Robust Basic Cyclic Scheduling Problem

Idir Hamaz, Laurent Houssin, Sonia Cafieri

Noname manuscript No.
(will be inserted by the editor)

Robust Basic Cyclic Scheduling Problem

Idir Hamaz · Laurent Houssin · Sonia Cafieri

Received: date / Accepted: date

Abstract This paper addresses the Basic Cyclic Scheduling Problem where the processing times are affected by uncertainties. We formulate the problem as a two-stage robust optimization problem with polyhedral uncertainty set. We propose three exact algorithms for solving the problem. Two of them use a negative circuit detection algorithm as a subroutine and the last one is an Howard's algorithm adaptation. Results of numerical experiments on randomly generated instances show that the Howard's algorithm adaptation yields efficient results and opens perspectives on more difficult robust cyclic scheduling problems.

Keywords Robust optimization · Cyclic Scheduling · Dynamic programming

1 Introduction

Scheduling problems are among the hardest combinatorial problems. In real world applications, a further source of difficulty is represented by the uncertainties on some parameters of the problem at hand. Indeed, the best solution for a deterministic problem can become quickly the worst one in the presence of uncertainties, involving bad schedules and high costs. Many sources of uncertainty can be encountered in scheduling problems, for example activities duration can decrease or increase, machines can breakdown, new activities can be incorporated, *etc.* In this paper, we focus on scheduling problems that are cyclic and where activity durations are affected by the uncertainties.

Several scheduling problems can be indeed simplified by considering them as cyclic. The objective is then to organize the activities by repeating an optimized pattern. Classical scheduling deals with a set of tasks that have to be executed once and optimizes an objective function such as makespan, tardiness or maximum tardiness, *etc.* In contrast, cyclic scheduling deals with a set of generic tasks that have to be executed infinitely. Three objective functions are mainly used in the

Idir Hamaz
7 Avenue du Colonel Roche, 31400 Toulouse
Tel.: +336-58-567363
E-mail: idir.hamaz@laas.fr

domain of the cyclic scheduling. The first objective is the minimization of the *cycle time*, which is the time difference between two successive occurrences of the same operation. The second one is the minimization of the *work-in-process*, which is a bound on the number of jobs executed at the same time. The last one combines the both precedent objectives. In this work, we focus only on the first objective.

Several applications of cyclic scheduling can be found in the literature, e.g. in robotics industry ([1]; [2]), in manufacturing systems ([3]; [4]), in parallel computing and computer pipelining ([5]; [6]; [7]). Depending on the target application, different mathematical models exist, based on graph theory, mixed linear programming, Petri nets or $(max, +)$ algebra. For more details, an overview of cyclic scheduling and different approaches can be found in [8] and [9]. Several heuristic and exact methods have been proposed for cyclic scheduling problems. However, few works consider cyclic scheduling problems under uncertainty. Che et al [10] investigate the cyclic hoist scheduling problem with processing time window constraints where the hoist transportation times are uncertain. The authors define a robustness measure for cyclic hoist schedule and a bi-objective mixed integer linear programming model to optimize the cycle time and robustness.

In this paper, we focus on the Basic Cyclic Scheduling Problem (BCSP) where task durations are affected by uncertainties. The BCSP is a central problem in cyclic scheduling, and represents a basis for modelling and resolving numerous application problems. Several extensions of the problem have been considered in the literature. the BCSP with deadlines is studied in [11]. The author proves the existence of the latest schedule and shows that its computation is generally difficult. In [12], constraints called "linear precedence constraints" are introduced. They generalize the uniform constraints. The linear precedence constraints allow one modelling more application problems, e.g., modelling cyclic assembly line problems where a given number of a first product is needed to build a second product. Note that the BCSP corresponds to the cyclic version of the PERT scheduling problem.

In order to handle uncertainties, we use a robust optimization approach. Robust optimization deals with uncertain problems taking their parameters in a given uncertainty set. The objective is to find a feasible solution within the uncertainty set while optimizing a given criterion such as worst case or maximum regret. Two classes of robust models exist. The first one is the classical static models class. In this case, the decisions have to be taken before knowing any realization of the uncertainty. The second one is the adjustable models class introduced in [13]. In this case, a subset of variables has to be fixed before the uncertainty is revealed and a second subset of variables is allowed to be adjusted and takes into account the uncertainty.

The first study on robust linear programming was published by Soyster [14]. He considers the case where all parameters take their worst-case values. Thus, this model leads to over-conservative solutions. Bertsimas and Sim [15] have extended this framework to reduce the over-conservatism. More precisely, they introduce a parameter called *budget of uncertainty*, that gives a full control on the conservatism of each constraint of the problem.

