate reference values with Gurobi 8.1.0, we implemented the mathematical model in GAMS 26.1.0 and used a 2.00 GHz Backton Xeon machine with 20 GB of RAM and 4 threads in a cluster node at Leibniz University Hannover, cf. www.luis.uni-hannover.de/scientific_computing. Since it was not possible to solve all instances to a proven optimal solution, we limited the computation time to 3600 s per instance.

We implemented the genetic algorithm in C++ and executed it on the same machine but now with only 5 GB of RAM and on a single thread. $POP = 80$ individuals were created in each generation. With a probability $q^{\epsilon} = q^{\lambda} = q^{\pi} = q^{\mu} = q^{\gamma} = 0.6$, the mother's gene is used in the crossover to generate a daughter individual. The mutation probabilities equal $p^{\epsilon} = 0.08$ for the project structures ϵ and $p^{\mu} = p^{\pi} = p^{\lambda} = p^{\gamma} = 0.1$ for remaining components.

5.2 Results

For problem class P2J15 with 2 projects and 15 non-dummy activities each, Gurobi was able to solve all 864 instances to proven optimality. 26 instances have an optimal objective function value of zero, i.e., all projects of the respective instance are schedulable without any delay. Those instances were ignored in the following when calculating the mean deviation to the optimal solution. Gurobi needed an average of 45.22 s to solve the instances to optimality. While only 11.11% of the instances needed more than 60 s to be solved to optimality with Gurobi, the maximum computation time to proven optimality was 2363.64 s, i.e., even those small problems can be hard to solve.

Table 5 shows results of the 24 different genetic algorithm variants for a computation time of 1, 60 and 300 s per instance. The percentage of instances for which an optimal solution was found by the respective representation and the mean deviation from the optimal solution are given. The best solution per column is highlighted dark green and the worst solution per column dark red.

The representations using random keys π to prioritize the activities clearly dominate the activity lists λ. The same holds true for the improvement step. Using project priorities $\mu \cdot c$ tends to lead to better results. The flexible serial-parallel SGS dominates the other SGS for the promising alternatives with random keys π and improvement step. Overall, the best results with a computational time of 1 s are achieved by the variant with project prioritization $\mu \cdot c$, a prioritization of activities by random keys π and with the flexible serial-parallel SGS with subsequent improvement step. With this representation, 92.82% of the instances are already solved to optimality with an average deviation of 0.59% after 1 s of computation time.

Table 5 Results for the problem class with 2 projects and 15 activities each

				1 sec.		60 sec.		300 sec.	
				Optimum		Optimum		Optimum	
Priorities of Projects	Priorities of Activities	SGS	Improvement Step	Optimal	Deviation	Optimal	Deviation	Optimal	Deviation
no consideration	activity list λ^{super}	SSGS	no	62.85%	7.59%	72.80%	4.72%	76.27%	4.04%
			yes	74.65%	3.41%	81.71%	2.09%	83.80%	1.68%
		self-adapting SGS	no	63.66%	7.45%	76.16%	3.86%	78.94%	3.12%
			yes	73.61%	3.62%	82.52%	2.29%	85.19%	1.80%
		flexible serial-parallel SGS	no	64.35%	6.90%	75.12%	4.08%	78.94%	3.15%
			yes	74.77%	3.88%	83.45%	2.09%	86.81%	1.53%
	random keys π	SSGS	no	85.30%	1.74%	92.59%	0.60%	93.98%	0.45%
			yes	88.08%	0.86%	93.06%	0.38%	94.56%	0.30%
		self-adapting SGS	no	84.38%	1.41%	92.94%	0.53%	95.02%	0.39%
			yes	90.97%	0.80%	95.72%	0.26%	96.99%	0.19%
		flexible serial-parallel SGS	no	87.96%	1.50%	95.25%	0.41%	97.34%	0.27%
			yes	91.90%	0.64%	96.53%	0.20%	97.22%	0.13%
random keys $\mu \cdot c$	activity list λ^{single}	SSGS	no	80.56%	2.40%	89.47%	0.99%	91.09%	0.83%
			yes	83.56%	1.65%	90.97%	0.82%	92.48%	0.57%
		self-adapting SGS	no	57.06%	8.31%	60.88%	7.55%	62.15%	7.27%
			yes	67.71%	5.52%	70.72%	4.88%	71.88%	4.65%
		flexible serial-parallel SGS	no	83.56%	2.05%	91.09%	0.95%	93.29%	0.80%
			yes	88.89%	1.04%	93.87%	0.40%	95.49%	0.29%
	random keys π	SSGS	no	86.23%	1.34%	91.90%	0.62%	94.91%	0.33%
			yes	89.12%	0.94%	93.17%	0.44%	94.91%	0.32%
		self-adapting SGS	no	85.76%	1.57%	93.40%	0.41%	95.49%	0.29%
			yes	89.93%	0.73%	95.37%	0.27%	96.76%	0.17%
		flexible serial-parallel SGS	no	90.16%	0.93%	95.72%	0.35%	96.88%	0.24%
			yes	92.82%	0.59%	96.99%	0.17%	97.92%	0.13%

