Open access • Journal Article • DOI:10.1007/S10951-015-0423-3

A purely proactive scheduling procedure for the resource-constrained project scheduling problem with stochastic activity durations — Source link

Patricio Lamas, Erik Demeulemeester

Institutions: Katholieke Universiteit Leuven

Published on: 01 Aug 2016 - Journal of Scheduling (Springer US)

Topics: Schedule and Robustness (computer science)

A purely proactive scheduling procedure for the resource-constrained project scheduling problem with stochastic activity durations

Patricio Lamas, Erik Demeulemeester
Research Center for Operations Management
Department of Decision Sciences and Information Management
Faculty of Business and Economics
KU Leuven (Belgium)

February 9, 2015

Abstract

The purpose of this research is to develop a new procedure for generating a proactive baseline schedule for the resource-constrained project scheduling problem. The main advantage of this new procedure is that it is completely independent of the reactive policy applied. This contrasts with the traditional methods that assume a predefined reactive policy. First, we define a new robustness measure, then we introduce a branch-and-cut method for solving a sample average approximation of our original problem. In a computational experiment, we show that our procedure outperforms two other published methods, assuming different robustness measures.

Keywords: proactive RCPSP, robust RCPSP, chance-constrained programming, SAA.

1 Introduction

In reality projects are subject to high levels of uncertainty. This might lead to schedule disruptions that make the schedules obtained by solving the traditional resource-constrained project scheduling problem (RCPSP) completely different from the actually executed schedules. That fact triggered the incorporation of uncertainty in the models and methods for solving the RCPSP that were developed during the last decade.

The RCPSP is a complex problem to solve even when deterministic parameters are assumed [8]. Consequently, solving the RCPSP incorporating uncertainty is an extremely challenging problem. Therefore, the published methods for solving the RCPSP under uncertainty have been based on its division into

two easier steps. One step is the generation of an initial proactive robust baseline schedule, which is protected as much as possible against disruptions. The second step aims to define a reactive policy, which is deployed every time a disruption occurs. The drawback of such a division is that it does not consider the dependency between the two steps, given that all the used robustness measures depend on both the proactive schedule and the reactive policy.

The contributions of this work are the following: we introduced a new robustness measure, defined as the probability that the actually executed schedule is identical to the baseline schedule, which is independent of the reactive policy applied. We propose a novel formulation for the RCPSP with stochastic durations. We developed a branch-and-cut algorithm to efficiently find a robust (purely) proactive baseline schedule, considering this new robustness measure.

The baseline schedules built using our method are the most robust solutions for a sample average approximation model, using this new fair robustness measure as an indicator. This fact contrasts with the previously published methods, where the comparison between robustness levels for different schedules is ambiguous and neither an optimality guarantee nor optimality gaps are provided.

The remainder of this paper is organized as follows. In the next section we discuss the methods that have been presented in the literature for solving the RCPSP under uncertainty. Section 3 provides our new problem statement. In Section 4 we present a method for solving this problem. The numerical results of our method are presented in Section 5. The last section offers the conclusions of this work.

2 The RCPSP with stochastic activity durations

The most frequently considered approach in the literature in order to cope with uncertainty is to assume that the activity durations are stochastic variables with a known probability distribution. Two alternative methodologies have been proposed for solving the RCPSP with stochastic activity durations: stochastic RCPSP (SRCPSP) [23, 24, 35] and proactive/reactive RCPSP [21, 42, 43, 44]. The former deals with uncertainty by viewing the scheduling problem as a multistage decision process. A *scheduling policy* is applied dynamically in order to decide which activities have to be started with the objective of minimizing the expected makespan.

It has been pointed out [16, 22] that the SRCPSP has the drawback of not generating a complete baseline schedule before the initiation of a project. The baseline schedule serves very important functions [4, 32]: allocating resources to different activities, quoting competitive and reliable due dates, scheduling the activities in accord with all parties within the inbound and outbound supply chain, determining time windows for work to be done by subcontractors, sharing production schedules with suppliers on a continuous basis using Internet technology, making cash flow projections, measuring the performance of both management and shop floor personnel, and project monitoring and control.

The proactive/reactive RCPSP methodology creates, before the project is

started, a proactive (robust) baseline schedule that is protected as much as possible against the disruptions that may happen. Then, during project execution, when disruptions that cannot be absorbed by the proactive baseline schedule occur, a reactive scheduling procedure is deployed.

Both proactive and reactive procedures have the objective of maximizing some robustness measure. The following measures have been considered:

- Solution stability: This measure refers to the expected difference between the baseline schedule S^B and the actually realized schedule during the project execution S^R (S^B and S^R are the vectors of activity starting times for the baseline and realized schedules, respectively). It is assumed that there is a non-negative cost w_i per unit time overrun or underrun on the start time of each activity i. The lower this measure is, the higher the robustness of the schedule:

$$\Delta\left(S^B, S^R\right) = \sum_i w_i \mathbb{E}\left[\left|S_i^R - S_i^B\right|\right] \tag{1}$$

- Expected makespan: In this case, the schedule will be considered more robust when the expected makespan is shorter.

- Timely project completion probability (TPCP): It measures the probability that the solution value of the realized schedule stays within a certain threshold. Larger values of the TPCP indicate that the schedule is more robust.

The above three robustness measures share the characteristic of depending on the reactive policy. The most competitive proactive methods cope with this dependency by dividing the problem into two steps. First, an early start (reactive) policy is determined assuming deterministic activity durations typically corresponding to its mean (or median) values. Then, in a second step a proactive method creates a robust schedule considering the reactive policy that is fixed in the first step. This two-step approach has been successful in (approximately) solving the problem in reasonable computation times. However, the simplification made in the first step undoubtedly has a negative impact on the quality of the solution reached.

In [12] an algorithm that integrates the proactive and reactive procedures is presented. It creates the optimal proactive schedule for a given reactive policy. Then, a heuristic iteratively modifies such a policy and recalculates the baseline schedule in order to increase the robustness. Clearly, such a procedure does not guarantee optimality. However, to the best of our knowledge, it is the only published method that integrates the proactive and reactive steps.

Based on the above facts, we can conclude the following:

- It is not possible, from a practical point of view, to determine a proactive schedule and a reactive policy that together are optimal.

- The quality of the proactive schedules is determined using robustness measures that depend on the applied reactive policy. Therefore, the comparison among different proactive schedules is not necessarily fair, e.g. a proactive schedule S^1 can be more robust than a proactive schedule S^2 for a given reactive policy Π^1, but S^2 can be more robust than S^1 considering another policy Π^2. In Section A we provide an example that proves this statement.

We propose a new robustness measure that is completely independent of the reactive policy that is planned to be applied, allowing us to develop a method that exclusively focuses on the optimization of the proactive baseline schedule. We call this new measure *confidence level* (CL), which is defined as follows:

$$CL = P\left(S_i^R = S_i^B, \forall i\right) \tag{2}$$

CL measures the joint-probability that each activity i starts exactly at its baseline starting time S_i^B. A reactive policy is never applied if each activity is actually started at its baseline starting time. Therefore CL is independent of any reactive policy that is planned to be applied during the execution of the project. It is assuming that a railway execution policy is applied, i.e. each non-dummy activity will never start earlier than its planned starting time in the baseline schedule.

There are three alternatives for creating a robust proactive schedule based on CL. One is to minimize the project makespan subject to a fixed confidence level. The second option is to maximize CL subject to a given project makespan. A third choice is to use a linear combination of the project makespan and CL as the objective to maximize or minimize. The three alternatives generate the same efficient frontier of solutions, hence they can be considered equivalent. Nevertheless, we choose the first alternative because it can be directly formulated as a chance-constrained (C-C) programming problem.

In general, C-C programming models aim to minimize or maximize an objective function subject to probabilistic constraints. This type of model was introduced in [10]. A description of these models, their solution methods and applications can be found in [7, 38].

3 Problem statement

The problem statement for the chance-constrained RCPSP (C-C RCPSP) is based on the definitions of the traditional (deterministic) RCPSP. Thus, first we present such definitions and a new mathematical formulation for that problem, which is an extension of the formulation introduced in [5]. Then, we present the problem statement for the C-C RCPSP and its corresponding formulation.

3.1 The deterministic RCPSP

An instance of the deterministic RCPSP is defined by the following elements: a set of *activities* $V = \{0, \ldots, n+1\}$, where 0 and $n+1$ are *dummy activities* that

represent the start and the end of the schedule respectively, a vector $d \in \mathbb{N}^{n+2}$ of *activity durations*, with d_i the duration of activity i and $d_0 = d_{n+1} = 0$, a set $E \subseteq V^2$ of *precedence relations*, where $(i, j) \in E$ indicates that activity j can start after the completion of activity i, a set of *renewable resources* $K = \{1, \ldots, k\}$, a vector of *resource availabilities* $B \in \mathbb{N}^k$ with B_k the availability of resource type k and a *resource consumption* matrix $\mathbf{b}$ of dimension $(n + 2) \times |K|$ such that $b_{ik} \in \mathbb{N}$ represents the amount of resource type k used per time period during the execution of activity i.

The (deterministic) RCPSP is the problem of finding a non-preemptive schedule of minimal makespan subject to the precedence and the resource constraints.

Defining the schedule by $S \in \mathbb{N}^{n+2}$, where S_i represents the starting time of activity i, the RCPSP can be conceptually formulated as follows:

$$\min S_{n+1} \tag{3}$$

subject to

$$S_j - S_i \geq d_i \qquad \forall (i, j) \in E \tag{4}$$

$$\sum_{i \in \Gamma_t} b_{ik} \leq B_k \qquad \forall k \in K, t = 0, \ldots, T \tag{5}$$

with $S_0 = 0$ and Γ_t denotes the set of activities that are in progress at time t and T is an upper bound on the project makespan.

It has been proven that the RCPSP is NP-hard in the strong sense by reduction from the 3-PARTITION problem [8].

The following definitions, lemma and propositions have been presented and proven in the project scheduling literature. The interested reader is referred to [6, 28, 29].

A *forbidden set* (introduced in [23, 24]) is a set F of activities (without precedence relations between them) that cannot be in progress simultaneously because of resource limitations due to some resource type k, such that $\sum_{i \in F} b_{ik} > B_k$. A *minimal forbidden set* is a forbidden set such that each of its proper subsets is not a forbidden set. Let Φ be the set of all minimal forbidden sets.

Let $E' \subseteq V^2 \setminus E$ be a relation over V^2. We consider the following requirements for E':

- *C1*: $E \cup E'$ is a strict partial order, i.e., $E \cup E'$ is a transitive and asymmetric relation containing E.
- *C2*: for each minimal forbidden set $F \in \Phi$, $F^2 \cap (E \cup E') \neq \emptyset$, that is, for every $F \in \Phi$ there exist $i \neq j \in F$ such that $(i, j) \in E \cup E'$.

Lemma 1. *A schedule S will be a feasible solution for the RCPSP if and only if S is feasible for the set of relations $E \cup E'$, and conditions C1* and *C2* hold for E'.

In general, an RCPSP instance can be represented in an activity on node (AoN) graph. In this (directed) graph $G(V, E \cup E')$, each activity $i \in V$ has a corresponding node i. Each edge (i, j) represents a direct (i.e. non-transitive) relation $(i, j) \in E \cup E'$. For each edge (i, j) in $G(V, E \cup E')$ a cost $c_{i,j}$ is defined as $c_{i,j} = d_i$.

Let a *selection of edges* $E' \subseteq V^2$ be a set of edges that defines a set of relations $E \cup E'$, such that *C1* holds for E'. Furthermore, a *feasible selection of edges* $E' \subseteq V^2$ is a selection of edges that defines a set of precedence relations such that *C1* holds for $E \cup E'$. Let $l_i(E')$ be the distance of a longest path between node 0 (dummy start activity) and node $i \in V \setminus \{0\}$ in $G(V, E \cup E')$.