Robust linear programming with right hand-side uncertainty is a special case of the column-wise uncertainty model proposed by Soyster [14]. More recently, Thiele [16] studies the two-stage robust linear program with uncertainties on the right hand-side and proposes a cutting planes method. Finally, Minoux [17] investigates

the class of two-stage robust linear programming problems with right-hand-side uncertainty and provides some complexity results. He shows that the general case is strongly NP-hard and exhibits some subclasses of polynomially solvable problems. In this paper, we consider the uncertainty set proposed by Bertsimas and Sim [15]. Each task duration belongs to an interval, and the number of parameters that can deviate from their nominal value is bounded by the budget of the uncertainty. This parameter allows us to control the degree of conservatism of the resulting schedule.

In this paper, we focus on the Robust Basic Cyclic Scheduling Problem under the budgeted uncertainty set. We propose a formulation of the problem in accordance with a robust optimization framework. More precisely, a robust two-stage linear program is proposed. A separation problem is then derived using the Farkas' Lemma. In the general case, the latter problem is NP-hard([17]), however using some relevant properties of the solution structure, we show that the problem is equivalent to a negative circuit detection in a particular graph. To solve the problem, we develop three algorithms. Two of them are based on negative circuit detection on a given graph and the last one is an Howard's algorithm [18] adaptation. Finally we perform numerical experiments on several instances to validate and compare the three algorithms.

This paper is organized as follows: Section 2 describes the Basic Cyclic Scheduling Problem. Section 3 introduces the uncertainty set on processing times, and proposes the problem modelling as two-stage robust optimization problem. A separation problem is also derived to build solution algorithms. These algorithms are proposed in Section 4. More precisely, three algorithms are presented for solving the Robust Basic Scheduling Problem. Section 5 describes the experimental methodology and discusses numerical results. Section 6 concludes the paper.

2 Basic Cyclic Scheduling Problem

The Basic Cyclic Scheduling Problem (BCSP) is characterized by a set of n generic operations $\mathcal{T} = \{1, ..., n\}$. Each operation $i \in \mathcal{T}$ has a processing time p_i and must be repeated infinitely often. The k^{th} occurrence of the generic operation i is denoted by $< i, k >$.

A schedule is an assignment of starting time $t(i, k)$ for each occurrence $< i, k >$ of tasks $i \in \mathcal{T}$. A schedule is called *periodic* with cycle time α if it satisfies

$$t(i, k) = t(i, 0) + \alpha k, \quad \forall i \in \mathcal{T},\ \forall k \geq 1. \tag{1}$$

We denote by t_i the starting time of the occurrence $< i, 0 >$. Since the schedule is periodic, a schedule can be completely defined by the vector of the starting times $(t_i)_{i \in \mathcal{T}}$ and the cycle time α.

The operations are subjected to a set of m *precedence constraints* (uniform constraints). Each of these constraints is represented by a quadruple (i, j, p_i, H_{ij}) and is given by

$$t(i, k) + p_i \leqslant t(j, k + H_{ij}), \quad \forall i, j \in \mathcal{T},\ \forall k \geq 1, \tag{2}$$

where i and j are two tasks and H_{ij} is an integer that represents the depth of recurrence, usually referred to as *height*.

Task	1	2	3	4
Processing time	2	1	3	1

Fig. 1: Instance data for Example 1.

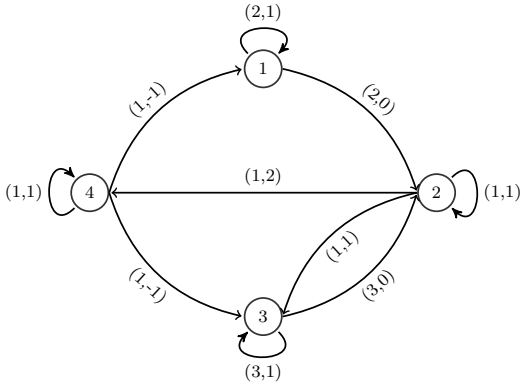

Fig. 2: Uniform graph G associated to the BCSP of Example 1.

Two successive occurrences of the same task i are not allowed to overlap. This constraint corresponds to the non-reentrance constraint and can be modelled as an uniform constraint with $H_{ii} = 1$.

The objective of the BCSP is to find a schedule that minimizes the cycle time α while satisfying precedence constraints. Note that other objective functions can be considered, such as work-in-process minimization or both cycle time and work-in-process minimization.

