Computers and Operations Research 109 (2019) 230–249

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

Resource-constrained project scheduling with activity splitting and setup times

Mario Vanhoucke [a,b,c,*], José Coelho [a,d,e]

[a] Ghent University, Tweekerkenstraat 2, Gent 9000, Belgium
[b] Vlerick Business School, Reep 1, Gent 9000, Belgium
[c] UCL School of Management, University College London, 1 Canada Square, London E14 5AA, United Kingdom
[d] INESC - Technology and Science, Porto, Portugal
[e] Universidade Aberta, Rua da Escola Politécnica, 147, Lisbon, 1269-001, Portugal

ARTICLE INFO

Article history:
Received 31 August 2018
Revised 2 April 2019
Accepted 4 May 2019
Available online 9 May 2019

Keywords:
Project scheduling
RCPSP
Activity preemption
Setup times

ABSTRACT

This paper presents a new solution algorithm to solve the resource-constrained project scheduling problem with activity splitting and setup times. The option of splitting activities, known as activity preemption, has been studied in literature from various angles, and an overview of the main contributions will be given.

The solution algorithm makes use of a meta-heuristic search for the resource-constrained project scheduling problem (RCPSP) using network transformations to split activities in subparts. More precisely, the project network is split up such that all possible preemptive parts are incorporated into an extended network as so-called activity segments, and setup times are incorporated between the different activity segments. Due to the inherent complexity to solve the problem for such huge project networks, a solution approach is proposed that selects the appropriate activity segments and ignores the remaining segments using a boolean satisfiability problem solver, and afterwards schedules these projects to near-optimality with the renewable resource constraints.

The algorithm has been tested using a large computational experiment with five types of setup times. Moreover, an extension to the problem with overlaps between preemptive parts of activities has been proposed and it is shown that our algorithm can easily cope with this extension without changing it. Computational experiments show that activity preemption sometimes leads to makespan reductions without requiring a lot of splits in the activities. Moreover, is shown that the degree of these makespan reductions depends on the network and resource indicators of the project instance.

1. Introduction

Scheduling projects under a limited availability of resources have been studied for decades, and have resulted in numerous exact and meta-heuristic solution approaches for the problem under various extensions. The well-known resource-constrained project scheduling problem (RCPSP) has been one of the most studied problem in this area, and has been used as the basic problem type for extensions to more realistic settings. In this basic problem type, all non-preemptive activities of the project must be scheduled within the precedence constraints and limited resource availabilities, such that the total project makespan is minimized. In a paper written by Hartmann and Briskorn (2010), a survey is given of variants and extensions of this basic problem type, and one of these variants is used as the foundation of this current research study. More precisely, this paper studies the resource-constrained project scheduling problem with the possibility of preempting each project activities in integer parts, and within the presence of five types of setup times between the preemptive subparts of the activities. This preemptive resource-constrained project scheduling problem with setup times (abbreviated as PRCPSP-ST) can be represented as problem $m, 1T|cpm, pmtn-st|C_{\max}$ using the classification scheme of Herroelen et al. (1999) or as $PS|prec, pmtn-st|C_{\max}$ following the classification scheme of Brucker et al. (1999). Note that the abbreviation $pmtn-st$ has been added to refer to preemption with setup times (not originally incorporated in the classification schemes).

The paper presents a meta-heuristic solution approach that makes use of an extended project network that contains the subparts of the preemptive activities as separate activity segments between which setup times exist. The solution approach to solve the problems is based on an extended version of the algorithm of

* Corresponding author.
E-mail address: mario.vanhoucke@ugent.be (M. Vanhoucke).

https://doi.org/10.1016/j.cor.2019.05.004
0305-0548/

Coelho and Vanhoucke (2011) that relies on a boolean satisfiability problem solver to select the preemptive activity parts of the activities and a genetic algorithm to schedule these activity segments aiming at minimizing the total project makespan. The contribution of this paper is threefold. First, it is - to the best of our knowledge - the first paper that incorporates up to five types of setup times in a resource-constrained project scheduling setting. Second, a new procedure that iteratively switches between a logical boolean optimizer and a fast and efficient scheduling algorithm is proposed to solve a project using the previously mentioned activity segments. We show that by using the activity segment concept, the preemptive RCPSP with setup times can be solved efficiently on a large set of project instances. Finally, the paper investigates the impact of network and resource parameters on the makespan reduction under various settings for the setup times.

The outline of the paper can be summarized along the following lines. Section 2 provides an overview of the literature on the use of activity preemption and setup times for resource-constrained project scheduling problems. This section also explains the design of an extended project network and the selection parts of this network in order to model activity preemption with setup times. In Section 3, the algorithm to solve the problem under study is explained in detail. Moreover, the algorithmic details to schedule the (preempted) activities in order to obtain a project schedule with a minimum makespan are explained and illustrated on an artificial project example. Section 4 presents the results of a computational experiment on four different datasets from literature, and Section 5 draws general conclusions and highlight avenues for future research.

2. Problem formulation

A resource-constrained project is normally modelled as an activity-on-the-node network in which a set of activities N is represented by the network nodes and the set A is used to refer to the set of pairs of activities between which a finish-start precedence relationship with time lag 0 exists. We assume graph $G(N, A)$ to be acyclic. The execution of the project activities requires the use of a set R of renewable resource types. Each activity i has a deterministic duration d_i and requires r_{ik} units of resource type $k = 1, \ldots |R|$, which has a constant availability a_k throughout the project horizon. We assume that $r_{ik} \leq a_k,\ i \in N,\ k \in R$. Each activity can be preempted, and it is assumed that this preemption can only occur at integer points, i.e. an activity can have maximum d_i integer parts with a minimum duration equal to 1. In reality, activities are often split at random (fractional) points. However, by rescaling activity durations (e.g. from integer weeks to integer days), the fractional setup times for weeks become integer values for days. Hence, the algorithm can be extended to fractional activity preemption, but this will lead not only to much bigger activity networks, but sometimes to more significant changes. More precisely, in some cases, such a transformation will be easy (e.g. a seventh of a week is a day) but in other cases it may be necessary to change the numerical basis to scale certain rational numbers (e.g. a third of a day is 0.33... days which can not be modelled as an integer number). In Section 2.2, the design of the activity network for preemptive activities will be discussed in detail. Furthermore, it is assumed that each activity preemption requires a setup time t^s. The values of t^s will vary according to five different underlying assumptions of calculating the setup times as will be discussed later in this paper. Furthermore, it is assumed that the activity requires the same set of resources with the same resource demand r_{ik} during its extra setup time. The project network can be extended with a dummy start activity and a dummy end activity, both with zero duration and no renewable resource demand, while the other activities have a non-zero duration and a non-negative resource requirement. A schedule S is defined by a vector of start times which implies a vector of finish times for each activity of the project. Due to the allowance of activity preemption, each activity can be split in multiple parts, and each part must get a start and finish time. A schedule is said to be feasible if the precedence and resource constraints are satisfied. The objective of the PRCPSP-ST is to find a feasible schedule such that the schedule makespan is minimised.

The outline of this section is as follows. Section 2.1 provides a short literature overview on the use of activity preemption and setup times in resource-constrained project scheduling. In Section 2.2, the design of an extended network using activity segments is outlined. In Section 2.3, the way the setup times are modelled in the extended project is explained. Finally, Section 2.4 presents a mathematical problem formulation given the concepts discussed.

2.1. Literature overview

In a recent survey on extensions on the resource-constrained project scheduling problem (Hartmann and Briskorn, 2010), both activity preemption and setup times are mentioned and discussed. This section gives a summary of the main contributions in literature on these two extensions for resource constrained project scheduling, and consists of three parts. First, it gives an overview of the existing literature of activity preemption for single- and multi-mode resource-constrained project scheduling problems. Secondly, the presence of various uses of setup times for these problem types in literature is discussed. Finally, the design of the project network to solve these problems is reviewed.

Activity preemption Various algorithms to solve the single-mode RCPSP with activity preemption have been proposed in the literature. Demeulemeester and Herroelen (1996a) present a depth-first branch-and-bound procedure to solve the preemptive RCPSP and conclude that activity preemption has little effect on the project duration. A 0/1 integer programming model has been presented by Nudtasomboon and Randhawa (1997) who solve the problem under three different objectives. Bianco et al. (1999) present a formulation with coloring techniques, and show that an exact weighted coloring solution is an optimal preemptive schedule. An LP based algorithm is proposed by Damay et al. (2007). Ballestin et al. (2008) show that the basic element of meta-heuristic procedures for solving the RCPSP can be adapted to cope with activity preemption. The authors have tested their solution approach on the J30 and J120 instances of the PSPLIB (Kolisch et al., 1995) and confirm the claim that activity preemption adds little value to reduce the project duration. Finally, Vanhoucke (2008) and Vanhoucke and Debels (2008) allow preemption as well as overlaps between the preemptive parts of the activities and measure the impact on the project duration and resource utilization. Extensions to the multi-mode version of the RCPSP (MMRCPSP) have been proposed by Buddhakulsomsiri and Kim (2007, 2006) and Van Peteghem and Vanhoucke (2010).

Activity setup times Setup times have been incorporated in the RCPSP models by various authors. Kaplan (1991) was the first to incorporate setup times in project scheduling and has assumed that a setup time is only required between activity interruptions and not for the initial start-up of an activity. In our study, we follow this logic and add setup times to preemptive activities as a way to penalize activity splits. More precisely, each split in an activity will result in an additional setup time, except for the first part of the activity. Mika et al. (2006) have written a summary paper that gives a full overview of how to model setup times for project scheduling problems. They show that setup times can be modelled in a variety of ways, and they refer to the difference between sequence-dependent, sequence-independent and schedule-dependent setup times. We suggest any researcher interested in

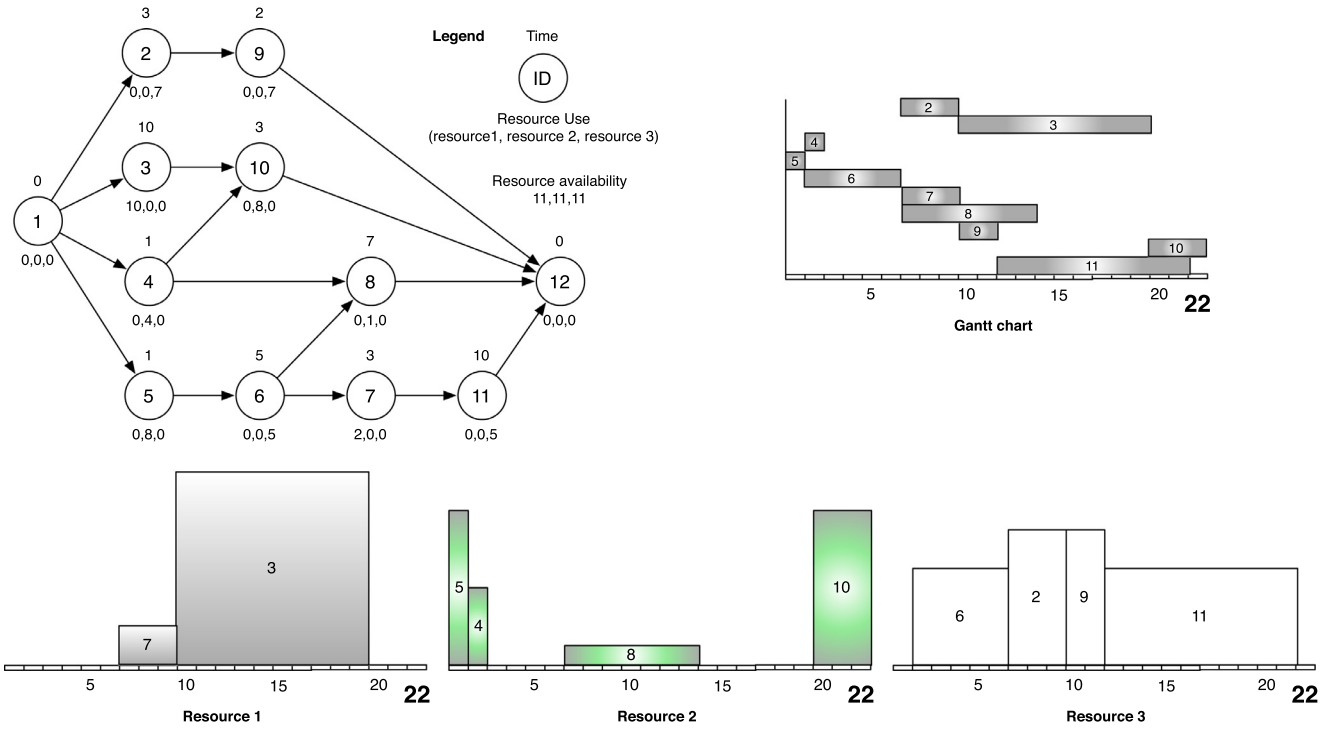

Fig. 1. An example project network and an optimal schedule of 22 days.

modelling setup times in resource-constrained project scheduling to start with this overview paper. In Mika et al. (2008), the same authors incorporate schedule-dependent setup times and develop a tabu-search to solve this problem. Vanhoucke (2008) adds setup times between preemptive activities and allow these parts to overlap. A slightly different approach is taken by Krüger and Scholl (2009) and Krüger and Scholl (2010) who model setup times as changeover times for resources in projects or between multiple projects. In their problem, the transfer times between resources in a multi-project environment can be considered as a special but very realistic case of setup times. Other examples of using setup times in resource-constrained scheduling are given in the process industry using batch scheduling (Demeulemeester and Herroelen, 1996b; Schwindt and Trautmann, 2000).

Network design In order to generate a feasible schedule, the technique of transforming project networks in extended networks by splitting the activities in preemptive subparts has been used in literature for solving the preemptive RCPSP, leading to extended activity networks with a much higher number of project activities. Demeulemeester and Herroelen (1996a) have translated the RCPSP to the PRCPSP by means of a subactivity project network G(N0,A0) and developed a branch-and- bound procedure to optimally solve the problem without the use of setup times. In a subactivity network, each activity i is split up into d_i subactivities i_s $(s = 1, \ldots, d_i)$ with each subactivity duration $d_{i_s} = 1$ and a corresponding resource requirement $r_{i_s k} = r_{ik}$. This approach has also be followed by Vanhoucke and Debels (2008) who used these extended networks to solve the RCPSP problem with preemption as well as fast tracking possibilities. This problem assumes that the preemptive parts of the activities can be scheduled in parallel, hereby allowing overlaps between the activity splits to reduce the project durations. In Vanhoucke (2008), the same extended network approach has been used in a branch-and-bound procedure, this time adding setup times between the various subactivities of the extended project network.

In the current paper under study, the preemptive resource-constrained project scheduling problem (PRCPSP) is studied within the presence of setup times between the preemptive parts of the activities. The traditional transformation of the original project network to an extended network by splitting each activity in subparts can also be used to solve the problem under study. However, in the current study, a slightly different network transformation approach will be followed compared to the traditional approach used in literature, making use of activity segments. The extended network is then subject to a metaheuristic search as described in Section 3 to schedule the project activities with activity preemption and setup times.