Proposition 1. *A schedule S defined as $S_0 = 0$ and $S_i = l_i(E'), \forall i \in V \setminus \{0\}$, with E' a feasible selection of edges, is a feasible solution for the RCPSP.*

A consequence of *Proposition 1* is that the RCPSP can be equivalently stated as follows: The RCPSP is the problem of finding a feasible selection of edges E^* such that the distance of a longest path between dummy activities 0 and $n+1$ in the graph $G(V, E \cup E^*)$ is of minimum length.

3.1.1 Integer programming formulation for the RCPSP

Following [1], we present a mathematical programming formulation for the RCPSP based on the above problem statement. Binary variable x_{ij} is equal to 1 if edge (i, j) is selected and 0 otherwise. Positive integer variable S_i represents the starting time of activity $i \in V$.

$$\min S_{n+1} \tag{6}$$

subject to

$$x_{ij} = 1 \qquad \forall (i,j) \in E \tag{7}$$

$$x_{ij} + x_{ji} \leq 1 \qquad \forall (i,j) \in V^2, i \neq j \tag{8}$$

$$\sum_{(i,j) \in F^2, i \neq j} x_{ij} \geq 1 \qquad \forall F \in \Phi \tag{9}$$

$$S_j - S_i \geq M(x_{ij} - 1) + d_i \qquad \forall (i,j) \in V^2, i \neq j \tag{10}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i,j) \in V^2, i \neq j \tag{11}$$

$$S_i \in \mathbb{Z}^+ \qquad \forall i \in V \tag{12}$$

Objective function (6) represents the starting time of dummy activity $n+1$ which is equivalent to the project makespan. The set of constraints (7) states that for each original precedence relation in E there is a selected edge in the graph. Constraints (8) ensure that the asymmetry condition in *C1* holds. Constraints (9) are sufficient (non-necessary) to ensure that *C2* holds. Constraints (10) relate the starting and finishing times of each two activities connected by an edge (with M any upper bound on the project makespan). Constraints (10) also imply that asymmetry condition in *C1* holds. Constraints (11) and (12)

are the integrality constraints. Given the assumption that the vector of activity durations d is integer, the integrality constraints (12) can be relaxed.

The above formulation is slightly different from the one introduced in [1]. In the latter, the edges represent either direct or transitive precedence relations, whereas in the formulation (6)-(12) the edges represent only direct precedence relations between two activities, even when there is a transitive precedence relation between such activities. An advantage of our formulation is that it does not require the transitive constraints $x_{ij}+x_{jk} \leq x_{ik}+1 \; \forall (i,j,k) \in V^3, i \neq j \neq k$. The total number of these constraints is in the order of $|V|^3$. Therefore, as soon as the number of activities increases up to realistic sizes, the negative impact on the linear programming (LP) solver performance becomes relevant.

An argument in favor of the introduction of the transitive constraints is that they make the formulation stronger, and thus the extra time spent in solving a larger LP problem is compensated by a stronger lower bound. This argument is not necessarily true, specially in the case of difficult instances where the minimal forbidden sets F have a large cardinality. Thus, by constraints (9) the linear relaxation solutions x^*_{ij} generally take values around $1/|F|$, resulting in the fact that the transitive constraints are then not violated.

3.1.2 Other formulations for the RCPSP in the literature

The mathematical programming models for the RCPSP can be divided into three broad categories: sequence-based, time-indexed and event-based formulations [3, 27]. In the sequence-based models, both the sequence of the activities and the starting time of each activity must be determined. In the case of time-indexed formulations, the starting time of each activity is the only type of decision to be made. Finally, in the event-based formulations, an event is defined as the point in time where an activity either starts or ends. Thus, in these type of models the decisions variables correspond to the definitions of the events that each activity starts at.

Formulation (6)-(12) is a sequence-based model. The variables x_{ij} are related to the sequencing decisions and the variables S_i are the starting times of each activity. In the literature only two others sequence-based models can be found in [1] and [2], which are based on the disjunctive graph representation for the job-shop scheduling problem presented in [5].

In the case of the time-indexed formulations, the first mathematical programming model was presented in [39]. Based on that paper, a stronger formulation is introduced in [11]. Three other formulations of this type can be found in [25, 33]. Lagrangean relaxation methods based on this type of formulations are introduced in [17, 34]. A cutting planes algorithm that uses a time-indexed formulation is presented in [19].

In [27] three different event-indexed models are proposed. Additionally, [27] is the only published paper that presents an exhaustive computational comparison among the different mathematical programming formulations available in the literature. More specifically, in their comparison they consider one sequence-based, two time-indexed and three event-based formulations. Their conclusions

are the following: The time-indexed formulations perform the best when the instances have low activity duration ranges. However, in the case of instances with high activity duration ranges, the event-based formulations perform the best and the time-indexed formulations the worst. The sequence-based model consistently performs in between time-indexed and event-based formulations for both type of instances.

An advantage of the sequence-based models is that the formulation of the resource constraints is time-independent [41]. This characteristic makes this type of formulations relatively easy to extend when uncertainty in the activity durations is considered. Unfortunately, in the literature the portion of exact solution methods for the RCPSP that make use of formulations based on the minimal forbidden sets is virtually negligible. This lack of attractiveness is possibly explained by the fact that the total number of minimal forbidden sets is exponential in the number of activities, thus the complete enumeration of all the resource constraints is impractical. We will introduce a delayed constraint generation method that allows us to avoid the initial complete enumeration of all the minimal forbidden sets, making our sequence-based formulation suitable from a practical point of view.

3.2 The chance-constrained RCPSP

The C-C RCPSP under stochastic activity durations is the problem of finding a non-preemptive schedule of minimal makespan such that the precedence and the resource constraints hold with a predefined confidence level. This problem is clearly NP-hard since it is a generalization of the deterministic RCPSP.

Let $(1-\alpha)$ denote the confidence level defined by the decision maker (typically, $1-\alpha = 0.99$ or 0.95). The set of constraints (10) is then replaced by the following constraint:

$$P\left(S_j - S_i \geq M\left(x_{ij} - 1\right) + d_i \quad \forall (i,j) \in V^2, i \neq j\right) \geq 1 - \alpha \tag{13}$$

Constraint (13) states that the probability that all the precedence relations hold is larger than or equal to the confidence level $1-\alpha$. Thus, the C-C RCPSP is formulated as: minimize (6) subject to: (7)-(9), (11)-(13).

This formulation is a simple modification of its deterministic counterpart, which is a consequence of the sequence-based models. However, its solution complexity is amplified due to the addition of randomness.

To the best of our knowledge, [9] is the only previous publication presenting a model that combines C-C programming and the RCPSP. The formulation introduced in that paper differs from ours, since the former is based on an initially fixed priority list policy. A numerical comparison of both formulations is presented in Section 5.

4 Solution method

There are three main difficulties that make the C-C RCPSP model impractical to solve by directly using the mixed integer programming methods:

- *D1*: the feasible region defined by the probabilistic constraint is not convex [31].
- *D2*: the probabilistic constraint is hard to compute. Just checking feasibility is already difficult [31].
- *D3*: the number of minimal forbidden sets is exponential in the number of activities.

Difficulties *D1* and *D2* are tackled by a sample average approximation. *D3* is approached by a branch-and-cut algorithm.

4.1 Sample average approximation

In the sample average approximation (SAA) method presented in [30], the original distribution of the random parameters, in this case the activity durations $d \in \mathbb{N}^{n+2}$, is replaced with an empirical distribution obtained from a random sample. Under some conditions, a feasible solution of the SAA problem will be feasible in the original C-C programming problem with a high probability. The only assumption made on the distribution of the random parameters is that it can be sampled from.

4.1.1 Sample size

A larger sample size implies a better approximation, however a larger sample size also implies a more difficult to solve SAA problem. Thus, the determination of the sample size should strike a balance between approximation quality and SAA problem difficulty. One of the results presented in [30] is the determination of a lower bound for the random sample size, such that the solution of the SAA problem is feasible for the original C-C programming problem.

Let W be the random sample of vector d ($d_0 = d_{n+1} = 0$ for each realization). Let $(1-\epsilon)$ be a confidence level for the SAA problem, such that $\epsilon < \alpha$, let $(1-\theta)$ be the probability that a feasible solution for the SAA problem will yield a feasible solution of the original C-C programming problem and let U be an upper bound on the random activity durations ($d_i \leq U, \forall i \in V$).

The sample size $|W|$ can be determined using the following expression:

$$|W| \geq \frac{1}{2\left(\alpha-\epsilon\right)^2} log\left(\frac{1}{\theta}\right) + \frac{|V|}{2\left(\alpha-\epsilon\right)^2} log\left(U\right) \tag{14}$$

In [30], it has been shown that the above lower bound is too conservative. Those authors suggest to solve the SAA problem with a smaller sample size and

then to use the obtained solution and a larger sample size to estimate the real confidence level reached.

Applying the above method to the RCPSP, in a first step the SAA for the C-C RCPSP is solved with $\epsilon < \alpha$ and a sample size $|W_1|$, giving a solution S. Then, using a second sample W_2 (with $|W_2| \gg |W_1|$) and the solution S, the estimated confidence level reached $1 - \hat{\alpha}$ can be calculated as the number of realizations belonging to W_2 for which solution S is feasible, divided by $|W_2|$.

4.1.2 Sample average approximation formulation

Formally, a *sample* W is a set of realizations of the random activity durations vector, such that $W = \left\{d^1, \ldots, d^{|W|}\right\}$. Let a *scenario* $w \in \{1, \ldots, |W|\}$ be a realization $d^w \in W$. In order to simplify the notation, we will refer to both the sample as well as its set of scenarios as W.

The basic idea of the reformulation introduced in [40] is to solve a problem such that it is infeasible for at most $\lfloor |W| \cdot \epsilon \rfloor$ realizations, thus the solution will be feasible with a confidence level (for the SAA problem) of at least $(1 - \epsilon)$. Based on that idea, we reformulate C-C RCPSP as follows:

Let y_w be a binary variable that takes the value 1 if the obtained solution is not necessarily feasible for scenario w and 0 otherwise.

Probabilistic constraint (13) can then be replaced by the following constraints:

$$S_j - S_i \geq M\,(x_{ij} - 1) + (1 - y_w)\,d_i^w \qquad \forall (i,j) \in V^2, i \neq j, w \in W \tag{15}$$

$$\sum_{w \in W} y_w \leq \lfloor |W| \cdot \epsilon \rfloor \tag{16}$$

$$y_w \in \{0, 1\} \qquad \forall w \in W \tag{17}$$

Constraints (15) state that each precedence relation holds if $y_w = 0$. Constraint (16) imposes that the maximum number of scenarios for which the solution is not necessarily feasible is $\lfloor |W| \cdot \epsilon \rfloor$. Thus, the SAA formulation for the original C-C RCPSP (SAA RCPSP) is: minimize (6) subject to: (7)-(9), (11), (12), (15)-(17).

The lower bound obtained by solving the LP relaxation of the SAA reformulation is weak in general. In [31] a set of strong valid inequalities for general C-C programming problems are presented. The following formulation considers such inequalities.

Let $\delta_i \in \mathbb{N}^{|W|}$ be a vector that contains the durations of activity $i \in V$ for each scenario $w \in W$ sorted in non-increasing order. δ_i^l is the duration of activity i in position $l \in \{1, \ldots, |W|\}$. Additionally, let σ_i^l be the scenario of position l for activity i. Finally, let z_i^l be a binary variable that takes the value 1 if the solution is not necessarily feasible for the duration of activity i at position l in its respective sorted vector, and 0 otherwise.