A directed graph $(G = (\mathcal{T}, E), L, H)$, called *uniform graph*, can be associated with a BCSP such that a node $v \in \mathcal{T}$ (resp. an arc $e \in E$) corresponds to a generic task (resp. uniform constraint) in the BCSP. The function $L : E \mapsto \mathbb{N}$ represents the *length function* and $H : E \mapsto \mathbb{Z}$ the *height function*.

We denote by $L(c)$ (resp. $H(c)$) the length (resp. height) of a circuit c in graph G, representing the sum of lengths (resp. heights) of the arcs composing the circuit c.

Example 1 Fig. 2 illustrates the uniform graph G of the instance of the BCSP described in Fig. 1. The problem has 4 generic tasks and 6 precedence arcs. Each arc (i, j) of G is equipped with two values, the processing time $L_{ij} = p_i$ and and the height H_{ij}.

Let us recall the necessary and sufficient condition for the existence of a feasible schedule.

Theorem 1 (C. Hanen [19]) *There exists a feasible schedule if and only if any circuit of has a positive height.*

In the following, we assume that the graph G satisfies this property. In other words, a feasible schedule always exists.

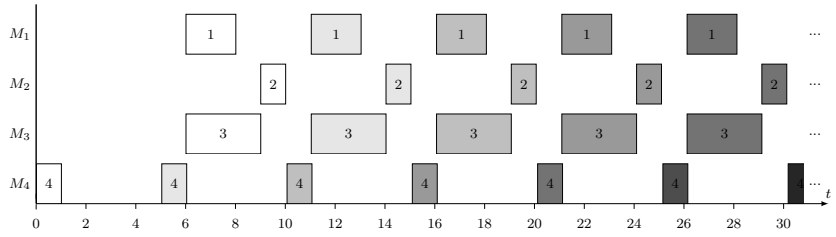

Fig. 3: An optimal periodic schedule associated to the BCSP of Example 1.

The minimum cycle time is given by the maximum circuit ratio of the graph that is defined by

$$\alpha = \max_{c \in \mathcal{C}} \rho(c)$$

where

$$\rho(c) = \frac{L(c)}{H(c)}$$

and $\mathcal{C}$ is the set of all circuits in G.

The circuit c with the maximum circuit ratio is called *critical circuit*. Thus, the identification of the critical circuits in graph G allows one to compute the minimum cycle time.

Example 2 The graph G in Fig. 2 have three circuits, $c_1 = (1, 2, 4, 1)$, $c_2 = (2, 3, 2)$ and $c_3 = (2, 4, 3, 2)$. The associated ratios are respectively, $\alpha_{c_1} = 4$, $\alpha_{c_2} = 4$ and $\alpha_{c_3} = 5$. Thus, the optimal cycle time is $\alpha = \max\{\alpha_{c_1}, \alpha_{c_2}, \alpha_{c_3}\} = 5$ and the critical circuit is $(2, 4, 3, 2)$.

Several algorithms have been proposed for the computation of critical circuits. Gondran and Minoux [20] have proposed a binary search algorithm with time complexity $\mathcal{O}(nm\,(log(n) + log(\max_{(i,j)\in E}(L_{ij}, H_{ij}))))$. An experimental study about maximum mean cycle algorithms was published in [18]. The author remarks that, among the several tested algorithms, the most efficient one is the Howard's algorithm. Although the algorithm has a pseudo-polynomial complexity, it shows noteworthy practical results. This motivates our choice to propose an Howard's algorithm adaptation for the robust version of the BCSP problem.

Once the optimal cycle time α is determined by one of the algorithms cited above, the optimal periodic schedule can obtained by computing the longest path in the graph $G = (\mathcal{T}, E)$ where each arc $(i, j) \in E$ is weighted by $p_i - \alpha H_{ij}$.

The BCSP can be also solved by using linear programming. The problem can be formulated as follows:

$$\min \quad \alpha \tag{3}$$

$$s.t.\ t_j - t_i + \alpha H_{ij} \geq p_i \quad \forall (i, j) \in E \tag{4}$$

where t_i represents $t(i, 0)$, i.e., the starting time of the first occurrence of the task i. Note that the precedence constraints (4) are obtained by replacing in (2) the expression of $t(i, k)$ given in (1).

Example 3 Fig. 3 represents the Gantt diagram of the optimal periodic schedule of the BCSP described in Example 1.

3 Robust Basic Cyclic Scheduling Problem

In this section, we propose a model for the robust version of the BCSP (R-BCSP) where processing times are affected by uncertainties. We first give a brief introduction on two-stage robust linear programming with right hand-side uncertainty. Then we describe the uncertainty set used for modelling uncertain processing times. Finally, we present a formulation of the R-BCSP based on this uncertainty set and derive a separation problem that we use in the solution approach.