In the remaining sections, both the existing network transformations from literature and our activity segment network design is discussed and illustrated on a small artificial project network. Fig. 1 displays an activity-on-the-node project network with 10 non-dummy activities taken from the DC_1 dataset (instance *mv25.rcp*) (Vanhoucke, 2006). The resource availability is equal to 11 for each renewable resource. The schedule is shown in a Gantt chart which shows that the minimal project duration is 22 days when no activity splitting is allowed. The resource profiles are shown at the bottom of the figure for each of the three resources. This example project will be used in this paper to illustrate the impact of activity preemption with setup times on the total project duration.

2.2. Network design

In order to explain the network transformation to design the extended network, each activity will be cut into smaller pieces using the concept of *activity segments* as discussed in the next section. These activity segments will then be used to extend the original network into an extended network containing a set of activity segments and relations between these segments. The

construction of such a network can be done in two fundamentally different ways, as will be discussed below.

2.2.1. Activity segments

In order to define preemptive parts of an activity i with a duration d_i, the concept of an *activity segment* is used to refer to any possible integer sub-part of each project activity. Each activity segment is a result of the original activity cut in pieces at a certain integer point in time, and has a duration which is lower than or equal to the original activity duration d_i. These activity segments for activity i are labelled as $S_x^i D_y$ ($x = 1, \ldots, d_i$ and $y = 1, \ldots, d_i - x + 1$) to clarify how these segments are generated. The first part, abbreviated by an S for segment, is used to refer to the point where the original activity is cut. The second part is abbreviated by a D and is used to refer to the duration of the activity segment. More precisely, the cutting point S_x refers to the point x at which the activity is cut into two pieces. Cutting points can occur at points $x = 1, \ldots, d_i$, and after cutting the activity into two parts, the second part will then be used to generate the activity segments (note that $x = 1$ means that the cut is done at the beginning of the activity, i.e. no real cut is executed and the activity's second piece is equal to the original activity). Obviously, the remaining duration of the second piece of the activity depends on the value of x. Enumerating over all possible durations of this second piece (between its minimum 1 and maximum value $d_i - x + 1$) results in a set of activity segments. Each of these activity segments has therefore a duration y as denoted by the D_y part of the activity segment $S_x^i D_y$. Note that in the remainder of this manuscript, we refer to any activity segment of activity i as $S_x^i D_y$ as stated before, unless we use the activity segment concept without explicitly referring to a specific activity. In that case, we will not mention the index of the activity and refer to the segment as $S_x D_y$ instead of $S_x^i D_y$.

Consequently, an activity segment $S_x^i D_y$ is any integer part of an activity with duration d_i for which $x \in \{1, \ldots, d_i\}$ and $y \in \{1, \ldots, d_i - x + 1\}$. These segments can be easily generated by enumerating over the activity's duration, cut the activity at each such integer point into two pieces and enumerate over all possible durations of the second piece. The number of activity segments for an activity with a duration d_i is equal to $\frac{d_i \cdot (d_i+1)}{2}$. Fig. 2 displays all 10 activity segments of an activity with an original duration of 4 time units, resulting in $\frac{4 \cdot 5}{2} = 10$ activity segments. As explained, these activity segments are labelled as $S_x D_y$ ($x = 1, \ldots, 4$, $y = 1, \ldots, 4 - x + 1$ and the index i is removed from $S_x^i D_y$ in the figure), with S_x the point at which the activity is cut into a smaller part (cutting points occur at points $x = 1, \ldots, 4$) and D_y the duration of the segment. These activity segments will be used to construct the new extended project network, by replacing each original activity by all its segments, as will be discussed hereafter.

2.2.2. Segment chains

The most straightforward approach to extend the network is to enumerate all possible feasible combinations of the different $S_x^i D_y$ activity segments. Each combination consists of a set of activity segments with the sum of their y-values equal to the original duration d_i. Moreover, precedence relations must be added between the activity segments of each such combination, and must respect the logic $x_1 + y_1 = x_2$ for any two consecutive activity segments $S_{x_1}^i D_{y_1}$ and $S_{x_2}^i D_{y_2}$ of a combination. In the remainder of this text, we refer to such a feasible combination of activity segments as a *segment chain*, and $2^{d_i - 1}$ of such segment chains can be found for any activity with a duration equal to d_i.

After enumeration of all possible feasible combinations, each segment chain must then be added to the network as a possible way of splitting the activity. As an illustration, the left picture in Fig. 3 shows that 8 feasible segment chains exists by combining the different activity segments for an activity with a duration of 4

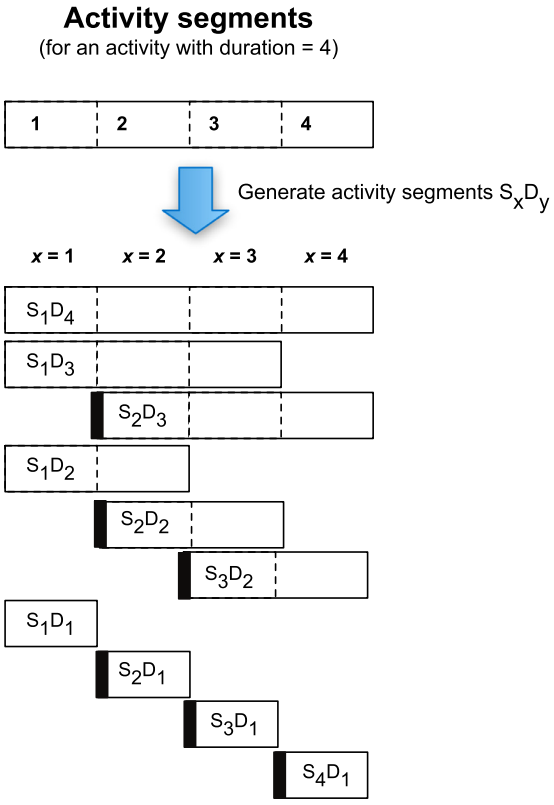

Fig. 2. 10 activity segments for an activity i with $d_i = 4$ (index i removed from $S_x^i D_y$).

time units. The sum of the durations of all segments of each combination is equal to the original activity duration (4 time units). Since this paper makes use of setup times each time an activity is interrupted, extra time must be added at the start of each segment, except for the first activity segment. This extra setup time is shown by the black bars in the figure, and the size of this additional time will vary along the setup time assumptions which will be discussed in Section 2.3.

Each segment chain in the left picture of Fig. 3 consists of a sequence of activity segments, and each activity segment can now be represented by a node in the project network (the index i is removed from $S_x^i D_y$ in the figure). Only one such segment chain should be selected to represent the planning of the original activity with possible preemptions and setup times. In the case of the 4 time units activity, this will result in a network with 20 nodes (activity segments) to represent the 8 segment chains.

2.2.3. Unique activity segments

For project networks with a large number of activities with long activity durations, the number of activity segments can grow quickly, and therefore the number of feasible segments chains will grow quickly too, leading to huge project networks. In order to avoid this huge increase in the number of nodes (represented by the number of activity segments) in the extended network, we have chosen to restrict the number of nodes as much as possible by only adding the unique activity segments. Each activity segment $S_x^i D_y$ will be added exactly once in the network, leading to only $\frac{d_i(d_i+1)}{2}$ nodes for each activity. The right picture of Fig. 3 shows that each $S_x^i D_y$ is displayed exactly once, resulting in $\frac{4 \cdot 5}{2} = 10$ nodes in the network. Compared to the left picture of the figure, all the duplicated activity segments have been removed, resulting in a reduction of 50% of nodes in the project network.

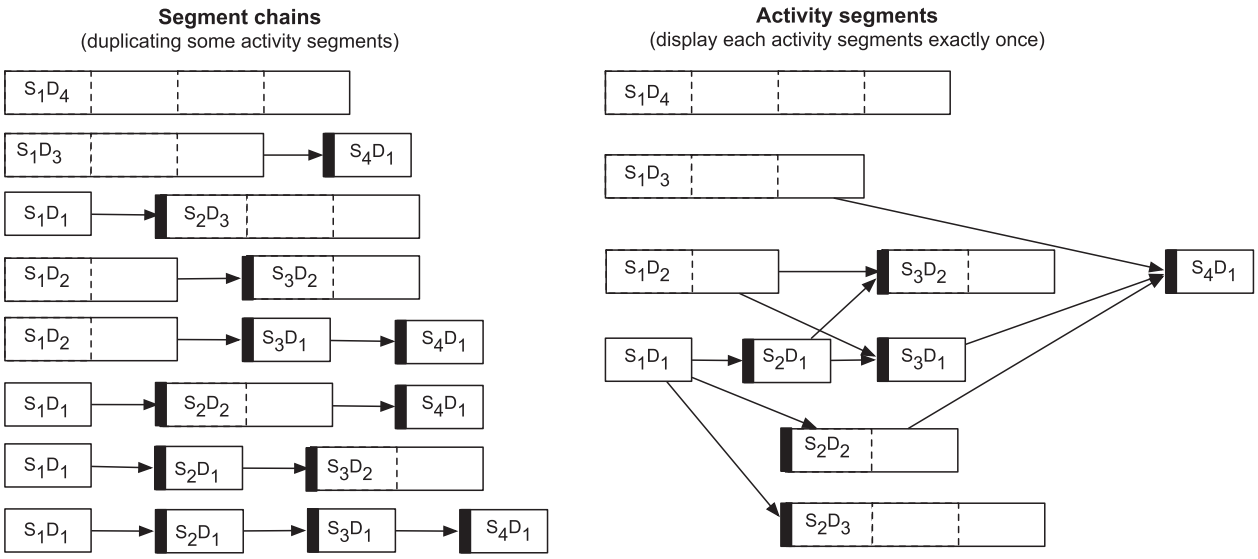

Fig. 3. Network design: Two approaches for combining activity segments for an activity i with $d_i = 4$ (index i removed from $S_x^i D_y$).

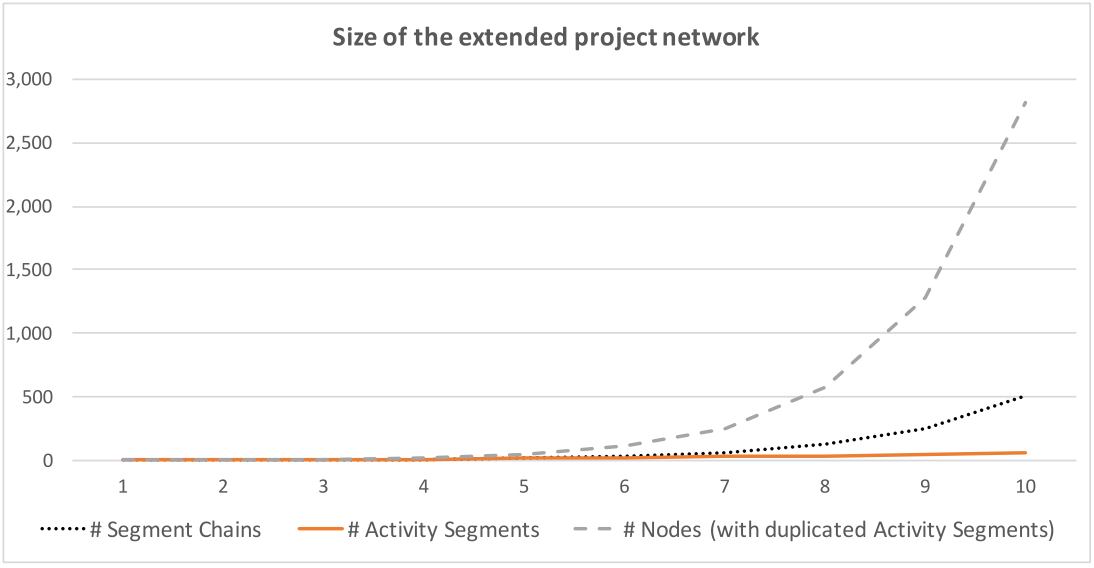

Fig. 4. The number of activity segments for an activity with increasing duration.

In order to assure that all the segment chains (8 in the example) can still be obtained by the restricted set of nodes of Fig. 2, precedence relations should be added between some of the $S_x^i D_y$ segments. More precisely, a precedence relation has to be added between every couple of parts $S_{x_1}^i D_{y_1}$ and $S_{x_2}^i D_{y_2}$ for which $x_2 = x_1 + y_1$. The right picture of Fig. 3 now shows that the extended network contains a lower number of nodes (10) compared to the previous approach (20), but the selection of a feasible segment chain is now defined as the selection of a set of nodes linked to each other. This selection of linked activity segments will be discussed in Section 3.

As a summary, using unique activity segments will lead to project networks with much fewer nodes than when segment chains are used, and this becomes particularly important for activities with large durations. As an example, Fig. 4 shows the number of activity segments for an activity along an increasing duration value (x-axis) (to a maximum duration of 10 integer units). The number of activity segments grows to 55 for an activity with duration equal to 10. However, the number of total activity segments grows to an astonishing 2816 due to the duplications, resulting in 512 segment chains. Hence, using the "segment chains" approach results in a larger extended network since activity segments will be duplicated. However, in the "unique activity segments" approach, activity segments are not duplicated and each activity segment represents exactly one single node. When the duration of the activity grows even further beyond 10, the size expands dramatically. As an example, for an activity with a duration of 20 integer units, the total number of activity segments is equal to 210. This would lead to 524,288 segment chains, and in 5,505,024 nodes in the network for this activity only. However, when the unique activity segments approach is used, it only requires 210 nodes.

2.3. Setup times

This section discusses five versions for the setup times between the preemptive parts of the activities (Section 2.3.1) and shows that some of the activity segments can be removed from the network, depending on the type and size of the setup times (Section 2.3.2).