Thus, we can replace (15) by:

$$S_j - S_i - M\left(x_{ij} - 1\right) \geq \delta_i^1 - \sum_{l=1}^{\lfloor |W| \cdot \epsilon \rfloor} \left(\delta_i^l - \delta_i^{l+1}\right) z_i^l \quad \forall (i,j) \in V \times V, i \neq j \quad (18)$$

and add:

$$z_i^l - z_i^{l+1} \geq 0 \qquad \forall i \in V, l \in \{1, \ldots, \lfloor |W| \cdot \epsilon \rfloor\} \qquad (19)$$

$$z_i^{\lfloor |W| \cdot \epsilon \rfloor + 1} = 0 \qquad \forall i \in V \qquad (20)$$

$$y_{\sigma_i^l} - z_i^l \geq 0 \qquad \forall i \in V, l \in \{1, \ldots, \lfloor |W| \cdot \epsilon \rfloor\} \qquad (21)$$

$$z_i^l \in \{0, 1\} \qquad \forall i \in V, l \in \{1, \ldots, \lfloor |W| \cdot \epsilon \rfloor + 1\} \qquad (22)$$

Constraints (18) and (19) together state that $S_j - S_i - M\left(x_{ij} - 1\right)$ must be larger than or equal to δ_i^l if $z_i^l = 0$. Constraints (20) state that $S_j - S_i - M\left(x_{ij} - 1\right)$ must be at least larger than or equal to $\delta_i^{\lfloor |W| \cdot \epsilon \rfloor + 1}$. Constraints (21) ensure that if a scenario is infeasible for the duration of activity i at position l, then this scenario is infeasible overall. Constraints (22) are the integrality constraints. Thus, the strong formulation for the SAA of the C-C RCPSP (Strong SAA RCPSP) is: minimize (6) subject to: (7)-(9),(11),(12), (16)-(22).

4.2 Branch-and-cut

A Branch-and-Cut (B&C) algorithm allows the introduction of valid inequalities as cutting planes (cuts) in the nodes of the Branch-and-Bound (B&B) tree. Typically, these cuts are introduced in order to make the LP-relaxation-based bound stronger. It normally implies a smaller B&B tree (less memory usage) and faster convergence to the optimum. Examples of applications of this approach are the cover inequalities for the knapsack problem and the comb inequalities for the traveling salesman problem (TSP) [36].

A delayed constraint generation method adds some of the constraints that define the feasible set of solutions only when they are violated. Basically, it initially solves a relaxed problem, which is obtained by leaving out a set of constraints. Then, using the initial solution, it is checked whether there are violated constraints or not. If there are no violated constraints, the solution found is optimal. If there are violated constraints, those are added to the problem and it is solved again. This last step is repeated until no violated constraints are found. For integer programming problems, the method can be applied in the nodes of the B&B tree. In that case, the violated constraints can be added as cuts, and therefore we obtain a B&C algorithm. A well-known example of this approach is the relaxation of the subtour elimination constraints in the TSP and the subsequent dynamic introduction of them as cuts [37].

We designed a B&C algorithm, which is based on a delayed constraint generation method, for solving Strong SAA RCPSP. Since the number of forbidden sets is exponential in the number of activities, the initial introduction of the

complete set of constraints (9) is not practical. Thus, the forbidden set related constraints are inserted as cuts when they are violated. Another positive feature of this approach is that it avoids initially solving the problem of determining all minimal forbidden sets.

4.2.1 Delayed constraint generation

The problem that is initially considered in the root node is a relaxation of Strong SAA RCPSP. We left out all the constraints that are related to minimal forbidden sets of cardinality $\geq \kappa$ in (9) (with κ any integer value larger than or equal to 2) and integrality constraints (11) (the same type relaxation is applied in [13]).

Then, the problem is to check in each (or some selected) node of the B&C tree whether the solution is feasible or not with respect to the resource constraints, or equivalently to check if the constraints (5) hold. If not, it provides one (or more) set(s) of activities $\Gamma_{t'}, t' \in \{0, \ldots, T\}$ that violates the resource constraints for at least one resource (note that $\Gamma_{t'}$ is a forbidden set). In order to separate a non-feasible schedule (with respect to the resource constraints) from the feasible set of solutions, we add the following cut for each $\Gamma_{t'}$:

$$\sum_{(i,j)\in\Gamma^2_{t'}, i\neq j} x_{ij} \geq 1 \tag{23}$$

Note that cuts (23) are equivalent to constraints (9).

Let (S^*, x^*, y^*, z^*) be the optimal solution of the relaxed problem in a node of the B&C tree.

Separation problem for the resource constraints (SP): Do the resource constraints hold for (S^*, x^*, y^*, z^*)? If *yes*, the solution found is feasible (not necessarily feasible for the integrality constraints). If *no*, it provides all the cuts of type (23).

An algorithm for solving SP (see Algorithm 1) is presented. First, some definitions and proofs are needed for its presentation: A *selection of scenarios* $W' \subseteq W$ is a set of scenarios for which a solution (S, x) is feasible. Moreover, a *feasible selection of scenarios* W' is a selection of scenarios such that $|W'| \geq |W| \cdot (1 - \epsilon)$. $W'_\eta = \{w \mid y^*_w = 0, \forall w \in W\}$ denotes a selection of scenarios in a node η of the B&C tree for which the solution found is feasible.

Proposition 2. *A selection of scenarios W'_η in a node η of the B&C tree, such that y^* is integer for η, is a feasible selection of scenarios.*

Proof. Given that $y^*_w \in \{0, 1\}, \forall w \in W$ in node η, the cardinality of W'_η can be calculated as $\left|W'_\eta\right| = \sum_{w\in W}(1 - y^*_w) = |W| - \sum_{w\in W} y^*_w$. But, constraint (16) holds for the solution in node η, then $\sum_{w\in W} y^*_w \leq \lfloor |W| \cdot \epsilon \rfloor \leq |W| \cdot \epsilon$. Thus, $\left|W'_\eta\right| \geq |W| - |W| \cdot \epsilon$. Therefore, W'_η is a feasible selection of scenarios. □

Let $d'(W')$ be the vector of maximum activity durations for a given selection W', such that $d'_i(W') = \max(d^w_i, \forall w \in W'), \forall i \in V$.

Proposition 3. *Given a feasible selection of scenarios W', SAA RCPSP reduces to the deterministic RCPSP with activity durations equal to $d'(W')$.*

Proof. Considering SAA RCPSP, constraints (16) and (17) always hold by definition of feasible selection of scenarios. Set of constraints (15) always holds for $w \notin W'$. For $w \in W'$ constraints (15) become: $S_j - S_i \geq M(x_{ij} - 1) + d_{iw}, \forall (i,j) \in V^2, i \neq j, w \in W'$, then $S_j - S_i \geq M(x_{ij} - 1) + d'(W'), \forall (i,j) \in V^2, i \neq j$. The rest of the constraints and the objective function define the formulation of the deterministic RCPSP. □

Corollary 1. *SAA RCPSP with $\epsilon = 0$ reduces to the deterministic RCPSP with activity durations $d_i = \max(d_i^w, \forall w \in W), \forall i \in V$.*

A consequence of *Proposition 4* is that any algorithm made for the deterministic RCPSP can be applied in each node η of the B&C tree, such that y^* is integer for η.

Given an optimal solution in a node for the relaxed problem $(S^*, d'(W'_n))$, Algorithm 1 determines whether there exists a violated forbidden set or not in $O(|V|^2|K|)$ time. It is based on the observation that the changes in the resource consumption occur only at the starting times or finishing times of the activities. These times define intervals in which the resource consumption is checked for each resource type. In case it is larger than the resource availability for at least one resource type, the algorithm returns *no* and also the sets of activities that violates the resources constraints. A similar algorithm is presented in [2], which is more restrictive since it assumes that the solutions are left-shifted schedules, thus the changes occur only at the finishing times of the activities.

In order to make the linear relaxation stronger, we could insert a cut for each minimal forbidden set belonging to each forbidden set $F = \Gamma_{t'}$, which is equivalent to find all the minimal covers for a set of knapsack constraints. Even if we have an algorithm that could enumerate all minimal covers in a computation time bounded by a polynomial in $|\Phi_F|$ and $|F|$, with Φ_F the set of all minimal covers in F. Nevertheless, $|\Phi_F|$ is exponential in $|F|$.

An alternative might be trying to find the most violated (not necessarily minimal) forbidden set constraint for each set F. However, such a problem is NP-hard. In order to prove the last statement, we formulate the problem as follows:

Let $G(F, E_F)$ be a complete undirected graph, with $|E_F| = \binom{|F|}{2}$. In this graph, both sets F and E_F are weighted. Each node $i \in F$ has a set of weights equal to the resource consumption r_{ik} of activity $i \in F$ for each resource type $k \in K$. Each edge $f = (i,j) \in E_F$ has a weight $\gamma_f = x^*_{ij} + x^*_{ji}$. The optimization problem *OP1* is the problem of finding a clique $C \subseteq F$ of minimum edge weight $\sum_{c \in E_C} \gamma_c$, such that for at least one resource type k, $\sum_{i \in C} r_{ik} \geq B_k + 1$.

Lemma 2. *Problem OP1 is NP-hard.*

Proof. First, let us define the associated decision problem for the above optimization problem. *DP1*: Given a complete undirected graph $G(F, E_F)$, a set of edge weights $\gamma_f, \forall f \in E_F$, a set of node weights r_{ik} for each $i \in F$ and $k \in K$,

Algorithm 1

Input: $(S^*, x^*, d'(W'_n))$ **Output:** *yes* or *no* , forbidden sets

1: Π= vector of finishing times, $\Pi \leftarrow \lfloor S^* \rfloor + d'(W'_n)$
2: Ψ= vector of events, $\Psi \leftarrow MergeVectors(\lfloor S^* \rfloor, \Pi)$
3: sort Ψ in non-decreasing order and eliminate duplicates
4: F= set of activities in progress in time interval $[e, e+1] \forall e \in \{0, \ldots, |\Psi| - 1\}$
5: $feasible$= variable that contains the answer for feasibility check, $feasible \leftarrow yes$
6: $infsets$= vector that contains the violated forbidden sets, $infsets \leftarrow \emptyset$
7: **for** $e = 0 \rightarrow |\Psi| - 1$ **do**
8: $F = \emptyset$
9: **for** $k = 1 \rightarrow |K|$ **do**
10: $req_k = 0$
11: **end for**
12: **for** $i = 1 \rightarrow |V|$ **do**
13: **if** $\lfloor S[i] \rfloor \leq \Psi[e]$ and $\Pi[i] \geq \Psi[e+1]$ **then**
14: add activity i to set F
15: **for** $k = 1 \rightarrow |K|$ **do**
16: $req_k = req_k + b_{ik}$
17: **end for**
18: **end if**
19: **end for**
20: **for** $k = 1 \rightarrow |K|$ **do**
21: **if** $req_k > B_k$ **then**
22: $feasible \leftarrow no$
23: add F to $infsets$
24: **end if**
25: **end for**
26: **end for**
27: return $feasible$,$infsets$

a vector of positive integers B_k and a constant J. Is there any clique $C \subseteq F$ such that $\sum_{i \in C} r_{ik} \geq B_k + 1$ for at least one k and $\sum_{c \in E_C} \gamma_c \leq J$?

Proposition 4. *The CLIQUE problem is a special case of DP1.*

Proof. We restrict DP1 to the CLIQUE problem by allowing only instances in which $\gamma_f = 0$ if edge f exists and $\gamma_f > 0$ otherwise. $|K| = 1$, $r_{ik} = r_i = 1, \forall i \in F$, $B_k = B \leq |F| - 1$ and $J = 0$. Thus, the decision problem is: is there a Clique of cardinality $\geq B+1$?, which corresponds to the CLIQUE problem. □

By *Proposition 5* and given that the CLIQUE problem is NP-complete [18] *DP1* is NP-complete. Therefore *OP1* is NP-hard. □

Given the fact that the separation problem must be solved many times during the B&C algorithm, simpler constraint generation methods seem to be reasonable. One option is to simply add the constraints related to the forbidden sets found with Algorithm 1. Thus, for each forbidden set F the constraint $\sum_{(i,j) \in F^2, i \neq j} x_{ij} \geq 1$ is added if $\sum_{(i,j) \in F^2, i \neq j} x^*_{ij} < 1$ (last condition must be checked when the variables x_{ij} are not all integer). We will refer to this constraint generation method as *CG1*. A second option is to approximately solve *OP1* for each forbidden set F found after running Algorithm 1. We propose a greedy heuristic (see Algorithm 2) for reaching that goal. Basically, in each step a new node is added such that the increase in the total edge weight is minimal. This step is repeated until the resource availability constraints are violated for at least one resource type. After running Algorithm 2, the constraint $\sum_{(i,j) \in C^2, i \neq j} x_{ij} \geq 1$ is added, if $\sum_{(i,j) \in C^2, i \neq j} x^*_{ij} < 1$ for each forbidden set C. We call this constraint generation method *CG2*.