3.1 Robust two-stage linear programming with right hand-side uncertainty

Robust optimization problems can be formulated using multiple levels of decisions [13]. In this paper, we focus on two-stage robust optimization. In two-stage formulations, the first-stage variables x have to be fixed before the uncertainty is revealed (without knowing the value of the uncertain parameters), while the second-stage variables y are allowed to adjust themselves and take into account the uncertainty. More precisely, our interest is in robust two-stage linear programming where uncertainties are on the right hand-side.

The problem that we are interested in can be written as follows:

$$\begin{aligned} \min\ & c^T x \\ s.t.\ & Tx + Wy \geq d \\ & x \geq 0, y \geq 0 \end{aligned}$$

where c is the objective vector, T is the first-stage coefficient matrix, W is the second-stage coefficient matrix and d the right-hand-side vector. We assume that the vector d is uncertain and belongs to a given polyhedral uncertainty set. Without loss of generality, the objective of the problem considered here depends only on the first stage variables.

Thiele et al. [16] study this class of problems and propose a cutting-plane algorithm based on Kelley's method. Gabrel et al [21] study a facility location problem under demand uncertainty. The authors formulate the problem as robust two-stage linear program and propose a cutting plane algorithm for solving the problem. More recently, Minoux ([22], [17]) provides complexity results. He shows that the general case is strongly NP-hard and exhibits some subclasses of polynomially solvable problems. The author studies in particular the robust version of the classical PERT scheduling problem. He first presents a two-stage robust formulation. He shows that, for a polyhedral uncertainty set, the problem is equivalent to compute the longest path over the uncertainty set. Finally, he describes a polynomial-time algorithm that solves the problem.

3.2 Problem formulation for the R-BCSP

We define the uncertainty set through the concept of budget uncertainty introduced in [15]. The processing times $(p_i)_{i\in\mathcal{T}}$ are uncertain and belong to the interval $[\bar{p_i}, \bar{p_i} + \hat{p_i}]$, where $\bar{p_i}$ is the nominal value and $\hat{p_i}$ the deviation of the processing time p_i from its nominal value. We associate a binary variable ξ_i to each task $i \in \mathcal{T}$. The variable ξ_i is equal to 1 if task i takes its worst-case value, 0 otherwise. For a given budget of uncertainty Γ (a positive integer representing the maximum number of tasks allowed to take their worst-case values), the uncertain parameters can be modelled as follows:

$$p_i(\xi) = \quad \bar{p}_i + \xi_i \hat{p}_i, \forall \xi \in \Xi^{\Gamma}, i \in \mathcal{T}$$

where

$$\Xi^{\Gamma} = \quad \left\{ (\xi_i)_{i\in\mathcal{T}} \Big| \sum_{i=1}^{\mathcal{T}} \xi_i \leq \Gamma, \xi_i \in \{0,1\} \right\}$$

represents the uncertainty polytope.

In this case, the budget of uncertainty can be considered as an upper bound on the number of deviating tasks. For $\Gamma = n$, the solution is the most conservative. On the other hand, if $\Gamma = 0$, the schedule has no protection against uncertainties.

Example 4 Fig. 5 illustrates a uniform graph G of the the Robust Cyclic Scheduling Problem instance described in Fig. 4. Contrary to the BCSP, the length of each arc (i, j) belongs to an interval $[\bar{L}_{ij} = \bar{p}_i, \hat{L}_{ij} = \hat{p}_i]$. If we consider an uncertainty budget of $\Gamma = 1$, there are 4 scenarios. The worst-case scenario is $\xi = (1, 0, 0, 0)$ and the optimal cycle time is $\alpha = 7$.

Task	1	2	3	4
Processing time	[2,5]	[1,2]	[3,4]	[1,2]

Fig. 4: Instance data for Example 4.

Considering a given cycle time α, the uncertainty on the task durations $(p_i)_{i\in\mathcal{T}}$ can lead to an infeasible schedule. In order to make the schedule robust, we look for the minimum value of the cycle time such that, for each scenario $\xi \in \Xi^{\Gamma}$, there exist a vector $(t_i(\xi))_{i\in\mathcal{T}}$ satisfying the precedence constraints. We model the Robust Basic Cyclic scheduling problem as a two-stage robust optimization problem. The mathematical formulation of the R-BCSP is given below.

$$\min \qquad \alpha \tag{5}$$

$$s.t.\ t_j(\xi) - t_i(\xi) + \alpha H_{ij} \geq p_i(\xi) \quad \forall (i,j) \in E, \forall \xi \in \Xi^{\Gamma} \tag{6}$$