For the larger instances in problem class P4J30 with 4 projects and 30 non-dummy activities each, Gurobi was given a time limit of 3600 s per instance. The average actual calculation time per instance was 1373.23 s with an average gap to the lower bound of the linear programming relaxation of 34.15%. The fraction of proven optimal instances was 63.19%. 41 instances were solved with an objective function value of zero and, as before, ignored when calculating mean deviations.

Compared to the results of all 24 variants of the genetic algorithm with computation times of 1 s per instance, Gurobi (also) finds the best known solution for 79.28% of the instances. For the remaining instances, the genetic algorithms found better solutions. The average deviation from the Gurobi upper bounds to the best known upper bounds including the genetic algorithms was 5.9% for the cases with positive objective function values.

Table 6 compares the genetic algorithms for 1, 60, and 600 s of computation time to the respective upper bound Gurobi solutions and best known upper bounds over all 25 solution approaches, i.e., the 24 variants of the genetic algorithm for the given computation time *and* Gurobi.

With a computation time of 1 s, the results differ from that of the small P2J15 instances. For this short computation time, again random keys π and the improvement step, but now using the self-adapting SGS without project prioritization lead to the best results. This alternative finds for 65.28% of the instances a solution that is at least as good as those found by Gurobi, however with a mean deviation of the Gurobi upper bounds of 7.99%. For 52.08% of the instances, the solution found equals the best known solution for this computation time of the genetic algorithm (and up to 3600 s for Gurobi), but the average deviation from the best known solution amounts to 12.43%.

With a longer computation time of 60 s per instance for the genetic algorithms, the alternative with project prioritization $\mu \cdot c$ using the flexible serial-parallel SGS