The running time of *CG1* will be smaller than the one of *CG2* since the latter runs both Algorithm 1 and Algorithm 2 while *CG1* only runs Algorithm 1. However, given that $C \subseteq F$, the cuts generated by *CG2* normally will be stronger.

Algorithm 2 Heuristic 1

Input: (F, B, r, γ) **Output:** Clique set C

initialize $C = \emptyset$
sort vector γ in non-decreasing order
add to C the pair of activities $e = (i, j)$ such that γ_e is minimum
while $\sum_{i \in C} r_{ik} \leq B_k, \forall k \in K$ **do**
    for all activity $a \in F \setminus C$ **do**
        A = set of edges of the subgraph composed by nodes $C \cup \{a\}$
        $w_a = \sum_{\alpha \in A} \gamma_\alpha$
    end for
    select activity $a \in F \setminus C$ with the smallest w_a
    add activity a to C
    extract activity a from F
end while
return C

Example

In order to show how the cutting planes algorithm works, let us consider a small instance of the RCPSP. It is composed of 8 non-dummy activities and 1 resource type with availability equal to 8. For ease of exposition, let us consider that a feasible selection of scenarios W' is given and fixed. This assumption does not affect how the delayed constraint generation method essentially works. The durations $d'(W')$ and resource consumptions for each activity, as well as the precedence relations are shown in Figure 1.

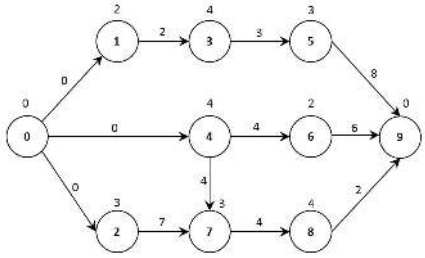

Figure 1: AoN graph representation of the RCPSP instance

The number above each node represents the resource consumption of the respective activity. The number next to each edge represents the duration of the activity corresponding to its tail node. Each edge represents a precedence relation.

Initially, we solve a relaxation of the problem that considers only forbidden sets of cardinality = 2. Algorithm 1 will be applied after an optimal integer solution is found for the relaxed problem. This is not a requirement of our method, but it helps us to graphically explain the procedure. Figure 2 shows the solution found initially.

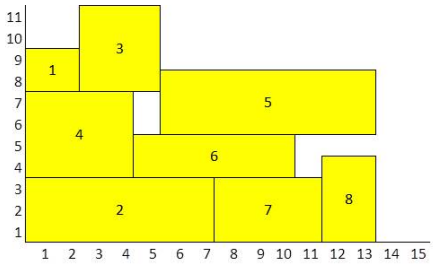

Figure 2: Initial solution

After the solution is found, we have to apply Algorithm 1 in order to test whether this solution is feasible (and optimal) or not. In Figure 3 the vertical lines represent the limits of each of the intervals defined by Algorithm 1.

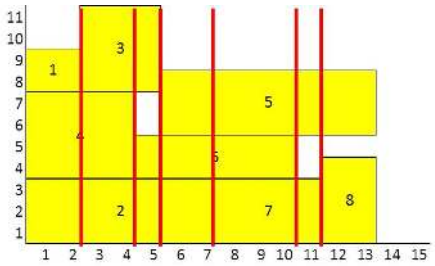

Figure 3: Application of Algorithm 1 to the initial solution

For the first interval, we can see that activities 1, 2 and 4 are in progress. Its total resource consumption is equal to 9, which is over the resource availability of 8, thus $\{1, 2, 4\}$ defines a violated forbidden set. Analogously for the second and third intervals, $\{2, 3, 4\}$ and $\{2, 3, 6\}$ are violated forbidden sets. The rest of the intervals have a total resource consumption that is smaller than or equal to the resource availability.

The algorithm proceeds adding a cut for each of the violated forbidden sets. The goal is that the activities belonging to a violated forbidden set are not all simultaneously in progress. Thus, the following three cuts are added:

$$x_{12} + x_{14} + x_{21} + x_{24} + x_{41} + x_{42} \geq 1$$

$$x_{23} + x_{24} + x_{32} + x_{34} + x_{42} + x_{43} \geq 1$$

$$x_{23} + x_{26} + x_{32} + x_{36} + x_{62} + x_{63} \geq 1$$

Then, the relaxed problem must be resolved including the newly added cuts. The new solution obtained and the intervals defined after the application of Algorithm 1 are shown in Figure 4.

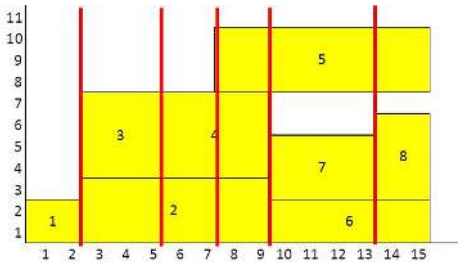

Figure 4: Application of Algorithm 1 to the second solution

The fourth and sixth intervals have a total resource consumption that exceeds the availability of 8, thus $\{2, 4, 5\}$ and $\{5, 6, 8\}$ define violated forbidden sets. Then, the respective cuts are added:

$$x_{24} + x_{25} + x_{42} + x_{45} + x_{52} + x_{54} \geq 1$$

$$x_{56} + x_{58} + x_{65} + x_{68} + x_{85} + x_{86} \geq 1$$

Next, the problem is resolved again, including the newly added cuts. The obtained solution is shown in Figure 5. It is clear that the resource consumption in each time interval is smaller than or equal to the resource availability, thus the cutting planes algorithm stops giving the optimal solution.

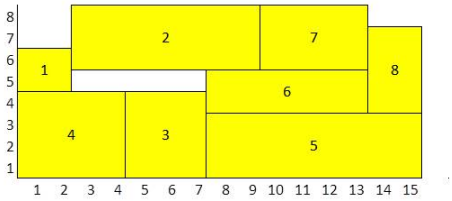

Figure 5: Optimal solution found

4.2.2 Feasible solution

The determination of an initial feasible solution has two main advantages. First, it normally improves the performance of the B&C algorithm as a consequence of a stronger initial upper bound on the objective function. Second, it provides an alternative solution in case the B&C algorithm is unable to find a solution in a reasonable amount of time.

We introduce an algorithm for determining a feasible solution. It is based on the two following observations:

- Given a feasible selection of scenarios W', the SAA RCPSP reduces to the deterministic RCPSP (by *Proposition 4*).
- Given a feasible selection of edges E', the C-C RCPSP (and equivalently the SAA RCPSP) reduces to the chance-constrained longest path problem (C-C LPP) (by *Proposition 2*).

Algorithm 3 sequentially uses these two observations. First, a feasible selection of scenarios is created randomly. Then, the deterministic RCPSP is solved and its solution used as an input for determining a feasible selection of edges. Finally, the SAA for the C-C LPP (SAA LLP) is solved, giving a feasible solution for the SAA RCPSP.

In [3], a polynomial time algorithm is presented for determining a feasible resource flow network given a feasible solution S as input. Each edge with a positive resource flow and not included in the original set of edges E will be in E'. We will refer to that algorithm as *FeasEdgeAlg(S)*.

The solution obtained after solving the RCPSP (S_1 in Algorithm 3) with a random feasible selection of scenarios is already a feasible solution for the SAA RCPSP, however its objective value (S^1_{n+1}) can be arbitrarily bad. It normally will be improved by solving the SAA LPP (lines 3 and 4 of Algorithm 3).

Algorithm 3

1: create a random initial feasible selection of scenarios W'
2: S^1=solve RCPSP($d'(W')$)
3: determine a feasible selection of edges E' using *FeasEdgeAlg(S^1)*
4: S^2=solve SAA LPP($G(N, E \cup E')$)
5: return S^2

Although both the deterministic RCPSP and SAA LPP are NP-hard, practically both problems can be solved more efficiently than the Strong SAA RCPSP (see [31] for an NP-hardness proof of general SAA linear programming problems). For the deterministic RCPSP, efficient exact algorithms have been developed for solving the problem [14, 15, 20]. For the SAA LPP, it could be solved efficiently, given that the deterministic LPP is easy to solve for a directed acyclic graph.

4.2.3 Lower bound

A lower bound for the project makespan (S_{n+1}) can be used to check the quality of the solution obtained. Unlike the feasible solution, this bound is merely an informational tool that will not improve the performance of the B&C algorithm. It is an alternative lower bound that can provide good information in the case that the B&C algorithm stops prematurely (or even is not executed).

Let $\underline{d} \in \mathbb{N}^{n+2}$ be a lower bound of the activity durations vector defined as follows. $\underline{d_0} = \underline{d_{n+1}} = 0$ and $\underline{d_i} = \delta_i^{\lfloor |W| \cdot \epsilon \rfloor + 1}, \forall i \in V \setminus \{0, n+1\}$. Where $\delta_i^{\lfloor |W| \cdot \epsilon \rfloor + 1}$ is the duration of activity i in position $\lfloor |W| \cdot \epsilon \rfloor + 1$ in its respective (non-increasing) sorted vector.

Proposition 5. *The optimal value of the deterministic RCPSP with activity durations equal to $\underline{d}$ is a lower bound for the SAA RCPSP.*

Proof. Relaxing the set of constraints (21) in the strong SAA RCPSP, the problem reduces to the deterministic RCPSP with activity durations equal to $\underline{d}$. □

Example

The purpose of this example is to illustrate the procedures for obtaining a feasible solution and a lower bound. Let us consider the same instance as the previous example, but in this case we will assume that the activity durations (for non-dummy activities) have a discrete uniform probabilistic distribution with limits $(\lfloor 0.5 d_i \rfloor, \lceil 1.5 d_i \rceil)$.

Table 1 shows the values of the activity durations for each realization of a sample W, with $|W| = 10$.

Act	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}
0	0	**0**	0	**0**	0	**0**	0	**0**	0	0
1	3	**3**	3	**2**	2	**3**	3	**1**	1	3
2	10	**5**	9	**6**	8	**6**	7	**11**	11	6
3	2	**3**	4	**3**	3	**2**	5	**4**	5	1
4	4	**3**	6	**6**	6	**5**	4	**2**	4	4
5	7	**7**	5	**9**	12	**6**	6	**5**	9	9
6	7	**9**	4	**6**	5	**4**	9	**7**	9	8
7	3	**4**	4	**2**	4	**5**	6	**4**	6	2
8	1	**1**	3	**3**	2	**3**	2	**2**	2	2
9	0	**0**	0	**0**	0	**0**	0	**0**	0	0

Table 1: Activity durations for each scenario

Considering an SAA confidence level $(1-\epsilon) = 0.6$, we have to find a feasible selection of scenarios W' with cardinality equal to 6. Applying Algorithm 3, we first randomly select $W' = W \setminus \{2, 4, 6, 8\}$. Then, we have to solve the deterministic RCPSP considering deterministic activity durations equal to $d'(W') = (0, 3, 11, 5, 6, 12, 9, 6, 3, 0)$. The schedule obtained is $S^1 = (0, 0, 3, 6, 0, 11, 11, 14, 20, 23)$.

The next step is to find a feasible selection of edges E' using *FeasEdgeAlg(S^1)*. The resulting set E' corresponds to the edges in dashed lines in Figure 6.