The variable α corresponds to the cycle time. It represents the only first stage variable and must be fixed before knowing the real values of the processing times. The second-stage variables $(t_i(\xi))_{i\in\mathcal{T}}$, which represent the starting times of the first occurrences of the tasks $i \in \mathcal{T}$, are fixed after the uncertainty is revealed. Since the objective function depends only on the first stage variable α, the second stage

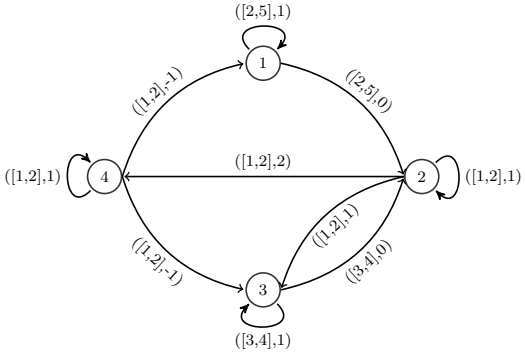

Fig. 5: Associated uniform graph for Example 4.

variables have only to ensure the feasibility of the generic precedence constraints for each scenario $\xi \in \Xi^{\Gamma}$.

Since the PERT scheduling problem and the Basic Cyclic Scheduling Problem have a different structure, the approach of Minoux ([22]) cannot be applied directly to the R-BCSP. In [22], using the special structure of the constraints matrix arising in the PERT scheduling problem, the author proposes a reformulation in terms of paths and shows that the problem can be solved polynomially. In contrast, we formulate a separation problem and propose a separation algorithm that solves it in polynomial time. This separation algorithm is used in Section 4 to address the R-BCSP.

3.3 Separation problem formulation

In the following, we show that we can decide whether a given cycle time $\bar{\alpha}$ is feasible or not in pseudo-polynomial time. Let us reconsider the two-stage robust optimization problem (5)-(6). In order to decide if a given cycle time $\bar{\alpha}$ is feasible or not we define the following separation problem.

For fixed cycle time $\bar{\alpha}$ and given scenario $\bar{\xi} \in \Xi^{\Gamma}$, according to the Farkas' Lemma, the constraints (6) are satisfied if and only if :

$$\sum_{e \in E} (p_e(\bar{\xi}) - H_e \bar{\alpha}) u_e \leq 0 \qquad \forall\, u_e \in C, \tag{7}$$

where C is the polyhedral cone defined by

$$C = \left\{ (u_e)_{e \in E} \;\middle|\; \sum_{e \in \sigma^-(i)} u_e - \sum_{e \in \sigma^+(i)} u_e = 0, \quad \forall\, i \in \mathcal{T} \text{ and } (u_e)_{e \in E} \geq 0 \right\}.$$

Note that $\sigma^-(i)$ and $\sigma^+(i)$ represent respectively the predecessor and the successor of the task $i \in \mathcal{T}$ and $(u_e)_{e \in E}$ are the dual variables of $(t_i)_{i \in \mathcal{T}}$.

Equivalently, the constraints (6) are satisfied for each scenario $\xi \in \Xi^{\Gamma}$ if and only if the optimal objective value of the following bi-linear program

$$\max \quad \sum_{e \in E} (p_e(\xi) - H_e \bar{\alpha}) u_e \tag{8}$$

$$s.t. \quad \xi \in \Xi^{\Gamma} \tag{9}$$

$$\sum_{e \in \sigma^-(i)} u_e - \sum_{e \in \sigma^+(i)} u_e = 0 \; \forall i \in \mathcal{T} \tag{10}$$

$$u_e \geq 0 \qquad \forall e \in E \tag{11}$$

is non-positive.

We can show that the separation problem $(8)-(11)$ is strongly NP-hard [17]. In the following, we exhibit a property that allows us to develop an efficient algorithm.

Observation 1 *A feasible solution of the bi-linear program* $(8)-(11)$ *corresponds to a circulation flow.*

A circulation flow is a flow problem where each node of the associated graph is balanced (the inflow is equal to the outflow). In the following, we recall a property of circulation flows.

Property 1 (Ahuja et al. [23]) *A circulation flow u can be represented as a cycle flow along at most m directed cycles.*

As a result, one can decompose each flow of the problem $(8)-(11)$ into flows along cycles and the cost of each cycle is equal to the sum of arcs composing the cycles. Thus, the problem $(8)-(11)$ can be reformulated as follows:

$$\max_{(u_c)_{c \in \mathcal{C}} \geq 0} \max_{\xi \in \Xi^{\Gamma}} \sum_{c \in \mathcal{C}} (L_c(\xi) - H_c \bar{\alpha}) u_c \tag{12}$$

where u_c is the flow variable along the circuit $c \in \mathcal{C}$. The parameter $L_c(\xi)$ represents the length of the circuit $c \in \mathcal{C}$ for the scenario $\xi \in \Xi^{\Gamma}$ and is defined by $L_c(\xi) = \sum_{i \in c} p_i(\xi)$.