Table 6 Results for the problem class with 4 projects and 30 activities each

				1 sec.				60 sec.				600 sec.			
				Gurobi		Best Known		Gurobi		Best Known		Gurobi		Best Known	
Priorities of Projects	Priorities of Activities	SGS	Improvement Step	Better/ Equal	Deviation	Best	Deviation	Better/ Equal	Deviation	Best	Deviation	Better/ Equal	Deviation	Best	Deviation
no consideration	activity list λ^{super}	SSGS	no	30.90%	74.68%	28.82%	82.03%	41.09%	42.11%	36.92%	51.73%	47.57%	29.41%	41.78%	40.13%
			yes	45.95%	36.71%	43.06%	43.42%	50.69%	25.15%	45.95%	34.51%	54.05%	19.88%	47.69%	30.46%
		self-adapting SGS	no	31.94%	65.32%	29.98%	72.47%	40.51%	43.39%	36.57%	53.12%	48.61%	29.53%	42.59%	40.21%
			yes	47.22%	33.89%	43.75%	40.62%	52.20%	24.31%	47.69%	33.69%	56.71%	17.80%	50.12%	28.25%
		flexible serial-parallel SGS	no	32.75%	68.22%	30.56%	75.49%	39.58%	44.73%	36.00%	54.56%	48.61%	26.90%	42.94%	37.59%
			yes	47.69%	34.34%	44.56%	41.14%	52.89%	23.63%	48.03%	33.02%	56.48%	17.18%	49.65%	27.65%
	random keys π	SSGS	no	58.22%	17.63%	45.25%	22.16%	70.95%	7.33%	51.85%	13.74%	75.35%	3.57%	53.59%	10.99%
			yes	64.24%	9.65%	50.12%	14.06%	73.84%	2.93%	53.94%	9.15%	76.62%	0.73%	54.40%	7.96%
		self-adapting SGS	no	57.52%	15.65%	45.14%	20.18%	67.94%	5.29%	50.46%	11.68%	74.19%	2.11%	53.01%	9.51%
			yes	65.28%	7.99%	52.08%	12.43%	74.42%	1.35%	55.56%	7.57%	78.01%	-0.54%	55.56%	6.68%
		flexible serial-parallel SGS	no	54.40%	19.94%	44.68%	24.85%	70.14%	4.68%	51.97%	11.08%	75.81%	1.03%	55.32%	8.41%
			yes	60.42%	13.07%	49.19%	17.90%	74.77%	1.55%	56.37%	7.80%	78.94%	-0.64%	58.45%	6.53%
random keys $\mu \cdot c$	activity list λ^{single}	SSGS	no	52.08%	24.48%	38.43%	28.96%	68.40%	8.29%	48.38%	14.55%	72.57%	3.25%	51.27%	10.48%
			yes	59.95%	13.85%	46.64%	18.23%	69.91%	5.41%	51.74%	11.65%	73.84%	2.22%	52.89%	9.44%
		self-adapting SGS	no	47.69%	25.47%	35.88%	30.18%	58.45%	12.30%	42.71%	18.90%	64.12%	6.89%	45.37%	14.49%
			yes	56.94%	16.35%	44.68%	20.95%	64.24%	7.87%	47.80%	14.38%	68.75%	4.64%	49.88%	12.14%
		flexible serial-parallel SGS	no	51.04%	22.62%	38.43%	27.31%	66.32%	8.42%	48.26%	14.83%	74.54%	1.76%	52.31%	8.99%
			yes	58.33%	14.90%	44.44%	19.48%	69.79%	4.24%	50.93%	10.49%	75.93%	0.48%	54.51%	7.63%
	random keys π	SSGS	no	59.49%	16.14%	46.18%	20.43%	72.22%	4.21%	52.89%	10.28%	76.16%	1.14%	54.40%	8.21%
			yes	63.89%	10.34%	51.04%	14.56%	73.73%	2.48%	54.28%	8.55%	77.43%	0.25%	55.90%	7.28%
		self-adapting SGS	no	57.64%	15.54%	43.87%	19.87%	68.75%	5.01%	50.23%	11.15%	73.50%	1.99%	53.13%	9.14%
			yes	64.35%	10.29%	51.04%	14.55%	72.22%	3.36%	52.89%	9.43%	76.50%	0.52%	54.51%	7.58%
		flexible serial-parallel SGS	no	54.17%	16.68%	41.09%	21.22%	73.26%	2.24%	54.05%	8.26%	79.17%	-0.99%	58.33%	5.93%
			yes	60.42%	12.06%	45.83%	16.53%	76.27%	0.03%	59.14%	5.98%	81.25%	-2.24%	61.00%	4.63%