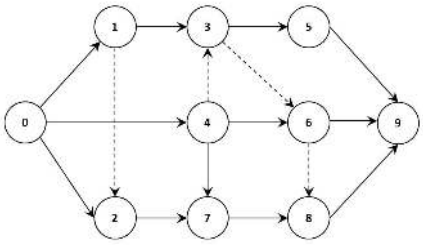

Figure 6: Feasible selection of edges E'

Then, we solve the SAA LPP considering $G(V, E \cup E')$ and the sample W. The new feasible selection of scenarios is $W'^2 = W \setminus \{3, 7, 8, 9\}$ and the final feasible schedule is $S^2 = (0, 0, 3, 6, 0, 9, 9, 13, 18, 21)$, which is two periods shorter than the first schedule S^1. Its corresponding diagram considering the activity durations $d'\left(W'^2\right) = (0, 3, 10, 3, 6, 12, 9, 5, 3, 0)$ is shown in Figure 7.

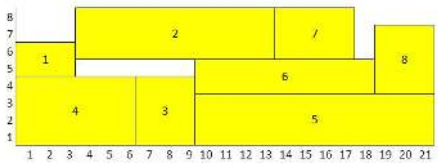

Figure 7: Final feasible schedule S^2

In order to obtain a lower bound for this instance, first we have to sort in non-increasing order the activity durations for each non-dummy activity i, and then determine $\underline{d_i}$. Each element of $\underline{d_i}$ corresponds to the element in position $\lfloor |W| \cdot \epsilon \rfloor + 1 = 4 + 1 = 5$ in the sorted vector. Those values are shown in Table 2.

Act	1	2	3	4	**5**	6	7	8	9	10
0	0	0	0	0	**0**	0	0	0	0	0
1	3	3	3	3	**3**	3	2	2	1	1
2	11	11	10	9	**8**	7	6	6	6	5
3	5	5	4	4	**3**	3	3	2	2	1
4	6	6	6	5	**4**	4	4	4	3	2
5	12	9	9	9	**7**	7	6	6	5	5
6	9	9	9	8	**7**	7	6	5	4	4
7	6	6	5	4	**4**	4	4	3	2	2
8	3	3	3	2	**2**	2	2	2	1	1
9	0	0	0	0	**0**	0	0	0	0	0

Table 2: Sorted activity durations

Following *Proposition 6*, the optimal value of the deterministic RCPSP with activity durations equal to $\underline{d} = (0, 3, 8, 3, 4, 7, 7, 4, 2, 0)$ is a lower bound for this instance. In this case, it corresponds to 17.

Finally, we can determine the relative optimality gap as: $\frac{21-17}{17} = 0.235$.

5 Computational results

All the algorithms that have been presented in the previous sections were computationally implemented and tested. We developed such tests in order to reach four goals. First, to compare our two different constraint generation methods *CG1* and *CG2*. Second, to analyze the impact of the sample size in the performance of our algorithms. Third, to check the performance of the B&C algorithm for different levels of variability on the activity durations. Fourth, to compare our C-C RCPSP approach with two alternative algorithms published in the literature. Supplementary, we include a performance comparison between our B&C algorithm and other mathematical programming based methods for solving the deterministic RCPSP.

The tests were made on 480 instances composed of 30 non-dummy activities and 4 types of resources belonging to the PSPLIB library [26]. Since those instances were created for the deterministic RCPSP, we modified them in the following way: the activity durations d_i for each non-dummy activity i follow a transformation of the beta distribution with shape parameters 2 and 5, and an expected value equal to the duration $\tilde{d}_i$ in the original RCPSP instances. The beta distribution is defined over the (continuous) interval $[0, 1]$, thus we first apply a linear transformation to the generated random variables (with beta(2,5) distribution) and then we round the result of such transformation to the closest integer. We considered three different linear transformations that allow us to generate random variables with three levels of variability: low variability $\left[0.75 \cdot \tilde{d}_i, 1.625 \cdot \tilde{d}_i\right]$, medium variability $\left[0.5 \cdot \tilde{d}_i, 2.25 \cdot \tilde{d}_i\right]$ and high variability $\left[0.25 \cdot \tilde{d}_i, 2.875 \cdot \tilde{d}_i\right]$. Note that the expected value of the random variables is equal to $\tilde{d}_i$ for the three levels of variability. This approach was originally used in [42]. These instances are available in http://feb.kuleuven.be/public/ndbaa92/.

Our B&C algorithm was implemented in C++ using the CPLEX 12.5 API and ran on a personal computer equipped with an Intel® Core™ i7-2720QM 2.20 GHz and 4 Gb RAM. All the remaining algorithms presented in this paper were implemented in C++ and ran on the same computer as well.

5.1 Comparison between CG1 and CG2

The first computational experiment was ran for comparing the performance of the constraint generation methods *CG1* and *CG2*. In the case of *CG1*, the best results were obtained when the cuts were generated only in the nodes where the solution found was integer. For CG2, the best option was to generate the cuts in each node of the B&C tree such that the variables y were integer. For both *CG1* and *CG2* the B&C algorithm started with a feasible solution and a lower bound obtained using Algorithm 3 and *Proposition 6* respectively. The optimal solutions for the deterministic RCPSP were generated using the B&B algorithm presented in [14, 15]. The same sample of size equal to 100 was considered for both methods. The time limit for the B&C algorithms (not including the time needed constructing the initial feasible solution and the initial lower bound) is fixed on 10 seconds (we refer to the next section for a discussion on the sample size and the time limit).

Table 3 shows the obtained results. Column "$1 - \epsilon$" contains the different values for the SAA confidence level. "Opt. solved" contains the total number of instances solved to optimality over the 480 instances. "Opt. gap" shows the average optimality gap for the 480 instances, calculated as $\frac{incumbent - LB}{LB}$, where *incumbent* is the value of the best feasible solution found and LB is the lower bound defined as the maximum value between the lower bound found by Cplex and the one obtained using *Proposition 6*. The column "N. nodes" contains the average number of nodes processed in the B&C tree, over the 480 instances. The results for $1 - \epsilon = 1$ are not included since in those cases the problem can be

solved to optimality using a solver for the deterministic RCPSP (see *Corollary 1*).

		Opt. solved		Opt. gap		N. nodes	
	$1-\epsilon$	CG1	CG2	CG1	CG2	CG1	CG2
low var	0.99	370	323	0.008	0.011	1035	49
	0.95	319	257	0.028	0.034	273	22
	0.9	256	205	0.043	0.048	51	9
	0.8	35	27	0.100	0.101	0	0
	0.7	0	0	0.176	0.176	0	0
med var	0.99	363	301	0.012	0.018	1052	47
	0.95	306	228	0.044	0.057	295	23
	0.9	232	160	0.072	0.081	61	9
	0.8	27	25	0.177	0.173	0	0
	0.7	0	0	0.351	0.351	0	0
high var	0.99	362	307	0.015	0.021	1100	49
	0.95	295	207	0.053	0.069	295	28
	0.9	204	130	0.091	0.101	59	9.9
	0.8	17	13	0.231	0.233	0	0
	0.7	0	0	0.516	0.516	0	0
Average	-	185.7	145.5	0.128	0.133	281.4	16.4

Table 3: Number of instances solved to optimality, average optimality gap and average number of nodes processed for CG1 and CG2

According to the results shown in Table 3, we can conclude that *CG1* performs better than *CG2* . The number of instances that were solved to optimality for *CG1* is strictly larger than for *CG2* for all variabilities and confidence levels, with the exception of $1-\epsilon = 0.7$. Analogous results are obtained considering the average optimality gap, with the exception of medium variability and $1-\epsilon = 0.8$, where *CG2* slightly outperforms *CG1*. As expected, the average number of nodes processed in the B&C tree is smaller for *CG2*. However, the reduction in time due to a smaller search tree does not compensate (on average) the extra time spent on a harder separation problem.

Consequently, all the results of the tests that are presented in the remaining sections were obtained using *CG1* as the constraint generation method for our B&C algorithm.

5.2 Comparison among different sample sizes

Given the combinatorial nature of the C-C programming problems in general, the sample size has an important impact on the execution time of the algorithms designed for solving such type of problems. Our tests show that the C-C RCPSP is no exception. In table 4 we present the number of instances that were solved to optimality (Opt. solved) and the corresponding optimality gaps (Opt. gap) with an execution time limit of 10 seconds (not including the time needed

constructing the initial feasible solution and the initial lower bound), for three different sample sizes (50, 100 and 200) and 5 different SAA confidence levels $(1-\epsilon)$. In the case of $1-\epsilon=1$ it was possible to obtain the optimal solutions for the 480 instances for all sample sizes and variabilities due to *Corollary 1*. We do not include the results for a sample size equal to 50 and $1-\epsilon$ equal to 0.99 and 0.95 given that there does not exist an integer number r such that $\frac{50-r}{50}=0.99$ or 0.95.

		Opt. solved			Opt. gap		
	$1-\epsilon$	50	100	200	50	100	200
low var	1	480	480	480	0.000	0.000	0.000
	0.99	-	370	363	-	0.008	0.011
	0.95	-	319	262	-	0.028	0.036
	0.9	317	256	37	0.030	0.043	0.074
	0.8	239	35	0	0.052	0.100	0.141
	0.7	101	0	0	0.082	0.176	0.177
med var	1	480	480	480	0.000	0.000	0.000
	0.99	-	363	347	-	0.012	0.018
	0.95	-	306	244	-	0.044	0.059
	0.9	294	232	27	0.051	0.072	0.135
	0.8	194	27	0	0.090	0.177	0.290
	0.7	77	0	0	0.130	0.351	0.355
high var	1	480	480	480	0.000	0.000	0.000
	0.99	-	362	350	-	0.015	0.020
	0.95	-	295	211	-	0.053	0.073
	0.9	275	204	18	0.065	0.091	0.172
	0.8	165	17	0	0.118	0.231	0.408
	0.7	50	0	0	0.167	0.516	0.520
Average	-	262.7	234.8	183.3	0.065	0.107	0.138

Table 4: Number of instances solved to optimality and average optimality gap for sample sizes 50, 100 and 200

Clearly, the number of instances solved to optimality decreases when the sample size increases. That impact becomes more relevant when $1-\epsilon$ decreases. The conclusion is opposite when we observe the optimality gaps, i.e. the optimality gaps increase when the sample size increases. Thus, we could conclude that smaller sample sizes allow us to obtain better (closer to the optimum for a given sample) solutions. However, the impact of the sample size on the optimality gaps (and the number of instances solved to optimality) of the algorithm represents only half of the analysis. A complete examination of the obtained solutions should also include a test on its real quality. Given the analytical complexity of the considered probability distributions, the most appropriate option is to obtain an estimation of the real quality of the solutions through simulation.

There is an interdependency between project makespans and confidence lev-

els, therefore we will analyze the efficient frontiers between estimated expected makespans and estimated confidence levels for each sample size. As explained in Section 2, the expected makespan of a given schedule depends on the applied reactive policy. Thus we fix such a policy for each solution. It is defined by the edges with a positive resource flow after applying algorithm *FeasEdgeAlg()* to the solution. We explain the procedure as follows:

Let S^* be the obtained solution (schedule), W be the set of replications of the activity durations in the simulation and E' be the edges with positive flow after applying *FeasEdgeAlg(S^*)*. The simulated starting time of activity i for replication $w \in W$ is defined as $S_i^w = \max\{S_i^*, (S_j^w + d_j^w)\}$ for each activity j such that $(j,i) \in (E \cup E')$. Then, the estimated expected makespan can be calculated by the following expression:

$$\hat{E}[S_{n+1}] = \frac{1}{|W|} \sum_{w \in W} S_{n+1}^w \tag{24}$$

In order to calculate the estimated confidence level, we can define a binary variable z^w that takes a value equal to 1 if the simulated schedule S^w is identical to S^* (i.e. $S_i^w = S_i^*$ for all activities i) and 0 otherwise. Then, the estimated confidence level can be computed as follows:

$$1 - \hat{\alpha} = \frac{1}{|W|} \sum_{w \in W} z^w \tag{25}$$

Our tests were made considering simulations of 1000 replications, for each activity duration variability, SAA confidence level $(1-\epsilon)$ and sample size. Table 5 shows the results of the average estimated confidence level $(1 - \hat{\alpha})$ and the average estimated expected makespan $(\hat{E}[S_{n+1}])$ for sample sizes equal to 50, 100 and 200.