Let us define $A^{\xi}_{\bar{\alpha}}(e) = p_e(\xi) - H_e \bar{\alpha}$ the amplitude of the arc $e \in E$ with respect to $\bar{\alpha}$ and the graph $G^{'} = (G, A^{\xi}_{\bar{\alpha}})$.

Theorem 2 *Let $\bar{\alpha} \in \mathbb{R}$ be a fixed cycle time. Then $\bar{\alpha}$ is feasible if and only if there is no circuit with positive amplitude in the graph $G^{'} = (G, A^{\xi}_{\bar{\alpha}})$.*

Proof Let $\bar{\alpha} \in \mathbb{R}$ be a fixed cycle time.
($\Rightarrow$) Assume that $\bar{\alpha}$ is a feasible cycle time. According to the Farkas' Lemma, the optimal value of (12) is non-positive. Therefore, each circuit in the graph G' has non-positive cost.
($\Leftarrow$) Now, suppose G' has no positive circuit. We also suppose, that $\bar{\alpha}$ is infeasible. According to the Farkas' Lemma, the problem (12) has a positive optimal solution (or unbounded). Therefore, there exists at least one circuit with positive cost, which is in contradiction with the initial assumption.

□

Let us denote the nominal length $\bar{L}$

Theorem 3 *The optimal cycle time α of the R-BCSP is characterized by*

$$\alpha = \max_{c \in \mathcal{C}} \left\{ \frac{\sum\limits_{(i,j)\in c} \bar{L}_{ij}}{\sum\limits_{(i,j)\in c} H_{ij}} + \max_{\xi \in \Xi^{\Gamma}} \left\{ \frac{\sum\limits_{(i,j)\in c} \hat{L}_{ij}\xi_i}{\sum\limits_{(i,j)\in c} H_{ij}} \right\} \right\},$$

where $\mathcal{C}$ is the set of all circuits in G.

Proof Let G be a graph associated to an R-BCSP. We showed in Theorem 2 that a given cycle time α is feasible if and only if there is no positive-amplitude circuit in the graph $G^{'} = (G, A_{\alpha}^{\xi})$. Alternatively,

$$L_c(\xi) - \alpha H_c \leq 0, \, \forall \, c \in \mathcal{C}, \, \forall \, \xi \in \Xi^{\Gamma}$$

where $\mathcal{C}$ is the set of all circuits of the graph $G^{'}$. Then we can deduce that

$$\alpha \geq \frac{L_c(\xi)}{H_c}, \, \forall \, c \in \mathcal{C}, \, \forall \, \xi \in \Xi^{\Gamma}.$$

Hence, the optimal value of the cycle time α is achieved when

$$\alpha = \max_{c \in \mathcal{C}} \left\{ \frac{\sum\limits_{(i,j)\in c} \bar{L}_{ij} + \max\limits_{\xi \in \Xi^{\Gamma}} \left\{ \sum\limits_{(i,j)\in c} \hat{L}_{ij}\xi_i \right\}}{\sum\limits_{(i,j)\in c} H_{ij}} \right\}.$$

□

Corollary 1 *Let P be the value of the longest circuit in terms of the number of nodes. The optimal cycle time for R-BCSP under the uncertainty set Ξ^{Γ} takes the same value for all $\Gamma \in [P, n]$.*

Proof Let $(G = (\mathcal{T}, E), L(\xi), H)$ be an associated graph to an R-BCSP problem under the uncertainty set Ξ^{Γ} and $\mathcal{C}$ the set of all circuits of the graph. Suppose Γ is greater than P, the longest circuit of the graph in terms of nodes. Then, according to Theorem 3, the optimal cycle time α can be expressed as follows:

$$\alpha = \max_{c \in \mathcal{C}} \left\{ \frac{\sum\limits_{(i,j)\in c} \bar{L}_{ij}}{\sum\limits_{(i,j)\in c} H_{ij}} + \max_{\xi \in \Xi^{\Gamma}} \left\{ \frac{\sum\limits_{(i,j)\in c} \hat{L}_{ij}\xi_i}{\sum\limits_{(i,j)\in c} H_{ij}} \right\} \right\}.$$