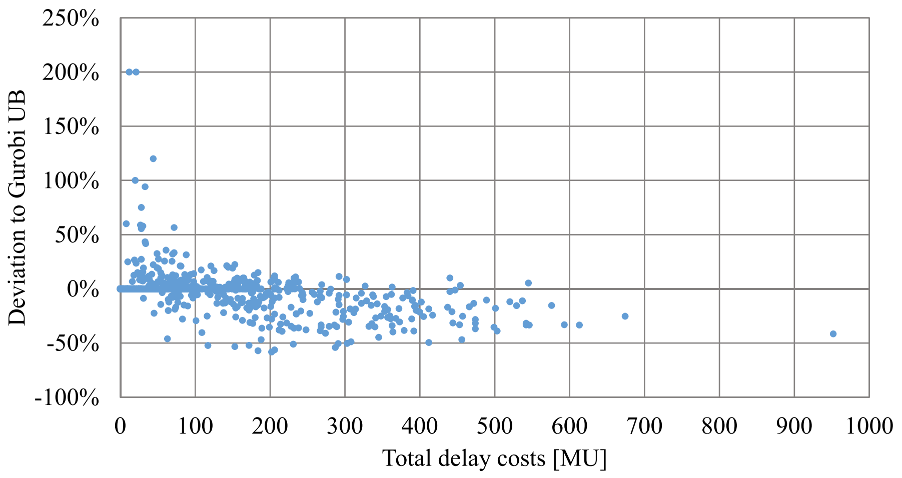


Fig. 6 Objective function values and percentage deviations

dominates again. The average deviation from Gurobi is only 0.03% so that the performance of both approaches is nearly the same but with a much shorter time limit for the genetic algorithm. For longer computation times of 600 s per instance, the genetic algorithm outperforms Gurobi on average.

Figure 6 clarifies an important relationship between objective function values and relative deviations for the bottom line algorithm variant from Table 6 and 600 s of computation time. For each instance, the relative deviation of the genetic algorithm solution from the Gurobi solution is given together with the total delay cost found with the genetic algorithm. There are some instances that can be solved with only very minor delays of the projects and hence only small objective function values. For those projects, the genetic algorithm sometimes finds solutions that are only slightly more expensive in *absolute* terms, but twice or three times as expensive in *relative* terms. When penalty payments become larger due to longer delays, however, the algorithm performs well.

Overall, we conclude that the described genetic algorithms are a useful approach to schedule multiple flexible projects. In particular, our results show that random keys π as proposed by Gonçalves et al. (2008) beat activity lists λ for activity prioritization even in the context of entire portfolios of flexible projects.

In addition, project priorities $\mu \cdot c$ as introduced in this paper lead to good results which is also promising for other multi-project settings. The newly introduced flexible serial-parallel SGS is useful as well. For this reason, it could be an attractive idea to test the flexible serial-parallel SGS for other types of resource-constrained project scheduling problems.

6 Conclusion

In this paper, we analyzed and formally modeled the problem to find both structures and schedules for entire portfolios of projects. We considered projects with flexible structures due to alternative procedures or technologies. For each of the projects, we assumed given project-specific due dates and agreed-upon penalty payments for each time unit of project delay. Tight capacity constraints can lead to situations in which some degree of delay and hence some penalty payments are inevitable. In this situation, it is both difficult and important to coordinate the different projects in a way which uses project flexibility options in an economically efficient way. The substantial cost benefit of the proposed simultaneous optimization approach can be seen clearly from the comparison with other approaches as such capacity quotas for or sequential scheduling of different projects within a portfolio.

In order to solve such projects and in line with the existing literature, we developed different solution representations which we then used with genetic algorithms. By combining different algorithmic options, we created an entire family of 24 different but related algorithms. These algorithms were tested in a large-scale numerical study. The performance of those algorithms turned out to differ substantially. We found that it is beneficial to explicitly consider project priorities in the solution representation. Our experiments confirm that random keys should be used instead of activity lists to prioritize activities. Furthermore, it is beneficial to operate with a flexible schedule generation scheme with frequent endogenous decisions between serial and parallel schedule generation. Finally, we saw that an improvement step tends to actually improve the solution quality.