		$1-\hat{\alpha}$			$\hat{E}[S_{n+1}]$		
	$1-\epsilon$	50	100	200	50	100	200
low var	1	0.874	0.951	0.987	78.4	80.8	82.4
	0.99	-	0.943	0.975	-	80.4	81.3
	0.95	-	0.903	0.936	-	78.0	78.5
	0.9	0.760	0.848	0.884	75.1	76.3	76.8
	0.8	0.652	0.735	0.778	73.1	74.1	74.4
	0.7	0.559	0.618	0.668	71.9	72.4	72.6
med var	1	0.797	0.916	0.977	98.2	102.8	106.2
	0.99	-	0.902	0.961	-	102.2	104.6
	0.95	-	0.857	0.914	-	98.7	99.9
	0.9	0.670	0.792	0.857	92.5	95.5	97.1
	0.8	0.568	0.675	0.742	89.1	91.5	92.4
	0.7	0.469	0.555	0.626	86.7	88.1	88.8
high var	1	0.762	0.898	0.969	117.8	124.8	129.6
	0.99	-	0.883	0.953	-	123.7	127.3
	0.95	-	0.831	0.904	-	118.5	121.1
	0.9	0.627	0.769	0.846	110.0	114.5	116.9
	0.8	0.532	0.648	0.725	105.1	108.6	110.3
	0.7	0.432	0.527	0.607	101.7	103.8	105.0
Average	-	0.642	0.792	0.850	91.6	96.4	98.1

Table 5: Average estimated confidence level and average estimated expected makespan for sample sizes equal to 50, 100 and 200

A visual analysis on the efficient frontiers is more suitable in this case. Figure 8 shows the results presented in Table 5. There, it is clear that the solutions obtained with a larger sample size dominate the ones obtained by considering a smaller one. Therefore, we can conclude that it is better to obtain solutions by solving an approximation with a larger sample size, even when the optimality gaps increase due to a more difficult to solve approximation problem.

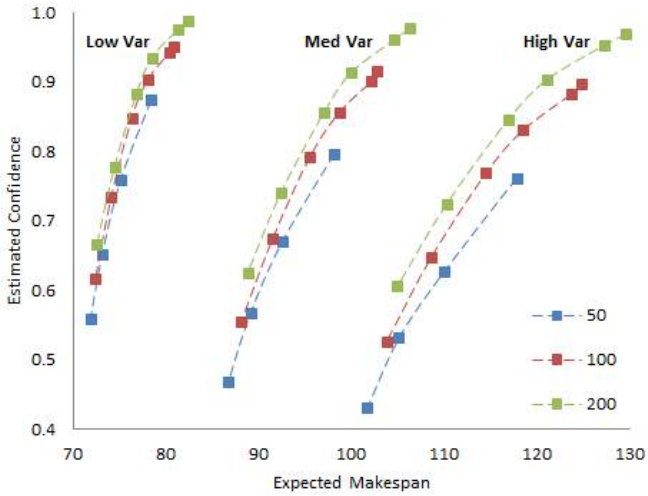

Figure 8: Average estimated confidence level versus average expected makespan for sample sizes 50, 100 and 200

In order to further analyze the impact of the sample size on the quality of the solutions, we will consider sample sizes varying in a wider range: 100, 200, 400, 800, 1600 and 3200. Consequently, we increased the time limit (for the execution of the B&C algorithm) to 600 seconds. Lastly, the sample size for simulations is increased up to 1,000,000.

Running the experiments, considering the new settings, on 480 instances would take (in the worst case) a time of 20 days only for one confidence level. Thus, we will consider a subset of 48 instances (instances "j30X_1", with X=1,..,48; following the notation presented in [26]).

The results of these experiments are presented in Table 6 to Table 9. In Table 6 we can see that the number of instances solved to optimality clearly decreases when the sample size increases from 100 to 3200. However, in Table 7 we can see that the optimality gaps are relatively low, reaching a value of 6.3% in the worst case. In Table 8 we can see that the obtained estimated confidence levels $(1 - \hat{\alpha})$ with a sample size equal to 3200 are practically identical to the real confidence levels $(1 - \alpha)$, hence we can conclude that a sample size of 3200 is enough for attaining a desired confidence level. Such a sample size is radically smaller than 600,000, which is obtained theoretically through (14)(considering $\alpha - \epsilon = 0.01$ and $\theta = 0.99$). The efficient frontiers of the estimated confidence levels (Table 8) and the expected makespan (Table 9) are plotted in Figure 9.

	$1-\epsilon$	100	200	400	800	1600	3200
low var	1	48	48	48	48	48	48
	0.99	43	45	42	40	40	38
	0.95	42	39	38	35	33	27
med var	1	48	48	48	48	48	48
	0.99	42	42	41	42	39	38
	0.95	40	40	38	35	29	24
high var	1	48	48	48	48	48	48
	0.99	43	43	42	40	39	38
	0.95	40	40	37	34	30	24
Average	-	43.8	43.7	42.2	41.1	39.3	37.0

Table 6: Number of instances solved to optimality for sample sizes 100, 200, 400, 800, 1600 and 3200

	$1-\epsilon$	100	200	400	800	1600	3200
low var	1	0	0	0	0	0	0
	0.99	0.002	0.003	0.006	0.006	0.006	0.010
	0.95	0.009	0.013	0.014	0.018	0.025	0.030
med var	1	0	0	0	0	0	0
	0.99	0.004	0.005	0.007	0.008	0.012	0.017
	0.95	0.015	0.017	0.018	0.029	0.035	0.057
high var	1	0	0	0	0	0	0
	0.99	0.004	0.004	0.007	0.011	0.013	0.021
	0.95	0.019	0.018	0.022	0.032	0.050	0.063
Average	-	0.006	0.007	0.008	0.012	0.013	0.022

Table 7: Average optimality gaps for sample sizes 100, 200, 400, 800, 1600 and 3200

	$1-\epsilon$	100	200	400	800	1600	3200
low var	1	0.952	0.985	0.993	0.997	0.997	0.999
	0.99	0.943	0.974	0.986	0.988	0.989	0.991
	0.95	0.907	0.936	0.949	0.953	0.953	0.954
med var	1	0.917	0.975	0.988	0.995	0.997	0.998
	0.99	0.905	0.957	0.977	0.985	0.987	0.988
	0.95	0.860	0.914	0.935	0.946	0.948	0.951
high var	1	0.899	0.967	0.985	0.992	0.995	0.998
	0.99	0.883	0.950	0.973	0.981	0.985	0.988
	0.95	0.829	0.901	0.931	0.940	0.944	0.949
Average	-	0.899	0.951	0.969	0.975	0.977	0.980

Table 8: Average estimated confidence levels for sample sizes 100, 200, 400, 800, 1600 and 3200

	$1-\epsilon$	100	200	400	800	1600	3200
low var	1	79.9	81.3	82.9	83.9	84.8	86.9
	0.99	79.6	80.2	81.0	81.1	80.8	81.0
	0.95	77.1	77.3	77.7	77.7	77.7	77.7
med var	1	101.8	104.9	107.1	109.5	111.1	114.2
	0.99	101.3	102.7	104.7	105.5	105.4	105.9
	0.95	97.5	98.3	98.7	99.2	99.7	100.2
high var	1	123.4	127.6	132.1	134.8	137.8	142.5
	0.99	122.4	125.4	128.2	129.1	129.5	130.2
	0.95	117.0	118.6	120.2	120.7	121.3	121.8
Average	-	100.0	101.8	103.6	104.6	105.3	106.7

Table 9: Average estimated expected makespan for sample sizes 100, 200, 400, 800, 1600 and 3200

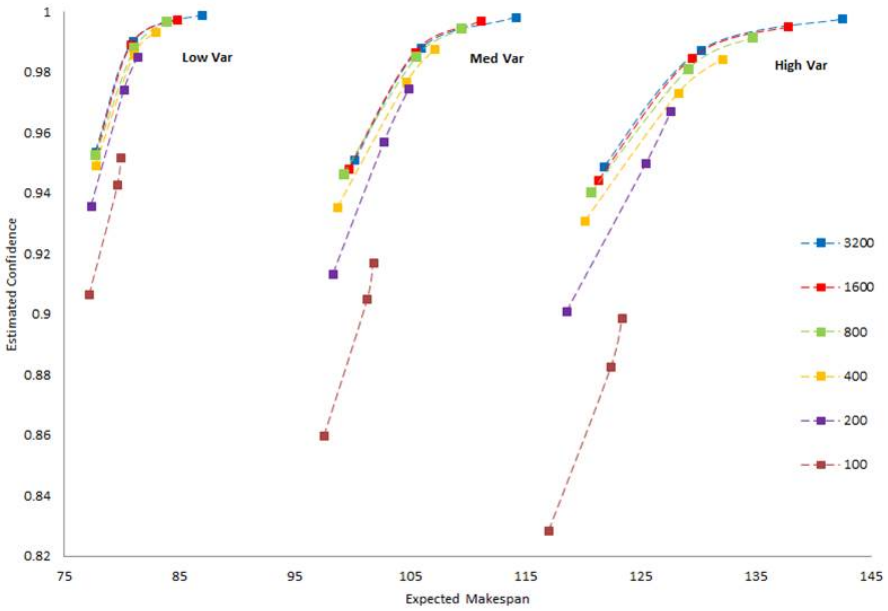

Figure 9: Average estimated confidence level versus average expected makespan for sample sizes 100, 200, 400, 800, 1600 and 3200

In Figure 9 we can observe that generally the solutions that are obtained with a larger sample size dominate the ones obtained through smaller samples. However, the efficient frontiers for sample sizes equal to 800, 1600 and 3200 are virtually identical. Thus, we can conclude that a sample size equal to 800 is enough for obtaining good quality solutions.

Regarding the effect of the activity duration variability, it is clear that the

number of instances solved to optimality decreases when the variability increases. Similarly, the optimality gaps increase when the variability increases. The differences between the quality of the solutions, due to a change in the sample size, are larger when the variability increases.

5.3 Comparison among C-C RCPSP and other methods in the literature

In this section we compare the performance of our method with the starting time criticality heuristic (STC) presented in [42] and the stochastic dynamic generation scheme (SDGS) introduced in [9]. The results of the STC heuristic were obtained running the original C++ implementation of the method on the same computer as for our method. In the case of SDGS, the results were obtained directly from the tables with results that are presented in [9].

5.3.1 Comparison between C-C RCPSP and STC

The STC heuristic has the objective of maximizing the solution stability. Thus, in order to make a fair comparison, we additionally calculated the expected solution stability defined by (1), with $w_i = 1$ for all activities i. Therefore, the expected solution stability can be estimated as follows:

$$\hat{E}[\Delta] = \frac{1}{|W|} \sum_{w \in W} \sum_{i \in V} S_i^w - S_i^* \tag{26}$$

With W, S_i^* and S_i^w defined as in the previous section. Note that $S_i^w - S_i^* \geq 0$ for every activity i and replication w, due to the fact that a railway execution policy is applied.

For both methods we considered the same 1000 replications for the simulations and a sample of size 100 for optimization. Our B&C method was limited to 10 seconds. The execution time for STC is negligible. Table 10 shows the results for this experiment. Figure 10 shows the efficient frontiers of expected confidence level versus expected makespan. There we can see that in general our approach outperforms STC. The best (average) performances of our CCP method are obtained when the variability and the estimated confidence level $1 - \hat{\alpha}$ are low. However, when the variability is high and $1 - \hat{\alpha}$ is close to 90%, STC slightly outperforms CCP. This situation can be explained by the fact that the sample size of 100 is not representative enough. Table 11 contains the results considering a confidence level equal to 1 and a sample size for optimization equal to 1000. Figure 11 shows the results of Table 10 and Table 11 together. There, we can see that our method tends to outperform STC.