Since the longest circuit has P nodes, allowing more than P deviations does not affect the following term:

$$\max_{\xi \in \Xi^{\Gamma}} \left\{ \frac{\sum\limits_{(i,j)\in c} \hat{L}_{ij}\xi_i}{\sum\limits_{(i,j)\in c} H_{ij}} \right\}.$$

Thus, the optimal cycle time for R-BCSP under the uncertainty set Ξ^{Γ} takes the same value for all $\Gamma \in [P, n]$. □

In the Basic Cyclic Scheduling Problem, the cycle time can be bounded (see [18]). In the following, we extend the lower and upper bounds of the Basic Cyclic Scheduling Problem to the case of the uncertainty set Ξ^{Γ}. We denote by $\alpha^*_{\min}$ (resp. $\alpha^*_{\max}$) the optimal cycle time where all processing times are fixed to the minimum (resp. maximum), $\bar{L}_{max}$ the maximum arc nominal length and $\hat{L}_{max}$ the maximum arc deviation length.

Proposition 1 *Let us consider a R-BCSP instance and α^* is the associated optimal cycle time under the uncertainty set Ξ^{Γ}, where $0 \leq \Gamma \leq n$. Then,*

$$\bar{L}_{max} \leq \alpha^*_{\min} \leq \alpha^* \leq \alpha^*_{\max} \leq n\bar{L}_{max} + \Gamma\hat{L}_{max}.$$

Proof Let us consider an R-BCSP problem under the uncertainty set Ξ^{Γ} where $0 \leq \Gamma \leq n$.

It is easily seen that the optimal cycle time is greater than the cycle time where $\Gamma = 0$ and lower than the cycle time where $\Gamma = n$.

The second lower bound holds from the non-reentrance constraints. More precisely, a cycle time cannot be lower than the difference between the starting times of two successive occurrences of the same task. This value is bounded by $\bar{L}_{max}$, which represents the maximum nominal duration. Finally, the second upper bound concerns the case where tasks belong to the critical circuit and the height of the critical circuit is equal to 1. Hence, the sum of the nominal values is bounded by $n\bar{L}_{max}$ and the sum of the Γ worst deviations is bounded by $\Gamma\hat{L}_{max}$ □

3.4 Separation procedure

We develop an algorithm detecting, for a given cycle time $\bar{\alpha}$, circuits having positive amplitude $A^{\xi}_{\bar{\alpha}}$ according to a given scenario $\xi \in \Xi^{\Gamma}$. To exploit the negative circuit detection algorithms (see [24]), instead of looking for a positive-amplitude circuit in a graph $G^{'} = (G, A^{\xi}_{\bar{\alpha}})$, we look for a negative-amplitude circuit in a graph $G^{'}_{neg} = (G, -A^{\xi}_{\bar{\alpha}})$. Several algorithms exist for negative circuit detection. They combine shortest path algorithms with negative circuit detection strategies. We examined two algorithms for negative circuit detection. The first one is the Bellman-Ford algorithm and the second one is the Floyd-Warshall algorithm. We performed numerical experiments, and it appears that the Floyd-Warshall algorithm is better than the Bellman-Ford algorithm in terms of running time. Thus, we keep only the Floyd-Warshall algorithm for our numerical experiments.

Let us denote by $d^k_{ij}[\gamma]$ the value of the shortest path from i to j with uncertainty budget γ ($\gamma \in 1, ..., \Gamma$) such that intermediate vertices on the path are chosen from the set $\{1, 2, ..., k\}$. Then, the recursion adapted to the case under investigation can be expressed as follows:

$$\begin{aligned} &\forall\, i, j \in \mathcal{T}, \gamma \in 1, ..., \Gamma : \\ &d^{k+1}_{ij}[\gamma] = \min \min_{0 \leq l \leq \gamma} \left\{ d^k_{ij}[\gamma],\, d^k_{ik}[\gamma - l] + d^k_{kj}[l] \right\}. \end{aligned} \tag{13}$$

A pseudo-code of the adapted Floyd-Warshall algorithm is given in Algorithm 1, where at each iteration we reuse for variable $d^{k+1}_{ij}[\gamma]$ the same space as $d^k_{ij}[\gamma]$, so we neglect the iteration index k. Note that $\bar{c}$ and $\hat{c}$ represent respectively the nominal