For relatively small problem instances Gurobi could be used to determine proven optimal solutions. Our best algorithm showed only a negligible deviation from these optimal solutions. For larger instances, Gurobi could not always provide proven optimal solutions and was outperformed by our algorithms with respect to both solution quality and computation time. While our algorithms have been designed to solve multi-project scheduling problems with due dates, they can also be used to solve weighted makespan minimization problems as those arise if due dates are zero for all projects. Given the modeling power of *flexible* projects and the speed and accuracy of our tailored algorithms, a wide variety of such problems can now be solved for non-trivial problem sizes, as indicated by our results.

Artigues (2017) has shown that different formulations of the RCPSP can lead to better results when using standard solvers. Further research could therefore discuss the influence of the formulation of the RCMPSP-PS on the computation time of standard solvers.

Furthermore, future research could address project portfolio scheduling problems under uncertainty. If there is flexibility with respect to the problem structure, it might be beneficial to make a first-stage decision about the project structures subject to some activity duration uncertainty and then determine the schedule as actual activity durations are unveiled.

Acknowledgements Open Access funding provided by Projekt DEAL. The authors thank the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) for funding this research within the Collaborative Research Centre 871 “Regeneration of complex durable goods” under grant agreement SFB 871/3–119193472. We also thank the anonymous reviewers for their constructive and insightful questions, comments, and suggestions.

Appendix 1: Detailed description of the flexible serial-parallel schedule generation

This appendix explains in detail how the schedule shown in Fig. 5 is created using Algorithm 2 for the example data given in Sect. 3.5.

At first, the values for τ, z and f_l are initialized (Lines 1–3). Since not all activities are scheduled yet (Line 4), we consider all projects l and all unscheduled but implemented activities $j \in \mathcal{J}_l^{impl}$ successively to determine which activities are eligible (Lines 5–6). In the first scheduling step $z = 1$ the SSGS is applied ($\gamma_1 = S$, Line 7). In this case, an activity is eligible if all predecessors are already scheduled (Line 8–9). In our example these are activities X-1, X-2, Y-1, Y-2 and Y-3. As there are eligible jobs, the procedure is continued for the given period τ (Line 14). Since project X has the highest priority value of both eligible projects ($\mu_{X,1} = 1.4 > 0.9 = \mu_{Y,1}$), it is selected ($l^* = X$, Line 15). Activity $j^* =$ X-2 has the highest priority out of all eligible jobs of project X and is chosen (Line 16). Since we use the SSGS in this step (Line 17), the starting period is defined as the earliest period in which all predecessors are finished and in which there is enough remaining capacity to schedule activity X-2 (Line 18). Due to the fact that no activity is scheduled yet, this is the case in period $t = 1$, so that activity X-2 is determined to start at the beginning of this period ($ST_{\text{X-2}} = 1$, Line 19). Therewith, the first scheduling step is completed and z as well as f_X are increased by one (Lines 22–23). The remaining capacity in period 1 is reduced to 3 units as activity X-2 consumes 2 units for a duration of one period (Line 24).

Not all implemented activities are scheduled yet and the procedure continues (Line 4). Again, it is checked which unscheduled but implemented activities are eligible. Due to the fact that the PSGS is used in the second planning step ($\gamma_2 = P$, Line 10), the completion of the predecessors and the resource availability in the current period $\tau = 1$ is checked (Lines 11–13). The activities X-1, Y-1, Y-2 and Y-3 are eligible. Project X is chosen ($\mu_{X,2} = 1.4 > 0.9 = \mu_{Y,1}$). For this project, only activity X-1 is eligible and thus is scheduled. It starts in the current period τ ($ST_{\text{X-1}} = 1$, Lines 20–21). The counters z and f_l^* are adjusted and the remaining capacity is reduced to 1 unit in period 1 and to 3 units in periods 2 and 3 as activity X-1 consumes two units of capacity for three periods.

The algorithm proceeds with the PSGS in the next scheduling step ($\gamma_3 = P$). In the current period τ there is only enough capacity to schedule activity Y-1 ($ST_{\text{Y-1}} = 1$).