		$\hat{E}[S_{n+1}]$		$1-\hat{\alpha}$		$\hat{E}[\Delta]$	
	$1-\epsilon$	CCP	STC	CCP	STC	CCP	STC
low var	1	80.8	80.9	0.951	0.920	0.093	0.115
	0.99	80.4	80.1	0.943	0.897	0.109	0.169
	0.95	78.0	77.6	0.903	0.794	0.188	0.389
	0.9	76.3	75.9	0.848	0.696	0.313	0.633
	0.8	74.1	73.7	0.735	0.561	0.620	1.138
	0.7	72.4	71.9	0.618	0.431	1.065	1.844
med var	1	102.8	103	0.916	0.917	0.186	0.126
	0.99	102.2	101.5	0.902	0.890	0.219	0.190
	0.95	98.7	97.8	0.857	0.809	0.336	0.409
	0.9	95.5	94.5	0.792	0.694	0.536	0.760
	0.8	91.5	90.3	0.675	0.548	0.978	1.358
	0.7	88.1	87.1	0.555	0.426	1.624	2.202
high var	1	124.8	125.3	0.898	0.917	0.281	0.145
	0.99	123.7	123.0	0.883	0.891	0.324	0.219
	0.95	118.5	117.3	0.831	0.785	0.502	0.515
	0.9	114.5	113.0	0.769	0.697	0.756	0.873
	0.8	108.6	107.0	0.648	0.544	1.361	1.676
	0.7	103.8	102.4	0.527	0.417	2.261	2.689
Average	-	96.4	95.7	0.792	0.713	0.653	0.858

Table 10: Average expected makespan, average estimated confidence level and average expected solution stability for CCP and STC

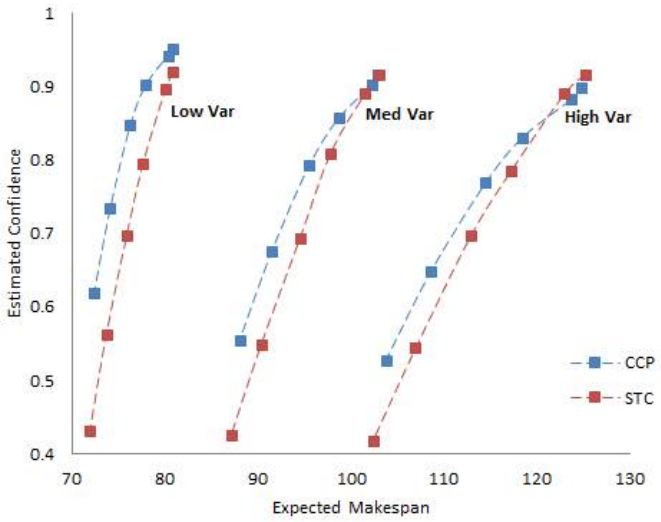

Figure 10: Average estimated confidence level versus average expected makespan

	$\hat{E}[S_{n+1}]$		$1-\hat{\alpha}$		$\hat{E}[\Delta]$	
	CCP	STC	CCP	STC	CCP	STC
low var	84.7	84.3	0.994	0.969	0.008	0.045
med var	111.0	110.1	0.991	0.979	0.014	0.027
high var	136.7	135.0	0.988	0.978	0.023	0.036

Table 11: Average expected makespan, average estimated confidence level and average expected solution stability for CCP and STC with $1-\epsilon = 1$

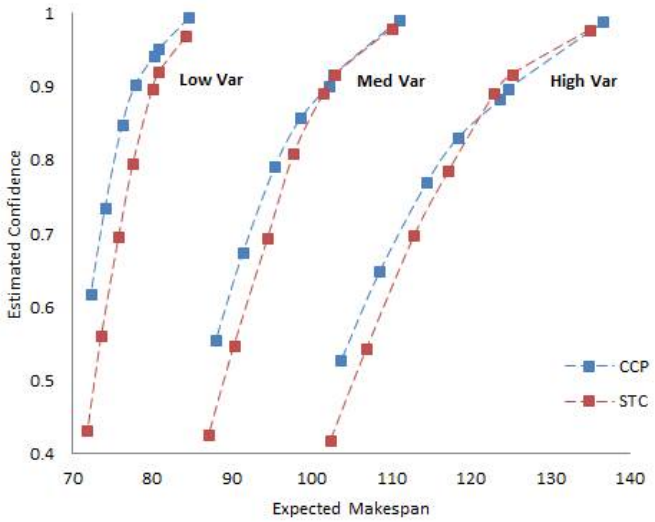

Figure 11: Average estimated confidence level versus average expected makespan (including data in Table 11)

We also show the efficient frontiers of the average expected solution stability versus the average expected makespan (including data in Table 11) in Figure 12. Again, our method tends to outperform STC even in this case, which considers the expected solution stability, a measure that is the objective of STC. The relative drawback of our method is the computation time. It ranges from approximately 1 second (on average) for high confidence levels to 20 seconds (namely, 10 seconds for the initial lower and upper bound procedures and 10 seconds for the B&C procedure) for low confidence levels, whereas STC has an average computation time of approximately 0.2 seconds.

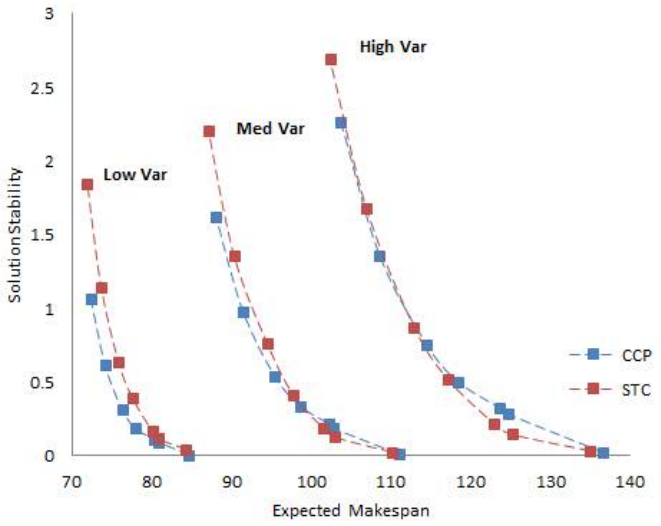

Figure 12: Average expected solution stability versus average expected makespan

5.3.2 Comparison among SDGS, C-C RCPSP and STC

In this section we compare our algorithm with STC and SDGS. The latter is also based on a C-C programming formulation, but in a completely different framework. However, the reported results contain the average estimated confidence levels and the average expected makespans. We used an experimental setting that is as similar as possible to the one presented in [9], considering the same 10 instances of the PSPLIB, activity durations that are Poisson distributed and a sample size equal to 1000 for simulations. For our algorithm and STC we consider a sample size equal to 100 for optimization. The average results (over the 4 variations of SDGS) that were obtained in [9] are presented in Table 12. Table 13 contains the results that we computed for CCP and STC.

$1-\epsilon$	$\hat{E}[S_{n+1}]$	$1-\hat{\alpha}$
0.99	150.6	0.978
0.95	132.4	0.903
0.9	125.4	0.778
0.85	127.4	0.763
0.8	126.1	0.718

Table 12: Average expected makespan and average estimated confidence level for SDGS

	$\hat{E}[S_{n+1}]$		$1-\hat{\alpha}$	
$1-\epsilon$	CCP	STC	CCP	STC
1	116.1	116.1	0.998	0.894
0.99	109.9	108.0	0.994	0.859
0.95	98.6	98.5	0.968	0.812
0.9	93.3	94.7	0.924	0.759
0.8	84.3	85.5	0.861	0.605
0.7	76.7	77.9	0.752	0.519

Table 13: Average expected makespan and average estimated confidence level for CCP and STC

Figure 13 shows that our method clearly outperforms SDGS and STC, and that STC outperforms SDGS. Also, we can see that the trade-off curve obtained for SDGS is not consistent since the point with the shortest makespan does not have the lowest confidence level. The relatively worse performance of SDGS can be explained by the fact that such a method considers a fixed initial priority policy, which in general does not perform well.

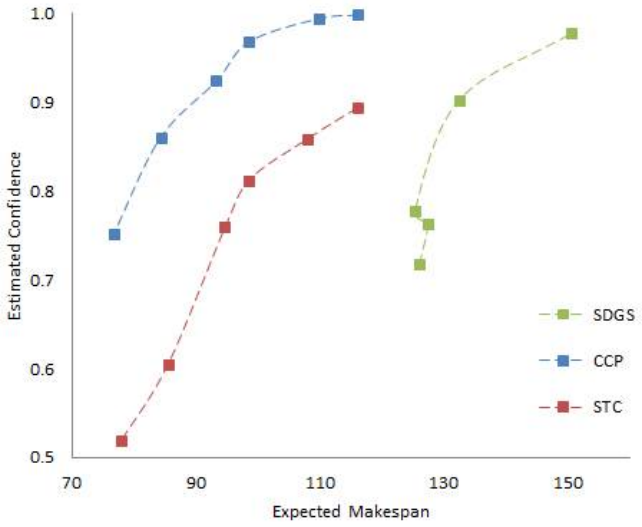

Figure 13: Average expected makespan versus average estimated confidence level

5.4 Comparison among different deterministic RCPSP formulations

A mathematical programming formulation for the deterministic RCPSP is an essential element of the C-C RCPSP. However, the design and comparison of different formulations for the former is beyond the scope of this research. In [27],

a comparative study of different formulations for the RCPSP is presented. In this section we compare the performance of our formulation (6)-(12) (combined with our B&C algorithm) with the results of the tests presented in [27]. Note that the latter results were also obtained by a B&C algorithm, but using the default options of CPLEX. This comparison is not intended to provide definitive conclusions, but rather to highlight the idea that there are potential benefits in applying our algorithm to the classical deterministic RCPSP.

In Table 14, we present the obtained results for our algorithm and the ones provided in [27]. The first row refers to the results of our sequence based approach (SB), which is defined by (6)-(12). The successor rows show the results for the other formulations: basic discrete time (DT), disaggregated discrete time (DDT), flow-based continuous time (FCT), start/end event-based (SEE), on/off event-based (OEE) and on/off event-based with preprocessing (OOE2). Column "% Integer" shows the percentage of instances for which a feasible integer solution was found. Analogously, column "% Opt" contains the percentage of optimal solutions that were found over the complete set of instances. These results were obtained considering a time limit equal to 500 seconds and 480 instances with 30 activities belonging to the PSPLIB library.

Although we ran the tests on a slightly updated hardware (and software), it seems that the combination of formulation (6)-(12) and our B&C algorithm outperforms the standard CPLEX B&C algorithm that was ran considering the other alternative models. More specifically, given that formulation (6)-(12) and FCT are the only two sequence-based formulations that were tested, we can conclude that the omission of the transitive constraints and the delayed generation of the forbidden sets-related constraints (9) are beneficial.

Method	% Integer	% Opt
SB	**100**	**95**
DDT	91	82
DT	86	78
FCT	**67**	**62**
OOE2	46	30
OOE	33	24
SEE	3.1	2.9

Table 14: Percentage of integer feasible and optimal solutions that were found with a time limit of 500 seconds

6 Conclusions

We introduced a new robustness measure that has the advantage of being independent of the applied reactive policy. That fact allowed us to develop a method that is completely focused on the optimization of the proactive baseline schedule. A novel chance-constrained programming formulation was created in order to model the problem considering this new robustness measure.

We developed a B&C algorithm for solving this problem, which took an average time of approximately 1 second for optimally (or approximately) solving the stochastic RCPSP instances with 30 activities and confidence levels close to 1%. As soon as the confidence level decreases, the performance of the algorithm drastically decreases as well. Similarly, the optimality gaps increase when the sample size increases. However, the quality of the solutions is superior when they are obtained considering larger sample sizes. In fact, in our numerical experiments, the solutions that were obtained with smaller sample sizes were virtually always dominated (in terms of efficient frontiers).

As expected, our algorithm in general outperformed two alternative methods published in the literature, considering our new robustness measure. Even more, it tended to outperform the STC method considering a traditional robustness measure (solution stability), which is its objective function.