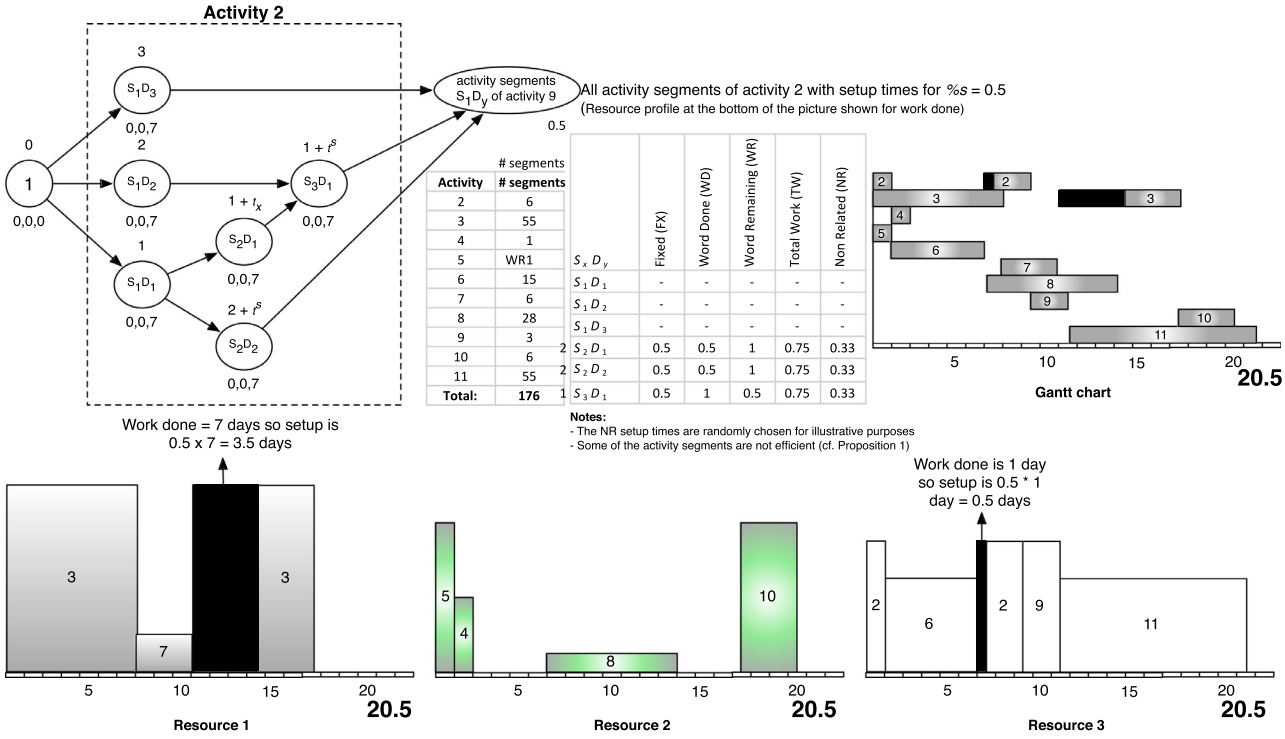

Fig. 5. Time decrease from 22 to 20.5 days with setup times (work done and $\%s = 0.5$).

2.3.1. Setup types

In this paper, five versions of setup times are modelled as the extra time needed to start the work on an activity as the result of premature interruption of parts of the activity. As shown by the black bars in Figs. 2 and 3, no setup time is required for the first part of the activity, but only after each activity interruption. In the remainder of this manuscript, a setup time for activity i is defined as $t_i^s(type, \%s, x)$ with $type \in \{NR, FX, TW, WD, WR\}$, $\%s$ the fraction of the unit duration d_i between 0 and 1, and x the point between 1 and d_i for activity segment $S_x^i D_y$, as will now be explained. First, the value for each setup time for activity i (expressed in time units) is calculated as a fraction $\%s$ ($0 \leq \%s < 1$) of its unit duration d_i. Second, the calculation of the setup time also depends on which type of setup time calculation is chosen. More specifically, the setup times will be modelled in five different ways, as random setup times unrelated to the activity duration (NR), as fixed setup times (FX), as values depending on the total activity duration (TW) or as a function of the work already done (WD) and/or the remaining work yet to be done (WR). Finally, the value of the setup time also depends on the specific activity segment $S_x^i D_y$ of activity i, i.e. at which point x the activity i is interrupted.

Non related (NR): This is probably the simplest base case where the setup times would only depend on the activity, independently of its duration. In this case, setup times are not expressed as a fraction of the unit duration, but are a randomly generated number independent of the activity duration. In order to generate duration independent setup times, scaled by the $\%s$ fraction to control its maximum value, the following formula is used:

$$t_i^s(NR, \%s, x) = \begin{cases} 0 & \text{if } x = 1 \\ \text{RAND}(0, d_i - 1) \cdot \%s & \text{if } x > 1 \end{cases}.$$

Fixed setup (FX): The setup time is a fixed value and is therefore independent of where the interruption took place during the activity duration. The setup time is then equal to a fixed value $\%s$, independent of the activity duration, as shown in the equation

$$t_i^s(FX, \%s, x) = \begin{cases} 0 & \text{if } x = 1 \\ \%s & \text{if } x > 1 \end{cases}.$$

Total Work (TW): The setup time is a fixed value that depends on the duration d_i of the activity, as follows $t_i^s(TW, \%s, x) = \begin{cases} 0 & \text{if } x = 1 \\ \frac{\%s}{2} \cdot d_i & \text{if } x > 1 \end{cases}$. Similar to the NR and FX setup times, the TW setup times are fixed values, independent on where the interruption took place. But while the FX setup times are one and the same value for each project activity, the TW setup time values increase as the activity duration increases. Note that the TW setup time value is calculated using $\frac{\%s}{2}$ instead of $\%s$ to make sure that the average value of the setup times is equal to the average value for the WD and WR setup times, as discussed hereafter.

Work done (WD): the setup time depends on the work already done. Early splits lead to lower setup times than splits late in the activity progress, as shown by the equation $t_i^s(WD, \%s, x) = \begin{cases} 0 & \text{if } x = 1 \\ \%s \cdot (x - 1) & \text{if } x > 1 \end{cases}$, which can be simplified to $t_i^s(WD, \%s, x) = \%s \cdot (x - 1)$.

Work remaining (WR): The setup time depends on the remaining work yet to be done. Early splits therefore lead to higher setup times than late splits, and therefore, it can be modelled as $t_i^s(WR, \%s, x) = \begin{cases} 0 & \text{if } x = 1 \\ \%s \cdot (d_i + 1 - x) & \text{if } x > 1 \end{cases}$.

Fig. 5 makes use of the example project of Fig. 1 to display the extended project network with activity segments. The subnetwork on top of the picture shows all 6 activity segments of activity 2 and the relations between these activity segments and with the predecessor activity 1 and successor activity 9. The figure also shows a table with the number of activity segments for all other activities, resulting in an extended project network with 176 activity segments. The setup times are displayed for activity 2 under the five different versions (fixed, total work, work done, work remaining and non related) using a $\%s$ value equal to 0.5. The black blocks in the resource profiles represent setup times for the split activities 2 and 3 equal to 0.5 and 3.5 days, respectively. These setup times can be calculated using the formulas described earlier with

a *%s* value equal to 0.5 and are shown for the work done version of the setup times calculations. The resource profiles shown at the bottom of the picture clearly illustrate that despite the huge setup time when splitting activity 3 (equal to 0.5 times the 7 days of the work already done) still result in a decrease of the project makespan from 22 days (cf. Fig. 1) to 20.5 days.

Although the primary goal of this paper is to include these five classes of setup times in a scheduling procedure, rather than to illustrate their practical value, we have nevertheless taken the time to share experiences with a small group of 35 project managers. During a workshop with project managers from different companies in Belgium during a project management training at Vlerick Business School (Belgium), a discussion was held about the importance of incorporating task complexity, resource skills and resource learning and forgetting into the construction of a project baseline schedule. While many people recognised the importance of these concepts, they quickly came up with the idea of changing the activity durations a bit (e.g. increase them for more complex tasks, or when highly skilled resources are required) as a quick-and-easy solution to implicitly take these concepts into account. When we turned the discussion into the direction of activity interruptions, most people continued to choose for the pragmatic approach to add additional time depending on the circumstances. We classified the setup types into different classes and asked for examples from their experience, for which a summary is given in Table 1. This table is only illustrative and displays some generic cases for each type of setup time, but a detailed study to the practical application of the different types of setup times is outside the scope of this paper.

Table 1
Generic case for using each type of setup time.

Type	Example
NR	A setup time might depend only on the activity and not on its duration. This is used to model the case where setup times depend on the cognitive load of the task, independently of its duration.
FX	Activity interruptions are avoided as much as possible. However, when they occur, the project manager organizes a team meeting to catch up with the work and to communicate the reactivation with the client. This meeting normally takes half a day.
TW	This type of setup time is considered as the default type. It assumes that the setup time increases along the "size" or "complexity" of the task. This size or task complexity is measured by longer durations and use of more resources (i.e. work content).
WD	This type is mostly used for routine task that do not require a lot of expertise. When activities are interrupted, a new team of people (different than the original team) are assigned to the activity after interruption. However, the activity can only be re-started after a so-called "handover meeting" in which the status of the progress is reported by the old team 1 (who worked on the first part of the activity) to the new team 2 (who will work at the remaining part). The more work is done by team 1, the more time it takes to handover all the work to provide a reliable status/progress report to the second team.
WR	This type is mostly used for complex tasks that involve some preparatory work to install auxiliary resources. This preparatory time is included in the original activity duration, and some of these auxiliary resources are released along the progress of the activity, while the ones that are still necessary stay at the activity. However, after activity interruption, all auxiliary resources are often released (and used elsewhere), and upon re-activation of the activity, these resources must be installed again for the remaining work. The more work remaining, the longer this installation time, as captured by the setup time.

Note: It should be noted that the use of different setup times must not always reflect practical settings of the project, but can also be used to steer the construction of a preemptive project schedule in a desired direction. As an example, when the project scheduler wishes to avoid interruptions in the early stages of the activities, WR setup times can be used to penalize early activity preemptions, and favour preemptions that occur when most of the work of the activity is done.

2.3.2. Network reduction

In order to decrease the size of the activity network, some of the $S_x^i D_y$ segments ($x > 1$) defined in Section 2.2 can be easily removed from the network without excluding any optimal combination of activity preemptions. The removal of any *inefficient activity segment* $S_x^i D_y$ is done when its total duration, including its setup time is equal to or larger than the duration of a non-preemptive segment without setup times $S_1^i D_{x+y-1}$ for that activity, as stated in the following proposition.

Proposition 1. *Inefficient activity segment An activity segment* $S_x^i D_y$ *with* $x > 1$ *and* $t_i^s(type, \%s, x) \geq x - 1$ *is inefficient and can be removed from the activity network*

Proof. Consider an activity with a duration d_i as shown in the left picture of Fig. 6. Assume that setup times are set equal to zero, i.e. $t_i^s(type, \%s, x) = 0$. Call $S_x^i D_y$ any preempted part of this activity cut at point $x > 1$ with a duration y (with $x = 2, \ldots, d_i$ and $y = 1, \ldots, d_i - x + 1$). Call the set $SC_{i,x,y}$ the set of all segment chains of activity i that includes activity segment $S_x^i D_y$. By definition, the point x of activity segment $S_x^i D_y$ is the point at which the activity is preempted. The sum of the durations of the activity segments of any segment chain of $SC_{i,x,y}$ that preceeds activity segment $S_x^i D_y$ is equal to $x - 1$, and consequently, the duration of all these activity segments of this segment chain plus the duration of activity segment $S_x^i D_y$ is equal to $x - 1 + y$ (cf. middle picture of Fig. 6). Obviously, this duration is equal to the duration of the non-preempted activity segment $S_1^i D_{x+y-1}$ if the setup time $t_i^s(type, \%s, x)$ is equal to zero.

Now assume a setup time $t_i^s(type, \%s, x) > 0$ for any activity segment $S_x^i D_y$ for $x > 1$. The activity segment $S_x^i D_y$ has now a total duration of $y + t_i^s(type, \%s, x)$ (i.e. including the setup time, as shown in the right picture of Fig. 6). The non-preempted activity segment $S_1^i D_{x+y-1}$ has a total duration of $x + y - 1$ since no setup time is incurred. Hence, if the total duration of activity segment $S_x^i D_y$ is at least as large as the total duration of the non-preempted activity segment $S_1^i D_{x+y-1}$, then activity segment $S_x^i D_y$ is dominated by activity segment $S_1^i D_{x+y-1}$. Hence, when $y + t_i^s(type, \%s, x) \geq x + y - 1$ it is never beneficial to add activity segment $S_x^i D_y$ in the network, and therefore, activity segment $S_x^i D_y$ is said to be inefficient if $t_i^s(type, \%s, x) \geq x - 1$ for any value $x > 1$. □

From proposition 1, three network reduction rules can be defined. Recall that each setup time $t_i^s(type, \%s, x)$ is defined by its

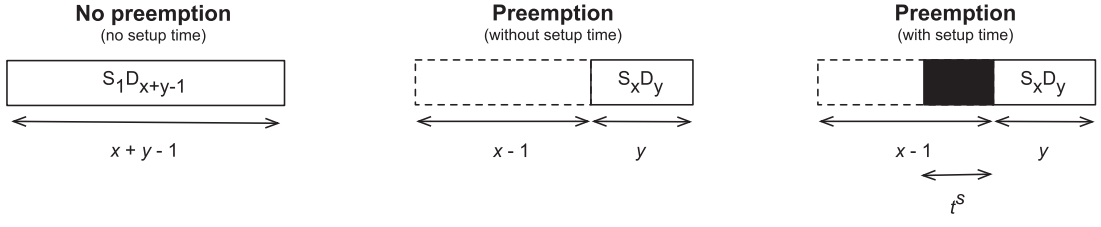

Fig. 6. Calculating inefficient activity segments (index i removed from $S_x D_y$).

Table 2
Total durations of all $S_x^i D_y$ segments for an activity with $d_i = 9$ and $t_i^s(WR, 0.6, x)$.

	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9
S_1	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0
S_2	**5.8**	**6.8**	**7.8**	**8.8**	**9.8**	**10.8**	**11.8**	**12.8**	
S_3	**5.2**	**6.2**	**7.2**	**8.2**	**9.2**	**10.2**	**11.2**		
S_4	**4.6**	**5.6**	**6.6**	**7.6**	**8.6**	**9.6**			
S_5	4.0	5.0	6.0	7.0	8.0				
S_6	3.4	4.4	5.4	6.4					
S_7	2.8	3.8	4.8						
S_8	2.2	3.2							
S_9	1.6								

type ∈ {NR, FX, TW, WD, WR}, its value $\%s < 1$ and its point x of the activity segment $S_x^i D_y$ (cf. Section 2.3).

Network reduction rule 1 (*work remaining*). *Any segment $S_x^i D_y$ with $x > 1$ and $x \leq \frac{\%s \cdot (1+d_i)+1}{1+\%s}$ of an activity with duration d_i and $t_i^s(WR, \%s, x)$ can be removed from the activity network.*

The condition of Proposition 1 states that an activity segment can be removed if $t_i^s(WR, \%s, x) \geq x - 1$. Since $t_i^s(WR, \%s, x) = \%s \cdot (d_i + 1 - x)$ for $x > 1$ for work remaining setup time, it follows from proposition 1 that $\%s \geq \frac{x-1}{d_i+1-x}$ or $x \leq \frac{\%s \cdot (1+d_i)+1}{1+\%s}$.