In the fourth planning step, the SSGS is applied ($\gamma_4 = S$). There is no capacity left in period $\tau = 1$. However, for the SSGS the available capacity and the *completion* of the predecessors are not checked when determining the schedulable activities. Thus, activities X-4, X-5, Y-2, Y-3 and Y-4 are eligible as all their predecessors are *scheduled*. Due to the priority values for the projects and the activities, activity Y-2 is planned to start in the earliest period in which all predecessors are finished and in which there is enough capacity left ($ST_{\text{Y-2}} = 2$).

In the fifth scheduling step again the PSGS is used ($\gamma_5 = P$). As this scheme focuses on the current period $\tau = 1$, in which no capacity is left, no activities are eligible (Line 25). Thus, the period τ is increased to the earliest period in which at least one activity ends and in which there is capacity left on at least one resource r (Line 26). In our example, this is the second period as the already scheduled activities consume the whole capacity before. For this reason, the eligibility is checked for $\tau = 2$.

The procedure continues until all activities are scheduled.

References

Artigues C (2017) On the strength of time-indexed formulations for the resource-constrained project scheduling problem. Oper Res Lett 45:154–159

Bean JC (1994) Genetic algorithms and random keys for sequencing and optimization. ORSA J Comput 6:154–160

Belhe U, Kusiak A (1995) Resource constrained scheduling of hierarchically structured design activity networks. IEEE Trans Eng Manag 42:150–158

Birjandi A, Mousavi SM, Hajirezaie M, Vahdani B (2019) A new weighted mixed integer nonlinear model and FPND solution algorithm for RCPSP with multi-route work packages under fuzzy uncertainty. J Intell Fuzzy Syst 37:737–751

Birjandi A, Mousavi SM (2019) Fuzzy resource-constrained project scheduling with multiple routes: a heuristic solution. Autom Constr 100:84–102

Blazewicz J, Lenstra JK, Rinnooy Kan AHG (1983) Scheduling subject to resource constraints: classification and complexity. Discrete Appl Math 5:11–24

Browning TR, Yassine AA (2016) Managing a Portfolio of product development projects under resource constraints. Decis Sci 47:333–372

Brucker P, Drexl A, Möhring R, Neumann K, Pesch E (1999) Resource-constrained project scheduling: notation, classification, models, and methods. Eur J Oper Res 112:3–41

Cajzek R, Klanšek U (2019) Cost optimization of project schedules under constrained resources and alternative production processes by mixed-integer nonlinear programming. Eng Constr Archit Manag 26:2474–2508

Čapek R, Šucha P, Hanzálek Z (2012) Production scheduling with alternative process plans. Eur J Oper Res 217:300–311

Chiu HN, Tsai DM (2002) An efficient search procedure for the resource-constrained multi-project scheduling problem with discounted cash flows. Constr Manag Econ 20:55–66

Demeulemeester EL, Herroelen W (2002) Project scheduling: a research handbook. Kluwer Academic Publishers, Boston

Gonçalves JF, Mendes JJM, Resende MGC (2008) A genetic algorithm for the resource constrained multi-project scheduling problem. Eur J Oper Res 189:1171–1190

Habibi F, Barzinpour F, Sadjadi SJ (2018) Resource-constrained project scheduling problem: review of past and recent developments. J Proj Manag 3:55–88

Hartmann S (1998) A competitive genetic algorithm for resource-constrained project scheduling. Nav Res Log 45:733–750

Hartmann S, Kolisch R (2000) Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. Eur J Oper Res 127:394–407

Hartmann S (2002) A self-adapting genetic algorithm for project scheduling under resource constraints. Nav Res Log 49:433–448

Hartmann S, Briskorn D (2010) A survey of variants and extensions of the resourceconstrained project scheduling problem. Eur J Oper Res 207:1–14

Hauder VA, Beham A, Raggl S, Parragh SN, Affenzeller M (2019) On constraint programming for a new flexible project scheduling problem with resource constraints. arXiv:1902.09244

Herroelen W, de Reyck B, Demeulemeester E (1998) Resource-constrained project scheduling: a survey of recent developments. Comput Oper Res 25:279–302