Finally, we found two contributions from an algorithmic point of view. First, our mathematical programming model and our B&C algorithm can be applied together in order to solve the traditional deterministic RCPSP. Our basic computational experiments showed a good performance when it was compared to alternatives formulations in the literature. Nevertheless, further research is necessary for making the last conclusion definitive. Also, the introduced methods for obtaining lower and upper bounds are not restricted to the RCPSP, therefore they could be applied for solving general mixed integer chance-constrained programming problems in an implicit enumerative method as, for example, a B&B algorithm.

A Appendix 1

The goal of this example is to show how, considering a traditional robustness measure, a baseline schedule S^1 is more robust than S^2 for a given reactive policy Π^1, however S^2 is more robust than S^1 for another reactive policy Π^2.

Let us consider an instance of the RCPSP composed by 5 activities (3 non-dummy activities). The durations of the non-dummy activities 1 and 3 are random variables that can take only a value equal to 1 or 2 with the same probability. For the non-dummy activity 2, the duration is a deterministic value equal to 2. There are no precedence relations between the non-dummy activities. There is only one resource type with an availability equal to 2. Finally, the resource consumption of all the non-dummy activities is equal to 1.

Let us assume that we are given the following two baseline schedules: $S^1 = (0, 0, 1, 1, 3)$ and $S^2 = (0, 1, 1, 0, 3)$. Also, two different reactive priority policies are given in order to calculate the robustness measure under consideration: $\Pi^1 = (1, 2, 3)$ and $\Pi^2 = (3, 2, 1)$. Reactive priority policies as Π^1 and Π^2 are completely defined by a list of activities, such that when a resource conflict occurs, the activities are starting following the priority list. Preemption is not allowed for activities that are in progress. Additionally, we will assume a railway execution policy, i.e. each non-dummy activity will never start earlier than their planned starting time in the baseline schedule. The robustness measure taken

into consideration is solution stability with weights $w_i = 1$ for each activity, which is defined by (1).

Figure 13 contains the schedule diagrams for each baseline schedule, reactive policy and realization of the random duration vector. $S^R\left(S^i, \Pi^i\right)$ represents the realized schedule considering a baseline schedule S^i and a reactive policy Π^i for i equal to 1 or 2.

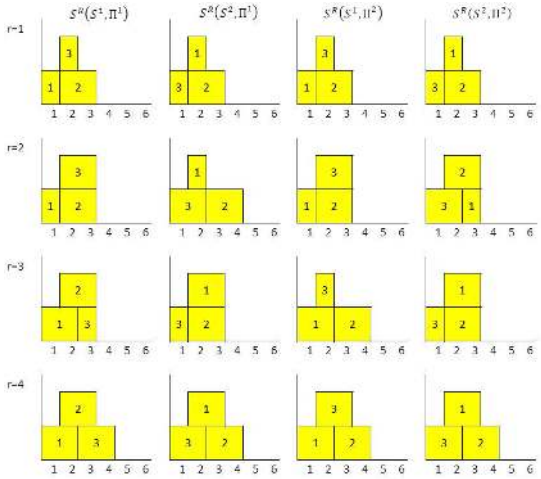

Figure 14: Realized schedule for each realization

Table 8 shows the results for each realization of the random activity durations. In Table 9, $\Delta\left(S^i, \Pi^i\right)$ denotes the solution stability for a given realized schedule and baseline schedule.

r	Dur.	$S^R\left(S^1, \Pi^1\right)$	$S^R\left(S^2, \Pi^1\right)$	$S^R\left(S^1, \Pi^2\right)$	$S^R\left(S^2, \Pi^2\right)$
1	(0,1,2,1,0)	(0,0,1,1,3)	(0,1,1,0,3)	(0,0,1,1,3)	(0,1,1,0,3)
2	(0,1,2,2,0)	(0,0,1,1,3)	(0,1,2,0,4)	(0,0,1,1,3)	(0,2,1,0,3)
3	(0,2,2,1,0)	(0,0,1,2,3)	(0,1,1,0,3)	(0,0,2,1,4)	(0,1,1,0,3)
4	(0,2,2,2,0)	(0,0,1,2,4)	(0,1,2,0,4)	(0,0,2,1,4)	(0,2,1,0,4)

Table 15: Realized schedule for each realization

r	$\Delta\left(S^1, \Pi^1\right)$	$\Delta\left(S^2, \Pi^1\right)$	$\Delta\left(S^1, \Pi^2\right)$	$\Delta\left(S^2, \Pi^2\right)$
1	0	0	0	0
2	0	2	0	1
3	1	0	2	0
4	2	2	2	2
$E[\Delta]$	0.75	1	1	0.75

Table 16: Solution stability for each realization

According to the results shown in Table 9, we could conclude that S^1 is more robust than S^2 using reactive policy Π^1. However, S^2 would be more robust than S^1 considering reactive policy Π^2.

References

[1] Ramon Alvarez-Valdes Olaguibel and JoseManuel Tamarit Goerlich. The project scheduling polyhedron: dimension, facets and lifting theorems. *European Journal of Operational Research*, 67(2):204–220, 1993.

[2] Christian Artigues, Sophie Demassey, and Emmanuel Neron. *Resource-constrained project scheduling: models, algorithms, extensions and applications*, volume 37. Wiley, 2010.

[3] Christian Artigues, Philippe Michelon, and Stéphane Reusser. Insertion techniques for static and dynamic resource-constrained project scheduling. *European Journal of Operational Research*, 149(2):249–267, 2003.

[4] Haldun Aytug, Mark A Lawley, Kenneth McKay, Shantha Mohan, and Reha Uzsoy. Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research*, 161(1):86–110, 2005.

[5] Egon Balas. Machine sequencing via disjunctive graphs: an implicit enumeration algorithm. *Operations research*, 17(6):941–957, 1969.

[6] Martin Bartusch, Rolf H Möhring, and Franz J Radermacher. Scheduling project networks with resource constraints and time windows. *Annals of operations Research*, 16(1):199–240, 1988.

[7] John R Birge and François V Louveaux. *Introduction to stochastic programming*. Springer, 2011.

[8] Jacek Blazewicz, Jan Karel Lenstra, and AHG Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24, 1983.

[9] Maria Elena Bruni, Patrizia Beraldi, Francesca Guerriero, and Erika Pinto. A heuristic approach for resource constrained project scheduling with uncertain activity durations. *Computers & Operations Research*, 38(9):1305–1318, 2011.

[10] Abraham Charnes and William W Cooper. Chance-constrained programming. *Management science*, 6(1):73–79, 1959.

[11] Nicos Christofides, Ramon Alvarez-Valdés, and José M Tamarit. Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research*, 29(3):262–273, 1987.

[12] Filip Deblaere, Erik Demeulemeester, and Willy Herroelen. Proactive policies for the stochastic resource-constrained project scheduling problem. *European Journal of Operational Research*, 214(2):308–316, 2011.

[13] Sophie Demassey, Christian Artigues, and Philippe Michelon. Constraint-propagation-based cutting planes: An application to the resource-constrained project scheduling problem. *INFORMS Journal on computing*, 17(1):52–65, 2005.

[14] Erik Demeulemeester and Willy Herroelen. A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management science*, 38(12):1803–1818, 1992.

[15] Erik Demeulemeester and Willy Herroelen. New benchmark results for the resource-constrained project scheduling problem. *Management Science*, 43(11):1485–1492, 1997.

[16] Erik Demeulemeester and Willy Herroelen. *Robust project scheduling*, volume 3. Now Publishers Inc, 2011.

[17] Marshall L Fisher. Optimal solution of scheduling problems using lagrange multipliers: Part i. *Operations Research*, 21(5):1114–1127, 1973.

[18] Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. Freeman New York, 1979.

[19] Jill R Hardin, George L Nemhauser, and Martin WP Savelsbergh. Strong valid inequalities for the resource-constrained scheduling problem with uniform resource requirements. *Discrete Optimization*, 5(1):19–35, 2008.

[20] Willy Herroelen, Bert De Reyck, and Erik Demeulemeester. Resource-constrained project scheduling: a survey of recent developments. *Computers & Operations Research*, 25(4):279–302, 1998.

[21] Willy Herroelen and Roel Leus. The construction of stable project baseline schedules. *European Journal of Operational Research*, 156(3):550–565, 2004.

[22] Willy Herroelen and Roel Leus. Robust and reactive project scheduling: a review and classification of procedures. *International Journal of Production Research*, 42(8):1599–1620, 2004.

[23] G Igelmund and Franz Josef Radermacher. Algorithmic approaches to preselective strategies for stochastic scheduling problems. *Networks*, 13(1):29–48, 1983.

[24] G Igelmund and Franz Josef Radermacher. Preselective strategies for the optimization of stochastic project networks under resource constraints. *Networks*, 13(1):1–28, 1983.

[25] Robert Klein and Armin Scholl. Computing lower bounds by destructive improvement: An application to resource-constrained project scheduling. *European Journal of Operational Research*, 112(2):322–346, 1999.

[26] Rainer Kolisch and Arno Sprecher. Psplib-a project scheduling problem library: Or software-orsep operations research software exchange program. *European Journal of Operational Research*, 96(1):205–216, 1997.

[27] Oumar Koné, Christian Artigues, Pierre Lopez, and Marcel Mongeau. Event-based milp models for resource-constrained project scheduling problems. *Computers & Operations Research*, 38(1):3–13, 2011.

[28] Roel Leus. Resource allocation by means of project networks: complexity results. *Networks*, 58(1):59–67, 2011.

[29] Roel Leus. Resource allocation by means of project networks: dominance results. *Networks*, 58(1):50–58, 2011.

[30] James Luedtke and Shabbir Ahmed. A sample approximation approach for optimization with probabilistic constraints. *SIAM Journal on Optimization*, 19(2):674–699, 2008.

[31] James Luedtke, Shabbir Ahmed, and George L Nemhauser. An integer programming approach for linear programs with probabilistic constraints. *Mathematical Programming*, 122(2):247–272, 2010.

[32] Sanjay V Mehta and Reha M Uzsoy. Predictable scheduling of a job shop subject to breakdowns. *Robotics and Automation, IEEE Transactions on*, 14(3):365–378, 1998.

[33] Aristide Mingozzi, Vittorio Maniezzo, Salvatore Ricciardelli, and Lucio Bianco. An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation. *Management Science*, 44(5):714–729, 1998.

[34] Rolf H Möhring, Andreas S Schulz, Frederik Stork, and Marc Uetz. Solving project scheduling problems by minimum cut computations. *Management Science*, 49(3):330–350, 2003.

[35] Rolf H Möhring and Frederik Stork. Linear preselective policies for stochastic project scheduling. *Mathematical Methods of Operations Research*, 52(3):501–515, 2000.

[36] George L Nemhauser and Laurence A Wolsey. *Integer and combinatorial optimization*, volume 18. Wiley New York, 1988.

[37] Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991.

[38] András Prékopa. Probabilistic programming. *Handbooks in operations research and management science*, 10:267–351, 2003.

[39] A Alan B Pritsker, Lawrence J Waiters, and Philip M Wolfe. Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science*, 16(1):93–108, 1969.

[40] Andrzej Ruszczyński. Probabilistic programming with discrete distributions and precedence constrained knapsack polyhedra. *Mathematical Programming*, 93(2):195–215, 2002.

[41] Frederik Stork and Marc Uetz. On the generation of circuits and minimal forbidden sets. *Mathematical programming*, 102(1):185–203, 2005.

[42] Stijn Van de Vonder, Erik Demeulemeester, and Willy Herroelen. Proactive heuristic procedures for robust project scheduling: An experimental analysis. *European Journal of Operational Research*, 189(3):723–733, 2008.

[43] Stijn Van de Vonder, Erik Demeulemeester, Willy Herroelen, and Roel Leus. The use of buffers in project management: The trade-off between stability and makespan. *International Journal of Production Economics*, 97(2):227–240, 2005.

[44] Stijn Van de Vonder, Erik Demeulemeester, Willy Herroelen, and Roel Leus. The trade-off between stability and makespan in resource-constrained project scheduling. *International Journal of Production Research*, 44(2):215–236, 2006.