Network reduction rule 2 (*total work*). *Any segment $S_x^i D_y$ with $x > 1$ and $x \leq (1 + \%s \cdot d_i)$ of an activity with duration d_i and $t_i^s(TW, \%s, x)$ can be removed from the activity network.*

The condition of Proposition 1 states that an activity segment can be removed if $t_i^s(TW, \%s, x) \geq x - 1$. Since $t_i^s(TW, \%s, x) = \frac{\%s}{2} \cdot d_i$ for $x > 1$ for total work setup time, it follows from proposition 1 that $\%s \geq 2 \cdot \frac{x-1}{d_i}$ or $x \leq (1 + \frac{\%s}{2} \cdot d_i)$.

Network reduction rule 3 (*non related*). *Any segment $S_x^i D_y$ with $x > 1$ and $x \leq (1 + \%s \cdot \text{rand}(0, d_i - 1))$ of an activity with duration d_i and $t_i^s(NR, \%s, x)$ can be removed from the activity network.*

The condition of Proposition 1 states that an activity segment can be removed if $t_i^s(NR, \%s, x) \geq x - 1$. Since $t_i^s(NR, \%s, x) = \%s \cdot \text{rand}(0, d_i - 1)$ for $x > 1$ for total work setup time, it follows from proposition 1 that $\%s \geq \frac{x-1}{\text{rand}(0, d_i-1)}$ or $x \leq (1 + \%s \cdot \text{rand}(0, d_i - 1))$.

Note that no network reduction rules for $t_i^s(FX, \%s, x)$ and $t_i^s(WD, \%s, x)$ are given since the condition of proposition 1 will never be satisfied for any given value of $x > 1$ and $\%s < 1$.

Consider as an example an activity with $d_i = 9$ and setup time $t_i^s(WR, 0.6, x)$ (i.e. $\%s = 0.6$ and work remaining), all segments with $x > 1$ and $x \leq 4.38$ can be removed from the network. This is illustrated in Table 2 where all $S_x^i D_y$ values indicated in bold are dominated by the $S_1^i D_{x+y-1}$ values for $x = \{2, 3, 4\}$. As an example, $S_2^i D_1$ is dominated by $S_1^i D_2$, $S_3^i D_1$ by $S_1^i D_3$, $S_3^i D_2$ by $S_1^i D_4$ and so on.

2.4. Mathematical problem formulation

In this section, a mathematical problem formulation is presented based on the network design using unique activity segment and setup calculations presented in the previous sections. Recall that a project is represented by a topological ordered activity-on-the-node format where A is the set of pairs of activities between which a finish-start precedence relationship with time lag 0 exists and N is the set of nodes that represent the activities. We assume that the project has n non-dummy activities and a dummy end activity $n + 1$ with a zero duration and resource use. Furthermore, each activity i has a duration d_i and can be split into activity segments $S_x^i D_y$ with $x = 1, \ldots, d_i$ and $y = 1, \ldots, d_i - x + 1$. The setup times between the activity segments are denoted by $t_i^s(type, \%s, x)$ as shown in Section 2.3.1 with *type* ∈ {NR,FX,TW,WD,WR} and $\%s < 1$. In the formulation, we will abbreviate this setup time to $t_i^s(x)$ for ease of reading (but of course, these setup times still depend on user input values for *type* and $\%s$). Each activity has a renewable resource demand r_{ik} for each renewable resource $k \in R$ with availability a_k. Furthermore, the parameter T is an upper bound on the project makespan which can easily be calculated using fast and efficient priority rules. The abbreviations es_i and lf_i are used to denote the earliest start and latest finish for activity i given the project upper bound T using traditional forward and backward critical path calculations.

The decision variable $w_{i,\,x,\,y,\,t}$ is equal to 1 if activity segment $S_x^i D_y$ of activity i starts at time instance t, and 0 otherwise. A schedule is defined by a vector of start times for each activity segment and is said to be feasible if all precedence and renewable and non-renewable resource constraints are satisfied. The problem under study can be mathematically formulated as follows:

$$\text{Minimize} \sum_{t=es_{n+1}}^{lf_{n+1}} t \cdot w_{n+1,1,0,t} \tag{1}$$

subject to

$$\sum_{t=es_i}^{lf_i} \sum_{x=1}^{d_i} (t + d_i - x + 1 + t_i^s(x)) \cdot w_{i,x,d_i-x+1,t} \leq \sum_{y=0}^{d_j} \sum_{t=es_j}^{lf_j} t \cdot w_{j,1,y,t} \qquad \forall (i,j) \in A \tag{2}$$

$$\sum_{t=es_i}^{lf_i} (t + y_1 + t_i^s(x)) \cdot w_{i,x,y_1,t} \leq \sum_{t=es_i}^{lf_i} t \cdot w_{i,x+y_1,y_2,t} \qquad \forall i \in N \quad x = 1, \ldots, d_i - 1 \quad y_1 = 1, \ldots, d_i - x \quad y_2 = 1, \ldots, d_i - x - y_1 + 1 \tag{3}$$

$$\sum_{y=1}^{d_i} \sum_{x=\max(c-y+1,1)}^{\min(d_i-y+1,c)} \sum_{t=es_i}^{lf_i} w_{i,x,y,t} = 1 \qquad \forall i \in N; c = 1, \ldots, d_i \tag{4}$$

$$\sum_{t=es_{n+1}}^{lf_{n+1}} w_{n+1,1,0,t} = 1 \tag{5}$$

$$\sum_{i=1}^{n} \sum_{x=1}^{d_i} \sum_{y=1}^{d_i-x+1} \sum_{s=max(t-y-t_i^s(x),es_i)}^{min(t-1,lf_i)} r_{i,k} \cdot w_{i,x,y,s} \leq a_k \qquad \forall k \in R \quad t = 1, \ldots, T \tag{6}$$

$$w_{i,x,y,t} \in \{0,1\} \qquad \forall i \in N \quad x = 1, \ldots, d_i \quad y = 1, \ldots, d_i - x + 1 \quad t = 1, \ldots, T \tag{7}$$

$$w_{i,1,0,t} \in \{0,1\} \qquad \forall i \in N \quad t = 1, \ldots, T \tag{8}$$

Eq. (1) minimizes the total project makespan by minimizing the dummy end activity $n + 1$. Since this dummy end activity has a duration of zero, it consists of only one dummy end activity segment $S_1^{n+1} D_0$. Eq. (2) take the finish-start precedence relations with a time-lag of zero into account between two activities i and j. More precisely, the constraint connects the activity segments $S_x^i D_{d_i-x+1}$ of activity i with the activity segments $S_1^j D_y$ of activity j for each pair $(i, j) \in A$. Eq. (3) takes the relations between the activity segments for each activity i into account. The constraint builds the

segment chains discussed in Section 2.2.3 and adds a precedence relation between every couple of parts $S_x^i D_{y_1}$ and $S_{x+y_1}^i D_{y_2}$ with $x = 1, \ldots, d_i - 1$, $y_1 = 1, \ldots, d_i - x$ and $y_2 = 1, \ldots, d_i - x - y_1 + 1$. In doing so, one equation for each pair of connected activity segments, as illustrated in the right picture of Fig. 3 is added to the problem formulation. Eq. (4) assures that a feasible segment chain is selected. The equation is identical to Eq. (9) and will be discussed in detail in the *activity selection problem* of Section 3.2. It selects for each integer point $c = 1, \ldots, d_i$ a feasible segment chain among the generated activity segments for each activity i. Likewise, Eq. (5) assures that the decision variable for the dummy end activity - that only has one activity segment with duration 0 - is set to 1 for exactly one time moment t. Eq. (6) satisfies the renewable resource constraints. Eqs. (7) and (8) force the decision variables to be binary values. Note that the $w_{i,\,1,\,0,\,t}$ variables represent activity segments for activities with a zero duration ($y = 0$) and are necessary to model the precedence relations between the non-dummy activities and the dummy end activity.

3. Algorithmic design

In this section, the algorithm to solve the preemptive resource-constrained project scheduling problem with setup times is discussed. The algorithm used to solve the problem is inspired by and follows a similar approach of Coelho and Vanhoucke (2011, 2015) and Vanhoucke and Coelho (2016) to solve the multi-mode resource-constrained project scheduling problem with additional constraints. Section 3.1 presents the general outline of the solution approach which consists of a repeated dialogue between a so-called activity selection algorithm and an activity scheduling algorithm. Section 3.2 explains how an activity list that represents a project schedule will be repeatedly restricted and unrestricted to select activity segments in the project. Finally, Section 3.3 presents a detailed stepwise approach of the algorithmic components to solve the problem to near optimality.

3.1. General approach

The general outline of the algorithm consists of three main building blocks, for which a graphical summary is given in Fig. 7. A summary of these blocks is given along the following lines, and details are provided further in this section.

Construct network. The algorithm starts by transforming the original project network into the extended activity network as discussed earlier. First, the original activities are split into unique activity segments as discussed in Section 2.2. Next, the setup times are added to each activity segment as proposed in Section 2.3.1. Finally, the network reduction rules of Section 2.3.2 will remove inefficient activity segments to reduce the size of the extended network. This network construction phase is called only once at the beginning of the algorithm and results in an extended network in which each node represents an activity segment.

Activity list. The obtained extended network is now not fundamentally different from the original activity network, except that it contains more nodes representing activity segments instead of the original activities. Consequently, any resource-constrained project scheduling algorithm from literature can be used to solve the extended network, and will treat the nodes as individual activities instead of activity segments. For this reason, we can still call this extended network an activity network, as if each node represents an activity instead of an activity segment. In order to construct a project schedule for this activity network, such schedule must be represented by an *activity list*, which acts as a solution vector to prioritize the activities (represented by its segments as outlined earlier) in the project network. This activity list will be used by two optimization algorithms that constantly communicate with each other, and hence, the activity list will serve two purposes: select activity segments from the extended network (the *activity selection problem*) and schedule the selected segments to construct a resource-feasible project schedule (the *activity scheduling problem*).

Construct schedule. The construction of a resource-feasible schedule now consists of two separate algorithms that will be in a constant dialogue as long as the algorithm runs. One procedure takes care of the selection of a feasible combination of activity segments, while a second procedure schedules the selected segments in order to obtain the lowest possible project duration. The iterative switches between an *activity (segment) selection* step and an *activity scheduling* step aim at heuristically improving the project schedule towards a near-optimal solution with a predefined stop criterion, as will be outlined in Section 3.3. As mentioned earlier, both algorithms will make use of one and the same activity list to represent a solution vector and will repeatedly exchange the activity list. The exchange of the activity list is necessary to transform an unrestricted version of the list to a restricted version, and vice versa, and will be explained in detail in Section 3.2.

3.2. Restricting activity lists

The new extended activity network consists of all unique $S_x^i D_y$ activity segments for each original activity i, with precedence relations between these segments to model all possible feasible segment chains with activity preemptions. In order to construct a resource feasible schedule for this extended project network, the problem should be represented by an activity list which consist of one element for each activity segment. Such an activity list is then used to rank each activity segment, and this ranking determines the sequence in which these segments are added to the schedule to construct a resource feasible schedule. However, since each element in the activity list now represents an activity segment, not every activity segments should be put in the constructed project schedule. Instead, for each activity of the original project network, only a subset of all its activity segments should be selected such that it combines to a feasible segment chain. The selection of a subset of these activity segments results in a reduced activity list in which some of the elements are excluded and cannot be put in the schedule. Such a reduced activity list is called a *restricted activity list* while the original list that contains all activity segments is the *unrestricted* version of the activity list. In this section, a procedure is proposed to transform an unrestricted activity list into a restricted activity list in order to select a feasible combination of activity segments for each activity. In the remainder of this paper, this problem is referred to as the *activity selection* problem.

In order to assure that such a feasible segment chain is chosen for each activity, each activity i with a duration of d_i should be accompanied with a set of constraints for each integer point c ($c = 1, \ldots, d_i$) as follows:

$$\sum_{y=1}^{d_i} \sum_{x=\max(c-y+1,1)}^{\min(d_i-y+1,c)} S_x^i D_y = 1 \tag{9}$$

For the example of the activity with a 4 time units duration of Fig. 2, the activity can be cut at three different points, resulting in four parts with a duration of one time unit. Therefore, the following four constraints ($c = 1, \ldots, 4$) should be added to guarantee that exactly one of the 8 feasible segment chains of Fig. 3 will be chosen (index i removed from $S_x D_y$).

$$c = 1 : S_1D_1 + S_1D_2 + S_1D_3 + S_1D_4 = 1$$
$$c = 2 : S_1D_2 + S_1D_3 + S_1D_4 + S_2D_1 + S_2D_2 + S_2D_3 = 1$$
$$c = 3 : S_1D_3 + S_1D_4 + S_2D_2 + S_2D_3 + S_3D_1 + S_3D_2 = 1$$
$$c = 4 : S_1D_4 + S_2D_3 + S_3D_2 + S_4D_1 = 1$$

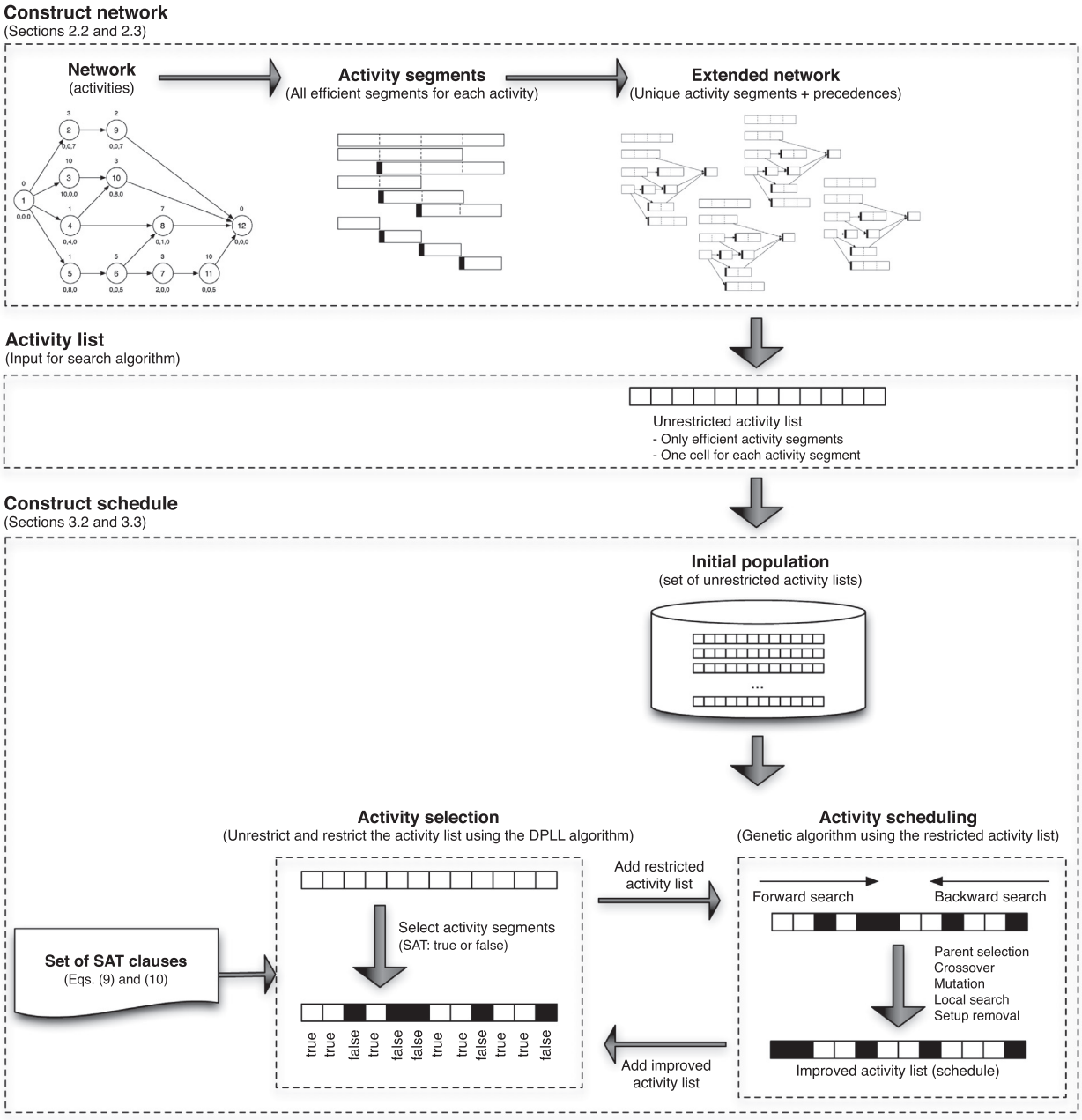

Fig. 7. A summary of the three building blocks for solving the PRCPSP-ST.