Hoffmann L-S, Kuprat T, Kellenbrink C, Schmidt M, Nyhuis P (2017) Priority rulebased planning approaches for regeneration processes. Proc CIRP 59:89–94

Hoffmann L-S, Kellenbrink C (2018) Scheduling multiple flexible projects with different variants of genetic algorithms. In: Proceedings of the 16th International Conference on project management and scheduling. Ed. by M. Caramia, L. Bianco, and S. Giordani, pp. 128–131

Holland JH (1975) Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. The University of Michigan Press, Ann Arbor

Kellenbrink C (2014) Ressourcenbeschränkte Projektplanung für flexible Projekte. Springer Fachmedien Wiesbaden, Wiesbaden

Kellenbrink C, Helber S (2015) Scheduling resource-constrained projects with a flexible project structure. Eur J Oper Res 246:379–391

Kellenbrink C, Helber S (2016) Quality- and profit-oriented scheduling of resourceconstrained projects with flexible project structure via a genetic algorithm. Eur J Ind Eng 10:574–595

Kolisch R, Sprecher A, Drexl A (1995) Characterization and generation of a general class of resource-constrained project scheduling problems. Manag Sci 41:1693–1703

Kolisch R (1996) Serial and parallel resource-constrained project scheduling methods revisited: theory and computation. Eur J Oper Res 90:320–333

Kolisch R, Padman R (2001) An integrated survey of deterministic project scheduling. Omega 29:249–272

Küpper H-U (1997) Controlling, 2nd edn. Schäffer-Poeschel, Stuttgart

Kurtulus I (1985) Multiproject scheduling: analysis of scheduling strategies under unequal delay penalties. J Oper Manag 5:291–307

Kuster J, Jannach D, Friedrich G (2009) Extending the RCPSP for modeling and solving disruption management problems. Appl Intell 31:234–253

Kuster J, Jannach D, Friedrich G (2010) Applying local rescheduling in response to schedule disruptions. Ann Oper Res 180:265–282

Li KY, Willis RJ (1992) An iterative scheduling technique for resource-constrained project scheduling. Eur J Oper Res 56:370–379

Lova A, Maroto C, Tormos P (2000) A multicriteria heuristic method to improve resource allocation in multiproject scheduling. Eur J Oper Res 127:408–424

Özdamar L, Ulusoy G (1995) A survey on the resource-constrained project scheduling problem. IIE Trans 27:574–586

Servranckx T, Vanhoucke M (2019a) A tabu search procedure for the resource-constrained project scheduling problem with alternative subgraphs. Eur J Oper Res 273:841–860

Servranckx T, Vanhoucke M (2019b) Strategies for project scheduling with alternative subgraphs under uncertainty: similar and dissimilar sets of schedules. Eur J Oper Res 279:38–53

Spears WM, de Jong KD (1991) On the virtues of parameterized uniform crossover. In: Proceedings of the Fourth International Conference on Genetic Algorithms. Ed. by R. K. Belew and L. B. Booker, pp. 130–136

Talbot FB (1982) Resource-constrained project scheduling with time-resource tradeoffs: the nonpreemptive case. Manag Sci 28:1197–1210

Tao S, Dong ZS (2017) Scheduling resource-constrained project problem with alternative activity chains. Comput Ind Eng 114:288–296

Tao S, Dong ZS (2018) Multi-mode resource-constrained project scheduling problem with alternative project structures. Comput Ind Eng 125:333–347

Tao S, Wu C, Sheng Z, Wang X (2018) Stochastic project scheduling with hierarchical alternatives. Appl Math Model 58:181–202

Tiwari V, Patterson JH, Mabert VA (2009) Scheduling projects with heterogeneous resources to meet time and quality objectives. Eur J Oper Res 193:780–790

Węglarz J, Józefowska J, Mika M, Waligóra G (2011) Project scheduling with finite or infinite number of activity processing modes—a survey. Eur J Oper Res 208:177–205