When each activity segment $S_x^i D_y$ is represented by a boolean variable z_k (true or false), the selection of a combination of segments according to Eq. (9) will always result in one of the 8 feasible segment chains of Fig. 3. This activity segment selection step modelled by a set of Eq. (9) for each activity of the original network can be easily modelled as a *boolean satisfiability problem* instance. The boolean satisfiability problem (SAT) is a well-known decision problem where an expression of boolean variables with the logical operators *and, or* and *not* is questioned to be true or false. A SAT algorithm makes use of a set of clauses that define the logical constraints of the decision problem. The problem has been studied extensively in literature (see e.g. the paper written by Marques-Silva and Sakallah, 1999, amongst others) and is known to be NP-hard (Cook, 1971). For example, when exactly one variable z_k must be chosen out of a set of K boolean variables, the following pair of clauses should be added to a SAT search algorithm.

$$\bigvee_{k=1,\ldots,K} z_k \tag{10}$$

$$\bigwedge_{\substack{k=1,\ldots,K-1 \\ l=k+1,\ldots,K}} (\overline{z_k} \vee \overline{z_l}) \tag{11}$$

where Eq. (10) models that at least one of the z_k variables can be chosen out of a set of K boolean variables, while Eq. (11) models that at most one z_k variable can be chosen.

Each variable z_k in a classic SAT model can now be replaced by an activity segment $S_x^i D_y$, and its two possible values (true or false) represent whether the activity segment will be added to the schedule or not. The SAT search makes now use of the unrestricted activity list containing all activity segments (i.e. all nodes of the

extended network), and its length is therefore equal to the sum of the number of activity segments for each activity. Each cell in this unrestricted activity list represents a boolean variable that determines whether this activity segment will be selected or not to be put in the schedule. Consequently, the purpose of the SAT search is to select a feasible combination of activity segments for each activity. This selection is done using SAT clauses to model the set of Eqs. (9), and returns an activity list with some of the activity segments set to true (selected activity segment) and some of them put to false (non-selected activity segments). This list is then forwarded to the activity scheduling step, where only the selected activity segments will be scheduled, and the non-selected activity segments receive a duration of zero.

More specifically, for each activity with duration d_i in the original project network, $2 \cdot d_i$ clauses will be added to the SAT search to model that exactly one feasible combination of activity segments can be selected. These selected activity segments are set to true, and will transform the unrestricted activity list into a restricted version. More precisely, a pair of clauses similar to the clauses of Eqs. (10) and (11) will be added for each activity i and for each value $c = 1, \ldots, d_i$. This set of clauses for each $c = 1, \ldots, d_i$ will be equal to:

$$\text{At least one :} \quad \bigvee_{\substack{x=1,\ldots,c \\ y=c-x+1,\ldots,d_i-x+1}} S_x^i D_y \tag{12}$$

$$\text{At most one :} \quad \bigwedge_{\substack{x_1=1,\ldots,c \\ x_2=x_1,\ldots,c \\ y_1=c-x_1+1,\ldots,d_i-x_1+1 \\ y_2=\begin{cases} y_1+1,\ldots,d_i-x_2+1, & \text{if } x_1=x_2 \\ c-x_2+1,\ldots,d_i-x_2+1, & \text{otherwise} \end{cases}}} (\overline{S_{x_1}^i D_{y_1}} \vee \overline{S_{x_2}^i D_{y_2}}) \tag{13}$$

For the example of the activity with a duration of 4 time units, the $c = 1$ equation can be modelled with the following 7 clauses (index i removed from $S_x D_y$):

At least one (1 clause): $S_1D_4 \vee S_1D_3 \vee S_1D_2 \vee S_1D_1$

At most one (6 clauses):

$(\overline{S_1D_4} \vee \overline{S_1D_3}) \wedge (\overline{S_1D_4} \vee \overline{S_1D_2}) \wedge (\overline{S_1D_4} \vee \overline{S_1D_1})$

$\wedge(\overline{S_1D_3} \vee \overline{S_1D_2}) \wedge (\overline{S_1D_3} \vee \overline{S_1D_1}) \wedge (\overline{S_1D_2} \vee \overline{S_1D_1})$

For the same example of the activity with a duration of 4 time units, the number of clauses grows to 16 for the $c = 2$ equation, as follows:

At least one (1 clause):

$(S_1D_2 \vee S_1D_3 \vee S_1D_4 \vee S_2D_1 \vee S_2D_2 \vee S_2D_3)$

At most one (15 clauses):

$(\overline{S_1D_2} \vee \overline{S_1D_3}) \wedge (\overline{S_1D_2} \vee \overline{S_1D_4}) \wedge (\overline{S_1D_2} \vee \overline{S_2D_1}) \wedge$

$(\overline{S_1D_2} \vee \overline{S_2D_2}) \wedge (\overline{S_1D_2} \vee \overline{S_2D_3}) \wedge (\overline{S_1D_3} \vee \overline{S_1D_4}) \wedge$

$(\overline{S_1D_3} \vee \overline{S_2D_1}) \wedge (\overline{S_1D_3} \vee \overline{S_2D_2}) \wedge (\overline{S_1D_3} \vee \overline{S_2D_3}) \wedge$

$(\overline{S_1D_4} \vee \overline{S_2D_1}) \wedge (\overline{S_1D_4} \vee \overline{S_2D_2}) \wedge (\overline{S_1D_4} \vee \overline{S_2D_3}) \wedge$

$(\overline{S_2D_1} \vee \overline{S_2D_2}) \wedge (\overline{S_2D_1} \vee \overline{S_2D_3}) \wedge (\overline{S_2D_2} \vee \overline{S_2D_3})$

and a similar approach can be followed for the $c = 3$ and $c = 4$ equations. It should be noted that the "at most one" constraint of Eq. (13) consist of all pairs of activity segments that overlap with the vertical line with value c in Fig. 2, and has been modelled in our SAT solver as a so-called *cardinality constraint*.

In order to transform the unrestricted activity list into a restricted activity list, a procedure is used that works in two phases. The first phase is executed in advance, and generates all clauses for Eqs. (12) and (13) for each activity of the original activity network. These clauses are used to transform each unrestricted activity list into a restricted activity list. This second phase is done iteratively and is embedded in the activity scheduling algorithm discussed in Section 3.3. Indeed, during the search for a feasible project schedule, the unrestricted activity list is used by a metaheuristic scheduling algorithm, which iteratively changes this unrestricted activity list to search for better solutions. For each such unrestricted activity list, a feasible set of activity segments must be selected, and this selection then results in a restricted version of the activity list. The selection of feasible activity segments satisfying all these clauses is done using a fast and efficient boolean satisfying problem solver. More specifically, each boolean satisfying problem instance can be easily solved using the efficient DPLL algorithm of Davis et al. (1962). The DPLL algorithm is a complete, backtracking-based algorithm for deciding the satisfiability of propositional logic formulae in the conjunctive normal form. The algorithm sequentially selects variables from a pre-defined variable list - which is equal to the unrestricted activity list - and assigns a true value to it in order to simplify the conjunctive normal form formula. If this assignments leads to an unsatisfiable simplified formula (referred to as a conflict), the opposite value (false) is set to the selected variable and the algorithm continues. The unit propagation rule checks whether a clause is a unit clause, i.e. it contains only a single unassigned variable. When this is the case, this clause can only be satisfied by assigning the necessary value to make this variable true. The algorithm stops when it has found a feasible set of activity segments, and returns a restricted activity list, in which the elements set to true are equal to the selected activity segments, while the other elements are set to false.

It should be noted that the DPLL algorithm only serves as an enumeration procedure that selects a feasible combination of elements from the unrestricted activity list, and sets all the non-selected elements to false. This activity selection problem is a separate propositional problem for every activity i, and each individual problem is independent from the activity selection problem of the other activities. Consequently, conflicts will never occur since the algorithm will always select the first feasible combination of elements in the unrestricted activity list that satisfies all clauses for that particular activity. We nevertheless have chosen to use the DPLL algorithm instead of a simplified enumeration algorithm since it is a general SAT solver that can be used even if more clauses are added to the activity selection problem, which can possibly lead to conflicts during the search.

As an example, the problem can be extended by an additional restriction that specifies that the total number of activity splits for the complete project cannot exceed a predefined limit. Such an extension could result in conflicts during the SAT search, and hence, the use of the DPLL algorithm would be preferred above a simplified enumeration procedure to fully exploit the ability for clause learning. Setting a limit L_s on the maximum number of splits in the project schedule can be easily done by adding the following equation:

$$\sum_{i=1}^{|N|} \sum_{x=2}^{d_i} \sum_{y=1}^{d_i-x+1} S_x^i D_y \leq L_s \tag{14}$$

Another straightforward extension could consist of restricting the sum of all the setup times T_s to a certain value, which can be easily done by extending Eq. (14) to a weighted version using the following equation:

$$\sum_{i=1}^{|N|} \sum_{x=2}^{d_i} \sum_{y=1}^{d_i-x+1} t_i^s(type, \%s, x) \cdot S_x^i D_y \leq T_s \tag{15}$$

with $t_i^s(type, \%s, x)$ the value of the setup time of activity segment $S_x^i D_y$ of activity i with duration d_i. This setup time value is calculated as proposed in Section 2.3.1.

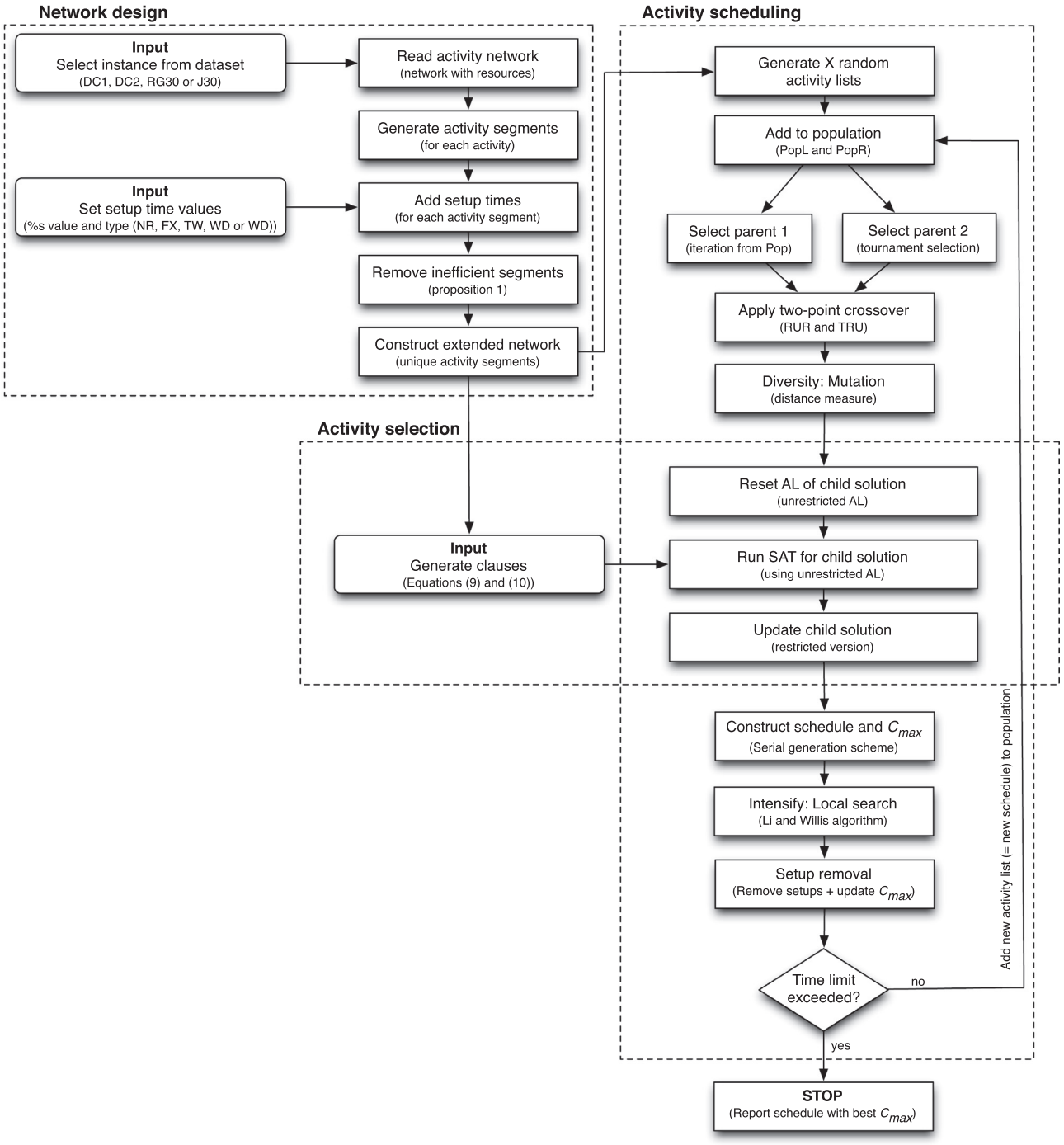

Fig. 8. Stepwise solution approach (network design, activity selection, activity scheduling).

3.3. Stepwise solution approach

This section presents a stepwise summary of the components of the solution approach consisting of the network design phase followed by the execution of both the activity selection and the activity scheduling algorithms that make use of the activity list as mentioned earlier. A graphical stepwise approach is shown in Fig. 8 and details are given along the following lines.

Phase 1. Network design. The *network design* phase has been explained earlier and aims at constructing an extended activity network containing all activity segments and setup times. In the computational experiments of Section 4, it will be shown that the input network instances will be selected from 4 separate datasets (labelled as DC1, DC2, RG30 and J30 in the picture), and the setup times will be generated using values for the fraction %s of each activity's unit duration.

Phase 2. Activity scheduling. The *activity scheduling* algorithm is implemented as a dual population-based genetic algorithm inspired by the decomposition based genetic algorithm of Debels and Vanhoucke (2007). The algorithm treats the extended activity network as a new project instance as if each node is an individual activity instead of an activity segment. Consequently, the algorithm starts with generating a random pool of solution vectors represented by unrestricted activity lists, and keeps working on these

lists until a certain predefined time limit is exceeded. Each element of these activity lists represents an activity segment and since these lists are unrestricted versions, all activity segments are initially set to true (i.e. $z_k = 1$).

After the generation of a random pool of unrestricted activity lists, two separate populations are created known as a left population PopL that contains left-justified schedules and a right population PopR that contains right-justified schedules (Valls et al., 2005). In doing so, the algorithm can fully exploit the advantages of the iterative forward/backward scheduling technique of Li and Willis (1992) by transforming activity lists into a resource-feasible schedules using a forward or backward scheduling approach. Such a dual population approach combined with an iterative forward/backward local search (which will be discussed later) have been shown to outperform a single population approach in Debels and Vanhoucke (2007). The iterative process consecutively adapts the population elements of PopL and PopR using the different solution components that will be explained in this section. The right (left) population is updated by feeding it with combinations of population elements taken from the left (right) population. Therefore, the left (right) population elements are transformed in right-justified (left-justified) schedules by means of the well-known serial schedule generation scheme.

The algorithm enters the *parent selection* step to select couples of population elements from each pool. The two parents are selected from the population, either PopL or PopR, as follows: the first parent is iteratively selected by selecting each element one by one from the pool, while the second parent is selected using the two-tournament selection where two population elements from the population are chosen randomly, and the element with the best makespan value is selected. Afterwards, the algorithm randomly labels one element as the father and the other element as the mother, and both elements will now be subject to changes in their unrestricted activity list to search for improved solutions.

Once the parents are selected, a two-point crossover is applied to generate a new child solution from these two parent solutions. The genetic algorithm relies on a modified version of the peak crossover operator of Valls et al. (2008) which works as follows. First, the algorithm calculates for each parent schedule the so-called *resource utilization ratio* (RUR) which measures the average utilization at each time unit t along the complete time horizon of the schedule. When the resource utilization at time t is high, it is referred to as a peak, while time instants with low resource utilizations are called valleys. The two-point crossover aims at selecting a so-called peak interval in which the total resource utilization is high (i.e. contains mostly peaks). Such an interval is chosen as the interval in which the *total resource utilization* (TRU) is maximized. To that purpose, the algorithm selects a time interval with length l randomly generated between 25% and 75% of the total project makespan. For each possible interval with this length l, the TRU is calculated as the cumulative value of RUR for each time instant of that interval. The interval with the highest value for RUR determines the two points (start time instant and end time instant) for the crossover. This approach is not new and has been successfully adapted in Debels and Vanhoucke (2007).

In order to avoid being trapped in a local optimum and to guarantee that the population contains heterogeneous solutions, diversification is added to the algorithm using a mutation operator. The implemented mutation method only operates on a child solution when it originates from two parents who are not mutually diverse enough. To that purpose, an easy distance measure must be used that simply calculates the average of the absolute deviations between the priority values of the activity lists of both parents. When this distance measure does not exceed a certain threshold, a small proportion of elements in each activity list is randomly changed to new values.

Phase 3. Activity selection. The newly obtained child activity list is now input for the *activity selection* step, in which some of the activity segments are set to true (i.e. $z_k = 1$) while other activity segments are set to false (i.e. $z_k = 0$), as proposed in Section 3.2. From unrestricted to restricted, such that it can be used to generate a feasible schedule. Based on this resulting activity segment selection, a project schedule is constructed using a serial generation scheme, with non-selected activity segments put to zero duration activities. In doing so, only the selected activities will be put into the resulting project schedule. The improved activity list is entered back into the activity selection step to select another feasible combination of activity segments. This means that the improved activity list is first reset to an unrestricted version in which all activity segments are set back to true, and the DPLL algorithm of the activity selection step will now use this new unrestricted activity list to select another set of activity segments. This will result in a new restricted activity list which will be given back to the activity scheduling step.

Phase 2. Activity scheduling. After the activity selection step, the restricted activity list is returned back to the *activity scheduling* step, and a serial generation scheme will transform this list into a resource-feasible schedule, using a forward or backward search through the activity list, depending on whether the schedule belongs to the PopL or PopR pool. Obviously, while each element of the activity list will be scheduled, the elements with a duration equal to zero will not appear in the schedule, and hence, only feasible combinations of activity segments will be visible. This restricted activity list is therefore not different than the unrestricted activity list from the activity scheduling algorithm's point of view, but will never create schedule with infeasible activity segments.

The resulting makespan $C_{\max}$ of the obtained project schedule can potentially be further improved using a local search procedure. This intensification to search for better neighbourhood solutions of the newly constructed activity list is done using the forward/backward local search procedure of Li and Willis (1992). This algorithm iteratively improves the total project makespan by scheduling the project using a forward and backward search and stops if no further makespan improvements can be found. Since our genetic algorithm makes use of two separate populations (a left-shifted and a right-shifted population), the local search procedure will feed only one of these sub-populations depending on how the best found schedule is a left- or right-shifted schedule.

Finally, the newly found schedule can still be improved using the *setup removal* step. During the construction of the scheduling using the serial schedule generation scheme, there is no built-in guarantee that two or more activity segments of the same activity are really scheduled with interruptions. It is therefore possible that the finish time of an activity segment $S^i_{x_1}D_{y_1}$ is equal to the start time of another segment of the same activity $S^i_{x_2}D_{y_2}$, even though a setup time is taken into account. Indeed, in case of scheduling two (or more) activity segments, the second activity segment will contain a setup time on top of its duration y_2, even though the activity is not really split in the schedule. Therefore, an easy and quick local optimisation procedure is called each time the scheduling algorithm schedules the individual activity segments. Every time a segment $S^i_{x_2}D_{y_2}$ is added to the partial schedule for which another segment $S^i_{x_1}D_{y_1}$ is already in the partial schedule with a finishing time equal to the start time of the current segment to be scheduled, the two activity segments $S^i_{x_2}D_{y_2}$ and $S^i_{x_1}D_{y_1}$ are replaced by activity segment $S^i_{x_1}D_{y_1+y_2}$, and therefore, the setup time of the second segment is no longer taken into account and will temporarily be removed from the activity segment for that specific schedule. After removing this setup time, a left shift in the schedule is performed to schedule some activity segments earlier in time, possibly leading to a makespan decrease. In case the

scheduling algorithm is performed as a backward run instead of a forward run (since the local search approach is based on the forward and backward scheduling algorithm of Li and Willis, 1992), a similar setup time removal approach is performed, this time checking whether the activity segment to be added to the partial schedule has a finish time equal to the start time of another activity already added to the schedule.

Dialogue between phases 2 and 3. As a summary, the algorithm operates under a constant dialogue between the two procedures, an activity selection algorithm that selects activity segments and restricts the original activity list by setting some segments to false and an activity scheduling algorithm that improves the restricted activity list to obtain a near-optimal project schedule. The size of the activity list remains constant and is equal to the total number of activity segments in the extended project network. The activity selection step puts each time different combinations of activity segments to true or false while returning it back to the genetic algorithm of the activity scheduling step. It has been previously shown by Coelho and Vanhoucke (2011) that the use of a single activity list for both the activity selection step (to solve the SAT instance) and the activity scheduling step (to solve the RCPSP instance) does not exclude optimal solutions. This back and forth switching mechanism between activity selection and activity scheduling is carried out each time the algorithm must evaluate the objective function (the project makespan) of a newly created activity list, and stops after a predefined number of schedules are created and evaluated, or a time limit is exceeded.

4. Computational results

In order to test the impact of setup times on the reduction of the project makespan (relative to the optimal project makespan without allowing activity preemption), a set of computational experiments have been carried out using different project datasets with different characteristics, as will be explained in the following subsections. All computational experiments were carried out using the Stevin Supercomputer Infrastructure at Ghent University. We have used a cluster with 62GB RAM and with a Intel(R) Xeon(R) CPU E5-2680 v3 with 2.50 GHz.

4.1. Project data

In order to create insights into the impact of activity preemption within the presence of setup times, different computational experiments have been performed using different datasets and settings for the setup times, as described below.

Datasets: The new algorithm has been tested using various instances from 4 different datasets with different values for the network structure (measured by the serial/parallel indicator SP (Vanhoucke et al., 2008)) and the resource constrainedness RC (Patterson, 1976). The number of activities (# Act) ranges from 10 to 50 and each set contains a different number of project instances (# Inst). These datasets include the DC2 set (only the npv_{25} subset is used) containing 25-activity instances, the RG30 dataset with 30-activity instances with a diverse topological network structure, the well-known J30 instances of the PSPLIB library and the DC1 set containing instances from 10 to 50 activities. The specific characteristics of these four datasets are summarized in Vanhoucke et al. (2016) and a brief summary of their main characteristics and their use for the computational experiments is given in Table 3.

Setup times: In order to test the impact of the size of the setup times on the makespan reduction when allowing activity preemption, different values for the setup time factor $\%s$ has been used (0.1, 0.2 and 0.5), each value resulting in five setup times (non-related, fixed, total work, work done and work remaining) as shown in Table 3 in the columns "$\%s$" and "Type". The exact value of the setup times $t_i^s(type, \%s, x)$ can be easily replicated by other researchers using the formulas discussed in Section 2.3.1. The setup times for the NR type for activity i are generated as $(997 + 487 \cdot i) \bmod d_i$, with "$a$ mod b" the modulo operation to find the remainder after division of a by b.

Algorithms: The optimal solutions of the RCPSP (no preemption) are obtained using the branch-and-bound procedure of Demeulemeester and Herroelen (1992). Table 3 displays the percentage of instances that could be solved to optimality within a time limit of 1 h (column "% Opt") and shows that only for the RG30 instances, not all solutions could be found. The test runs for the RCPSP with activity preemption but without setup times (i.e. problem PRCPSP and $\%s = 0$) are obtained by two procedures. First, the exact procedure of Demeulemeester and Herroelen (1996a) is used within a predefined time limit of 60 s, as well as the heuristic procedure of this paper truncated after 500,000 generated schedules. The test runs for the PRCPSP-ST problem (i.e. with $\%s > 0$) are carried out by the procedure discussed in this paper, truncated after 50,000 generated schedules.

Performance indicators: In order to assess the performance of the meta-heuristic procedure for the RCPSP with preemption and setup times, three indicators haven been used. First and foremost, the percentage deviation (%Dev) relative to the optimal makespan when the instance is solved to optimality without activity preemption and setup times is calculated. This measure is used for obvious reasons to assess whether and in which cases activity preemption, with and without the presence of additional setup times, is able to reduce the total project duration. Secondly, the resource utilization (%RU) is measured, expressed as a percentage which equals the total work content used divided by the total available work content (i.e. the makespan multiplied by resource availability). The metric is calculated to test whether activity preemption (with and without setup times) results in a more efficient use of resources. Finally, the percentage of instances with an improvement (%Imp) in the makespan compared to the makespan of the RCPSP solution (no preemption, no setup times) is measured. As explained earlier, in the RG30, only 89.17% of the solutions are known to be optimal, and for the remaining instances, the best known solutions are used as a benchmark.

4.2. Impact of problem size

Table 4 shows the size of the problem instance for the DC1 dataset, which is the only set with a variable number of activities ranging from 10 up to 50. The number of activity segments is shown in the column "#Seg" (cf. Fig. 2 for a visualisation of these activity segments) as well as the number of feasible segment chains "#SC" when these activity segments are combined, as shown in Fig. 3. The results are shown for the five types of setup times, after applying the network reduction rules of Section 2.3.2. The table shows that these network reduction rules have no impact on the non-related (NR), fixed (FX) and work done (WD) setup times, but drastically reduce the size of the project network for the work remaining (WR) and, albeit to a lesser extent, the total work (TW) setup times. Nevertheless, the table also shows that the activity network can grow quickly, ending with project networks with more than 1000 activity segments for 50-activity networks, resulting in more than 5000 feasible segment chains. Hence, an efficient meta-heuristic search procedure that quickly iterates between the activity selection and scheduling steps is necessary to obtain high-quality solutions in short times.

Table 3
Summary of datasets used in the computational experiment.

	# Inst	# Act [min/max]	SP [min/max]	RC	%s	Type	% Opt
DC2[*]	180	25	[0.08; 0.41]	[0.25; 0.75]	0.1, 0.2, 0.5	NR,FX,TW,WD,WR	100%
J30	480	30	[0.17; 0.41]	[0.10; 0.60]	0.5	NR,FX,TW,WD,WR	100%
RG30	1800	30	[0.10; 0.90]	[0.4; 0.4]	0.5	NR,FX,TW,WD,WR	89.17%
DC1	1800	10–50	[0.22; 0.84]	[0.26; 0.82]	0.5	NR,FX,TW,WD,WR	100%

Table 1 Only subset npv_{25}.

Table 4
Number of activity segments and feasible segment chains (after using reduction rules).

	NR, FX, WD		WR		TW	
#Act	#Seg	#SC	#Seg	#SC	#Seg	#SC
10	222	1002	120	169	148	287
20	442	2048	238	339	297	584
30	678	3220	365	527	455	911
40	894	4166	483	684	603	1186
50	1104	5120	601	851	744	1454

4.3. Impact of setup times

Table 5 displays the results for "%Imp", "%Dev" and "%RU" for the four datasets with %s values set to 0.5 for the DC1, J30 and RG30 dataset, and to 0.1, 0.2 and 0.5 for the DC2 dataset. Each time, the results are displayed for each of the five types of setup times, and these results are compared with the preemptive problem without setup times (i.e. PRCPSP with $\%s = 0$) and when necessary, with the problem without preemption (i.e. RCPSP).

The table shows that activity preemption pays off and leads to an average reduction in the project makespan of the RCPSP up to 1.44% (which is the case for the RG30 dataset, as can be seen in the PRCPSP row). This percentage deviation %Dev obviously decreases when setup times are taken into account, but reductions around 0.5% are still observed. The table also shows that this is the case for approximately 40% to 50% of the project instances without setup times and to somewhat less than one third when setup times are taken into account (%Imp). These two performance metrics (%Dev and %Imp) illustrate that activity preemption with setup times pays off, not in a huge extent, but nevertheless reduces the project makespan with a few days. Obviously, the impact on the resource utilization (%RU) is clear, and the table shows that the %RU increases only a little bit when activity preemption without setup times (PRCPSP) and with setup times (PRCPSP-ST) is taken into account (compared to the bottom row with no preemption and no setup times (RCPSP)).

Table 5 also illustrates that activity preemption can lead to makespan reduction under a wide variety of settings. The table shows not much differences between the different values for the setup times, and illustrates that preemption might be useful even with high values for the setup times (up to $\%s = 0.5$ in the experiments). Moreover, while it is obvious that the fixed setup times lead to the biggest gain in makespan reduction, also the other setup times which values depend on the activity duration contribute, although to a lesser extent, to a makespan reduction.

It is important here to note that the small makespan reductions nevertheless show that activity preemption is useful, even within the presence of setup times. These reductions should also be interpreted in the light of the newly obtained schedule. Splitting activities in the project schedule can totally disturb the logic of the activity network, and cut every single activity into little pieces, making the new schedule useless, or at least less clear to the project manager and his/her team. Therefore, setup times between preemptive activity segments can be seen as a tool to avoid

Table 5
Results for the PRCPSP-ST ($\%s > 0$) and comparison with PRCPSP ($\%s = 0$) and RCPSP (no preemption).

		DC1	J30	RG30	DC2		
	%s	0.5	0.5	0.5	0.5	0.20	0.10
%Imp	NR	26%	24%	25%	21%	24%	28%
	FX	36%	33%	32%	27%	29%	29%
	WD	26%	20%	23%	20%	25%	28%
	WR	28%	26%	24%	21%	27%	27%
	TW	21%	17%	18%	17%	25%	27%
	PRCPSP	43%	41%	53%		38%	
%Dev	NR	0.46%	0.51%	0.40%	0.54%	0.67%	0.74%
	FX	0.76%	0.73%	0.55%	0.68%	0.79%	0.83%
	WD	0.38%	0.36%	0.28%	0.40%	0.65%	0.72%
	WR	0.49%	0.45%	0.35%	0.46%	0.65%	0.70%
	TW	0.33%	0.30%	0.24%	0.34%	0.56%	0.70%
	PRCPSP	1.16%	1.32%	1.44%		1.25%	
%RU	NR	45.75%	48.74%	54.29%	60.82%	60.89%	60.95%
	FX	45.90%	48.85%	54.25%	60.92%	61.01%	61.03%
	WD	45.70%	48.64%	54.15%	60.68%	60.88%	60.95%
	WR	45.75%	48.69%	54.21%	60.74%	60.88%	60.95%
	TW	45.66%	48.59%	54.10%	60.61%	60.80%	60.92%
	PRCPSP	46.16%	49.24%	55.23%		61.38	
	RCPSP	45.55%	48.47%	54.26%		60.55%	

Table 6
Average and maximum number of splits for each setup time type.

		Proposed procedure (all instances)				Priority rules (only improved instances)				Priority rules (all instances)			
		DC1	J30	RG30	DC2	DC1	J30	RG30	DC2	DC1	J30	RG30	DC2
AvgSplits	NR	1.9	1.9	2.6	2.2	3.7	5.5	3.6	1.0	32.0	21.5	43.9	64.3
	FX	2.2	2.4	3.1	2.3	5.4	7.4	7.5	1.0	30.0	19.3	39.9	61.0
	WD	1.8	1.8	2.1	1.6	3.0	4.4	3.0	1.0	30.4	19.4	39.9	60.6
	WR	1.8	1.8	2.3	1.7	2.7	3.0	3.7	1.0	30.5	20.0	40.5	61.6
	TW	1.6	1.6	2.0	1.2	2.8	4.3	2.8	1.0	30.1	19.3	40.1	60.7
MaxSplits	NR	8	5	9	5	14	12	27	1	224	131	145	130
	FX	9	8	20	6	17	17	37	1	215	102	132	130
	WD	7	5	10	4	10	6	12	1	219	106	141	130
	WR	6	5	8	4	7	4	17	1	218	107	142	130
	TW	6	4	10	3	8	7	15	1	215	118	133	130

to many activity splits, and steer the algorithm towards a schedule with makespan improvements (%Dev) without cutting every single activity in pieces. Therefore, Table 6 shows the number of splits in the problem (expressed as the average number of splits, as well as the maximum number of splits in all project activities). The table also compares the results obtained by our procedure with the results of an easy-and-fast procedure that relies on priority rules. More precisely, this procedure simply transform each project instance to a project with activities with unit durations and then schedules this project using the serial generation scheme with some well-known priority rules. When an activity with unit duration is not scheduled immediately after the finish of a unit activity belonging to the same original activity, a setup time is added. The table displays results for the *latest start time* (LST) priority rule, but similar results were found for other priority rules found in the literature. Most of the instances could not reduce the project makespan when activity preemption and setup times are introduced. That is the reason why the table makes a distinction between the priority rules results for the instances with improvements (i.e. for which the makespan is reduced) and results for all instances (improvements or not). First, the table shows that the number of splits is reasonably low and hence will not lead to a completely different schedule compared to its non-preemptive version. This low number of activity split could not be reached with the priority rule based scheduling approach. Secondly, the table also illustrates that most datasets have identical values, except the RG30 dataset. The RG30 set is the most diverse set in terms of network structure (with SP values ranging between 0.10 and 0.9) and more room for improvements in the schedule is possible. Thanks to the wider range of network topology, the number of splits grows to 2–3 (on average) and up to 20 (maximum), which also results in higher makespan reductions (%Dev and %Imp) as seen in Table 5. It is therefore recommended to use the RG30 set for research purposes when the morphological structure might influence the results under study.

It should be noted that the results are somewhat in line with literature, but nevertheless, some difference can be found. It has been argues earlier in literature (Demeulemeester and Herroelen, 1996a; Vanhoucke and Debels, 2008) that activity preemption has almost no effect on both the lead time and the resource utilization. In Vanhoucke and Debels (2008) it is concluded that the *task splitting option of project scheduling software, which results in preemptive and often less clear schedules, is no good alternative to improve the schedule quality*. Despite the observation that in our study no major makespan reduction could be found either, it has been shown that values up to 1.44% could be found for the PRCPSP and values still up to 0.5% and higher with the presence of setup times. These values are higher than the values found in literature, and can mainly be attributed to the use of another dataset. Varying and widening the range for the network topology, as is done for the RG30 dataset, clearly shows that improvements can be found in the project makespan. It should also be noted that the average resource utilization is lower in the current paper than the values found by Vanhoucke and Debels (2008). This is because our data instances all make use of 4 renewable resources, while the previous study analysed project instances with only 1 renewable resource type.

4.4. Fast tracking

In this section, the algorithm is used for solving a similar problem discussed in the literature where setup times are used when parts of activities overlap. The *within-activity fast tracking* option is proposed in Vanhoucke (2008) and assumes that activities can be split in d_i sub-activities that can be executed in parallel. Their problem formulation is inspired by the observation that activities are often executed by groups of resources (with a fixed availability), but the total work can often be done by multiple groups (in parallel). They therefore present the *pre-emptive resource-constrained project scheduling problem with fast tracking* (PRCPSP-FT) problem and split each activity in d_i sub-activities without precedence relations between the various subparts, each with a resource requirement r_{ik}.

It should be noted that the concept of setup times can be interpreted in various ways, and are used to model the lost time due to overlaps between activity segments, or alternatively, are ways to model the risk for rework when parts of an activity that normally should be scheduled sequentially are now scheduled in parallel. However, this section only serves as an illustration that our algorithm is able to cope with the problem from literature, without the need to change anything in the procedure. Fig. 9 shows the schedule found for this PRCPSP-FT using the same settings as the previous example (the PRCPSP-ST problem in Fig. 5, i.e. setup times using the work done option with $\%s = 0.5$).

In Table 7, the most important results are displayed for the PRCPSP and show that, even without using setup times, the makespan reductions are significantly higher when fast tracking is used. The values in this table should be compared with the values of Table 8 which penalizes activity preemption and the activity fast tracking with setup times. The comparison leads to obvious results. Using setup times for activity preemption pays off, as seen in the previous paragraphs, but with setup times up to $\%s = 0.5$ the reduction in the project makespan relative to the RCPSP makespan decreases. Secondly, allowing activity fast tracking obviously leads for further makespan reductions which are significantly higher, even within the presence of setup times, but still have a relatively low number of activity splits and therefore leads to a manageable project baseline schedule.

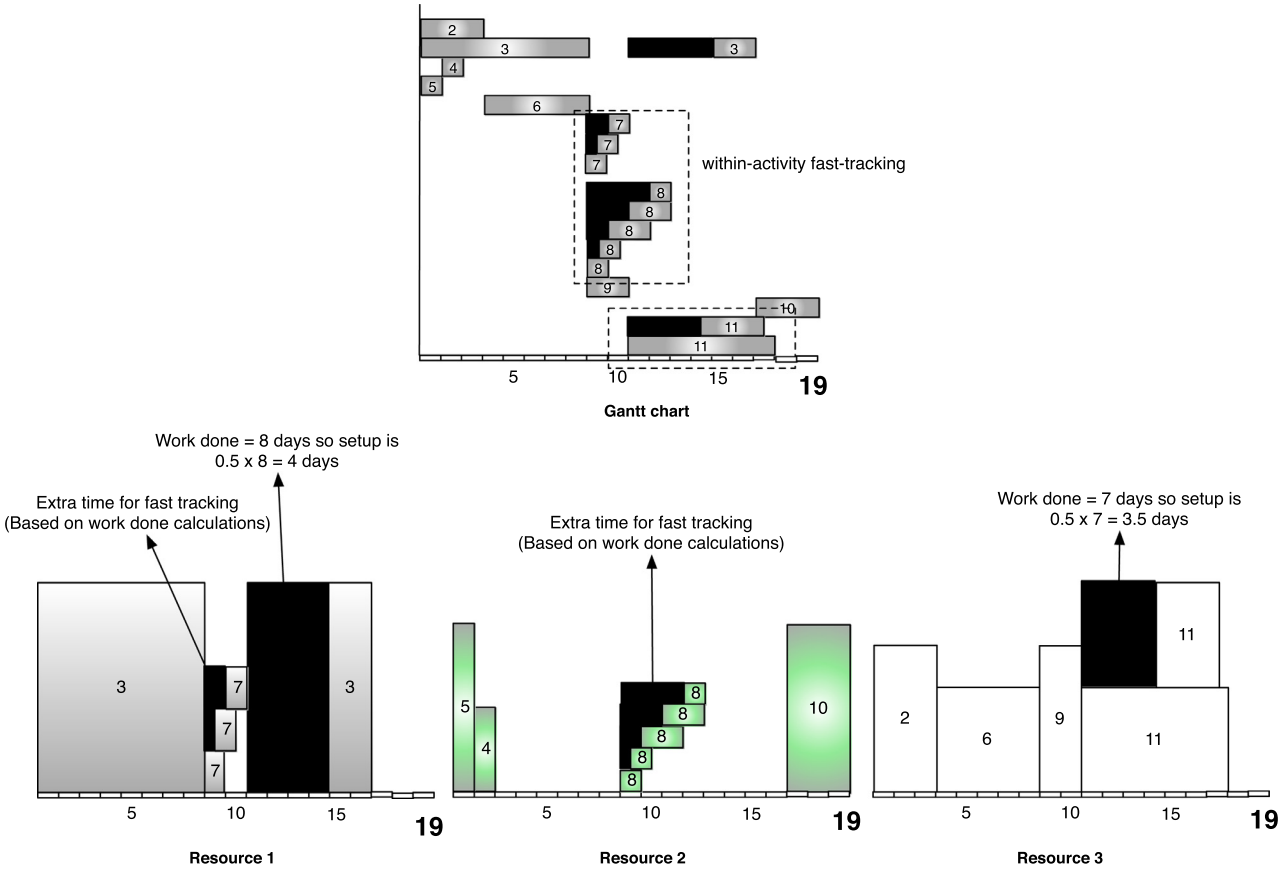

Fig. 9. Time decrease from 22 to 19 days with within-activity fast-tracking (work done and %s = 0.5).

Table 7
Impact of fast tracking for the PRCPSP (%s = 0).

	No fast tracking				Fast tracking			
	DC1	J30	RG30	DC2	DC1	J30	RG30	DC2
%Imp	43%	41%	53%	38%	91%	100%	97%	77%
%Dev	1.16%	1.32%	1.44%	1.25%	14.26%	25.32%	14.52%	9.93%
%RU	46.16%	49.24%	55.23%	61.38%	52.39%	64.83%	62.44%	67.21%

Table 8
Impact of fast tracking for the PRCPSP-ST (%s=0.5) (only DC2 dataset).

	No fast tracking				Fast tracking			
	%Imp	%Dev	%RU	#Splits	%Imp	%Dev	%RU	#Splits
FX	28%	0.69%	60.67%	2.3	74%	8.08%	66.09%	7.8
WD	20%	0.40%	60.54%	1.6	66%	5.50%	65.45%	6.9
WR	21%	0.47%	60.57%	1.6	67%	6.60%	65.68%	7.2
TW	17%	0.34%	60.47%	1.2	61%	5.79%	65.52%	6.8

4.5. Impact of problem parameters

Fig. 10 shows the impact of the project parameters (both the network topology and the resource scarceness) on the reduction in the project makespan (%Dev) and the number of instances with a reduction (%Imp) for a setup time value %s = 0.5 with (right y-axis) and without (left y-axis) fast tracking. The results are reported for the instances in the RG30 dataset.

The network topology is measured by three indicators. The well-known and widely used order strength (OS, Mastor, 1970) is used and compared with the alternative serial/parallel indicator (SP, originally called the I_2 indicator in Vanhoucke et al., 2008 and later rebranded as the SP indicator in Vanhoucke, 2010). Both indicators aim at measuring the closeness of a project network to a complete serial or parallel network structure. Moreover, the activity distribution indicator (AD, also known as the I_3 indicator) is used that measures how the activities are distributed in the project network. The scarceness of the renewable resources is measured by the resource strength (RS, Kolisch et al., 1995). The alternative indicator known as the resource constrainedness (RC, Patterson, 1976) used in Table 3 is not used since its values are fixed to 0.4 for the RG30 set. For an extensive discussion on network topology and resource indicators, the reader is referred to Demeulemeester et al. (2003) and Vanhoucke et al. (2008).

The graphs show a similar behaviour for all indicators, but also illustrates that the impact of activity preemption with setup times

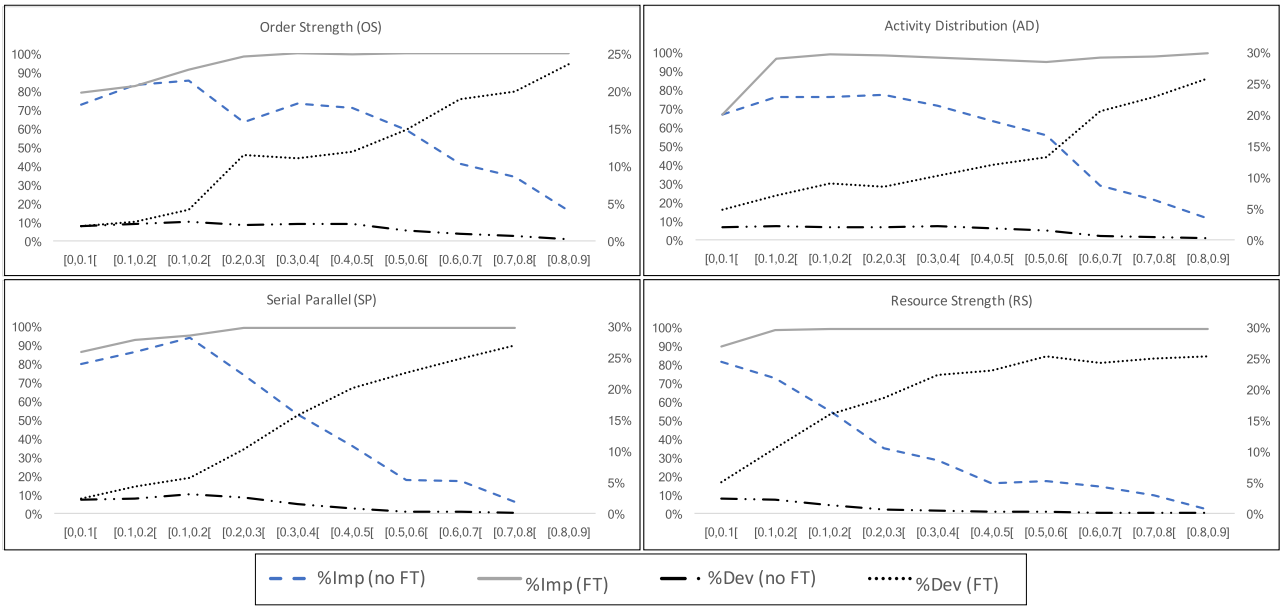

Fig. 10. Impact of the network structure (OS, SP and AD) and resource scarceness (RS) for the RCPSP-ST and RCPSP-FT with $\%s = 0.5$.

shows an opposite behaviour for the problem with and without fast tracking.

The results for the PRCPSP-ST without fast tracking obviously show that projects with a lower degree of scheduling freedom benefit less from activity preemption. First, the more serial the project network, the less likely that splits will lead to makespan reductions. Activity splits will hardly have any impact on the makespan, since these preemptive parts cannot be scheduled much earlier to the sequential structure of the project. Likewise, the higher the resource availability, measured by higher values for the RS, the lower the makespan reduction when preemption is allowed. Low RS values, on the contrary, means that resources are more restricted, and hence, more activities will be scheduled sequentially due to the low availability of resources. Activity preemption will occur more often to schedule parts of the preemptive activities earlier in time at places where resources are not fully congested. These results are obvious, since highly serial networks and/or high values for the resource strength leave little room for improvements, and hence, activity preemption will not add much value to the schedule. A similar observation can be drawn for the AD indicator. When activities are uniformly distributed over the network structure (i.e. high AD values), the less degrees of freedom in the network structure, and lower the beneficial impact of activity splitting.

The results for the problem with fast tracking shows opposite results, simply because allowing within-activity fast tracking relaxes and partially changes the network structure. More serial projects benefit more from fast tracking since many activity segments can be scheduled in parallel (while obviously keeping the serial structure between activities) and hence a reduction in the makespan will occur much easier for these projects. For highly parallel networks (low SP and OS values), fast tracking reduces much less from the project makespan reduction, and hence, the preemption will be less beneficial. A similar reasoning can be applied for the resource strength (RS).

5. Conclusions

In this paper, the well-known preemptive resource-constrained project scheduling problem (PRCPSP) has been extended with five types of setup times to penalize the presence of activity splits. We propose a meta-heuristic search algorithm to solve this so-called preemptive resource-constrained project scheduling problem with setup times (PRCPSP-ST) that iteratively solves two underlying subproblems. The activity scheduling problem consists of a collection of components based on the best performing algorithm to solve the RCPSP without activity preemption. The activity selection problem selects the best possible splits for each activity, and makes use of a boolean satisfiability problem solver. The unique contribution of this algorithm is that the algorithm only requires one single activity list to solve the two underlying subproblems as was previously shown in Coelho and Vanhoucke (2011). The paper also proposes an extended version of this problem that allows overlaps between preemptive parts of activities, a feature that is known in the literature as within-activity fast tracking.

We have run a computational experiment on a set of project instances by varying the size of the projects, the topological structure of the network as well as the resource indicators, and made use of different datasets from literature to perform our experiments. Our computational results showed that (1) activity preemption (sometimes) leads to makespan reductions (while a general agreement in literature exists that it mostly does not), (2) the degree of reductions depend on network and resource indicators and (3) the makespan reductions do not require a lot of splits in the activities (not even in the case the activity parts can overlap as is the case for fast tracking).

Contrary to the common belief that allowing activity splitting in the project schedule complicates the scheduling algorithms, results in very complex messy project schedules with a huge number of activity splits and does not lead to makespan reductions, our results show that this conventional wisdom should be put into a different perspective. Our experiment show that resource-feasible schedules can be obtained using a very general algorithm that can be easily extended to a variety of settings (e.g. preemption with setup times, fast tracking, extra restrictions on the setup times) without changing the fundamental logic of the procedure. Moreover, the experiments reveal that adding setup times to activity splits is an easy way to penalise the presence of too many activity splits. Indeed, the obtained Gantt charts in our experiments contain a relatively low number of splits, while significant makespan reductions are still reachable for resource-constrained project scheduling problems. Consequently, activity preemption can sometimes reduce the project makespan even with relatively

high values for the setup times, and this reduction is the highest for projects with a parallel network structure, with an irregular distribution of the activities over the progressive levels, and with low values for the resource strength. The extension to within-activity fast tracking also shows that major improvements in the project makespan can be obtained with a relatively small number of activity splits.

One of the main weaknesses of our study is that we could not show the quality of our solution procedure compared to state-of-the-art results. We could simply not compare our results with previous research results, since no other procedures have been proposed in literature that solve the preemptive RCPSP with the five types of setup times. However, we have compared our algorithm with the RCPSP with preemption but no setup times (PRCPSP) in Table 5 and show that activity preemption can lead to project makespan reductions, even with the presence of setup times. This comparison was of course only to show the percentage of makespan reductions using setup times relative to the maximum possible makespan reduction (the PRCPSP reductions), and does not illustrate the quality of our solution procedure. We also have compared the solutions of our algorithm with the RCPSP without preemption, also in Table 5, to measure the impact of the resource utilisation and show that only small increases can be observed when preemption with and without setup times is introduced in the paper. Finally, we compared our algorithm with priority rules in Table 6 to show that the number of splits is kept relatively low when using our algorithm. Future researchers who wish to develop faster and more efficient algorithms for the preemptive RCPSP with setup times can easily compare their results with ours. The experiments can be replicated since all the settings are described in detail. First, all datasets of Table 3 are publicly available, for which a summary of overview can be found in Vanhoucke et al. (2016). Moreover, Vanhoucke and Coelho (2018) have recently introduced a tool to test and validate algorithm for the resource-constrained project scheduling problem which can be easily extended to the presence of preemption with setup times. Finally, the values for the setup times can be easily calculated using the formulas of Section 2.3.1 and the values of *%s* of Table 3, which makes the comparison between our results and future results unambiguous.

Of course, future research could and should also include the extension of this model to other types of setup times, and an analysis of realistic values for activity setup times should bring more clarity into the real needs of scheduling algorithms with activity splitting and setup times. Moreover, extensions to other environments in which setup times can be used to model e.g. rework, resource learning and other real life behaviour is on the research agenda.

Acknowledgements

We acknowledge the financial support provided by the “Bijzonder Onderzoeksfonds” (BOF) and by the National Bank of Belgium (NBB) for the project with contract number BOF12GOA021 and by the Fundação para a Ciência e Tecnologia with contract number FRH/BSAB/127795/2016. All computational experiments were carried out using the Stevin Supercomputer Infrastructure at Ghent University (Belgium), funded by Ghent University, the Hercules Foundation and the Flemish Government - department EWI.

References

Ballestin, F., Valls, V., Quintanilla, S., 2008. Pre-emption in resource-constrained project scheduling. Eur. J. Oper. Res. 189, 1136–1152.
Bianco, L., Caramia, M., Dell'Olmo, P., 1999. Solving a preemptive project scheduling problem with coloring techniques. In: Weglarz, J. (Ed.), Project Scheduling, volume 14. In: International Series in Operations Research & Management Science. Springer, US, pp. 135–145.
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.
Buddhakulsomsiri, J., Kim, D., 2007. Priority rule-based heuristic for multi-mode resource-constrained project scheduling problems with resource vacations and activity splitting. Eur. J. Oper. Res. 178, 374–390.
Buddhakulsomsiri, J., Kim, D.S., 2006. Properties of multi-mode resource-constrained project scheduling problems with resource vacations and activity splitting. Eur. J. Oper. Res. 175 (1), 279–295.
Coelho, J., Vanhoucke, M., 2011. Multi-mode resource-constrained project scheduling using RCPSP and SAT solvers. Eur. J. Oper. Res. 213, 73–82.
Coelho, J., Vanhoucke, M., 2015. The resource-constrained multi-mode project scheduling problem. In: Schwindt, C., Zimmermann, J. (Eds.), Handbook on Project Management and Scheduling. Springer International Publishing AG, pp. 491–511.
Cook, S., 1971. The complexity of theorem-proving procedures. In: Proceedings of the third annual ACM symposium on Theory of computing, pp. 151–158.
Damay, J., Quilliot, A., Sanlaville, E., 2007. Linear programming based algorithms for preemptive and non-preemptive RCPSP. Eur. J. Oper. Res. 182, 1012–1022.
Davis, M., Logemann, G., Loveland, D., 1962. A machine program for theorem proving. Commun. ACM 5 7, 394–397.
Debels, D., Vanhoucke, M., 2007. A decomposition-based genetic algorithm for the resource-constrained project scheduling problems. Oper. Res. 55, 457–469.
Demeulemeester, E., Herroelen, W., 1992. A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. Manag. Sci. 38, 1803–1818.
Demeulemeester, E., Herroelen, W., 1996a. An efficient optimal solution for the preemptive resource-constrained project scheduling problem. Eur. J. Oper. Res. 90, 334–348.
Demeulemeester, E., Herroelen, W., 1996b. Modelling setup times, process batches and transfer batches using activity network logic. Eur. J. Oper. Res. 89, 355–365.
Demeulemeester, E., Vanhoucke, M., Herroelen, W., 2003. RanGen: a random network generator for activity-on-the-node networks. J. Schedul. 6, 17–38.
Hartmann, S., Briskorn, D., 2010. A survey of variants and extensions of the resource-constrained project scheduling problem. Eur. J. Oper. Res. 207, 1–15.
Herroelen, W., Demeulemeester, E., De Reyck, B., 1999. A classification scheme for project scheduling problems. In: Weglarz, J. (Ed.), Project Scheduling - Recent Models, Algorithms and Applications. Kluwer Academic Publishers, Dortrecht, pp. 1–26.
Kaplan, L., 1991. Resource-constrained project scheduling with setup times. Department of Management. University of Tenessee, Knoxville. Unpublished paper
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.
Krüger, D., Scholl, A., 2009. A heuristic solution framework for the resource constrained (multi-)project scheduling problem with sequence-dependent transfer times. Eur. J. Oper. Res. 197, 492–508.
Krüger, D., Scholl, A., 2010. Managing and modeling general resource transfers in (multi-)project scheduling. OR Spectr. 32, 369–394.
Li, K., Willis, R., 1992. An iterative scheduling technique for resource-constrained project scheduling. Eur. J. Oper. Res. 56, 370–379.
Marques-Silva, J., Sakallah, K., 1999. GRASP: a search algorithm for propositional satisfiability. IEEE Trans. Comput. 48, 506–521.
Mastor, A., 1970. An experimental and comparative evaluation of production line balancing techniques. Manag. Sci. 16, 728–746.
Mika, M., Waligóóra, G., Węglarz, J., 2008. Tabu search for multi-mode resource-constrained project scheduling with schedule-dependent setup times. Eur. J. Oper. Res. 187 (3), 1238–1250.
Mika, M., Waligóra, G., Weglarz, J., 2006. Perspectives in modern project scheduling, volume 92. In: International Series in Operations Research & Management Science, chapter Modelling Setup Times in Project Scheduling. Springer, US, pp. 131–163.
Nudtasomboon, N., Randhawa, S., 1997. Resource-constrained project scheduling with renewable and non-renewable resources and time-resource tradeoffs. Comput. Industr. Eng. 32, 227–242.
Patterson, J., 1976. Project scheduling: the effects of problem structure on heuristic scheduling. Naval Res. Logist. 23, 95–123.
Schwindt, C., Trautmann, N., 2000. Batch scheduling in process industries: an application of resource-constrained project scheduling. OR Spektrum 22 (4), 501–524.
Valls, V., Ballestin, F., Quintanilla, S., 2005. Justification and RCPSP: a technique that pays. Eur. J. Oper. Res. 165 (2), 375–386.
Valls, V., Ballestin, F., Quintanilla, S., 2008. A hybrid genetic algorithm for the resource-constrained project scheduling problem. Eur. J. Oper. Res. 185 (2), 495–508.
Van Peteghem, V., Vanhoucke, M., 2010. A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem. Eur. J. Oper. Res. 201, 409–418.
Vanhoucke, M., 2006. Work continuity constraints in project scheduling. J. Construct. Eng. Manag. 132, 14–25.
Vanhoucke, M., 2008. Setup times and fast tracking in resource-constrained project scheduling. Comput. Industr. Eng. 54, 1062–1070.
Vanhoucke, M., 2010. Measuring time - improving project performance using earned value management, volume 136. International Series in Operations Research and Management Science. Springer.

Vanhoucke, M., Coelho, J., 2016. An approach using SAT solvers for the RCPSP with logical constraints. Eur. J. Oper. Res. 249, 577–591.
Vanhoucke, M., Coelho, J., 2018. A tool to test and validate algorithms for the resource-constrained project scheduling problem. Comput. Industr. Eng. 118, 251–265.
Vanhoucke, M., Coelho, J., Batselier, J., 2016. An overview of project data for integrated project management and control. J. Mod. Project Manag. 3 (2), 6–21.
Vanhoucke, M., Coelho, J., Debels, D., Maenhout, B., Tavares, L., 2008. An evaluation of the adequacy of project network generators with systematically sampled networks. Eur. J. Oper. Res. 187, 511–524.
Vanhoucke, M., Debels, D., 2008. The impact of various activity assumptions on the lead time and resource utilization of resource-constrained projects. Comput. Industr. Eng. 54, 140–154.