Algorithm 1 Modified Floyd-Warshall algorithm

input : A digraph $G = (\mathcal{T}, E)$, a budget of uncertainty $\Gamma \in [0, |\mathcal{T}|]$ and weights $\bar{c} : E \mapsto \mathcal{R}$, $\hat{c} : E \mapsto \mathcal{R}$.
output: report whether a negative circuit exists or not.

```
\\ Initialization
 d_ij[γ] ← ∞            , ∀ i ≠ j, γ ← 0, ..., Γ;
 d_ii[γ] ← 0            , ∀ i ← 1, ..., n, γ ← 0, ..., Γ;
 d_ij[0] ← c̄_{i,j}      , ∀ (i, j) ∈ E;
 d_ij[γ] ← c̄_{i,j} + ĉ_{i,j} , ∀ (i, j) ∈ E, γ ← 1, ..., Γ;
\\ Distance computation
 for k ← 1, ..., n do
    for i ← 1, ..., n do
        for j ← 1, ..., n do
            for γ ← 0, ..., Γ do
                if d_ij[γ] > min_{0≤l≤γ} {d_ik[γ − l] + d_jk[l]} then
                    d_ij[γ] ← min_{0≤l≤γ} {d_ik[γ − l] + d_kj[l]};
                    pred_ij[γ] ← k;          ▷ pred_ij[γ] is a predecessor pointer
                    ind_ij[γ] ← arg min_{0≤l≤γ} {d_ik[γ − l] + d_kj[l]};
                end
            end
        end
    end
end
\\ Negative circuit checking
 for i ← 1, ..., n do
    if d_ii[Γ] < 0 then
        return ("A negative circuit exists");
    end
end
```

value and the deviation value of an arc. The value $\bar{c}$ corresponds to $\bar{p}_e - H_e\bar{\alpha}$ and $\hat{c}$ to $\hat{p}_e$ in the R-BCSP.

The computational complexity of the adapted Floyd-Warshall algorithm can be computed as follows.

Proposition 2 *The modified Floyd-Warshall algorithm runs in $\mathcal{O}(n^3\Gamma^2)$ time.*

Proof Each value $d_{ij}[\gamma]$ can be computed in $\mathcal{O}(\Gamma)$ time. Since i, j and k all iterate from 1 to n and γ from 0 to Γ, the total complexity is $\mathcal{O}(n^3\Gamma^2)$. □

Note that both the negative circuit and the associated scenario can be extracted using the matrix $pred_{ij}[\gamma]$ and $ind_{ij}[\gamma]$.

4 Resolution approaches for the R-BCSP

We present three algorithms for solving the Robust Basic Cyclic Scheduling Problem. Two of them are based on the separation procedure defined in Section 3 and the last algorithm is an adaptation of the Howard's algorithm.

4.1 Negative circuits cancelling algorithm

Algorithm 2 Negative circuits cancelling algorithm (NCC)

Step 1 : Start with a lower bound α_{lb}.
Step 2 : Run the negative circuit detection algorithm with G'_{neg}.
Step 3 : **If** a negative circuit c is detected **Then**
Let ξ be the scenario realizing the negative circuit c.
update α_{lb} to the circuit ratio of c ($\alpha_{lb} = \frac{L_c(\xi)}{H_c}$).
return to **step 2**.
Else the cycle time α_{lb} is optimal.

To solve the Robust Basic Cyclic Scheduling Problem, we develop an iterative algorithm based on the modified Floyd-Warshall algorithm. A pseudo-code of the procedure is presented in Algorithm 2.

The algorithm starts with a lower bound α_{lb} on the optimal cycle time. Then the negative circuit detection algorithm is executed on the graph $G'_{neg} = (G, -A^{\xi}_{\bar{\alpha}})$. If a negative circuit is detected, then the lower bound $\bar{\alpha}$ is updated to the mean value of this circuit. The algorithm stops when there is no more negative circuit in the graph G'_{neg}.

Proposition 3 *The negative circuits cancelling algorithm runs in $\mathcal{O}(n^3\Gamma^2|\mathcal{C}|)$ time, where $|\mathcal{C}|$ is the number of simple circuits in the graph G'_{neg}.*

Proof The complexity of the negative cycle detection algorithm is $\mathcal{O}(n^3\Gamma^2)$. This subroutine is executed until there is no negative cycle in the graph G'_{neg}. Then, the number of passes is bounded by $|\mathcal{C}|$, the number of simple circuits of G'_{neg}. Consequently, the algorithm runs in $\mathcal{O}(n^3\Gamma^2|\mathcal{C}|)$. □

4.2 Binary search algorithm

This algorithm performs a binary search over the possible values for the optimal cycle time α_{opt}. The search interval is $[\alpha_{lb}, \alpha_{ub}] = [\bar{L}_{max}; n\bar{L}_{max} + \Gamma\hat{L}_{max}]$ (see Proposition 1). The algorithm sets α_{mid} to the middle of the interval, then invokes the negative circuit detection algorithm on the graph $G'_{neg} = (G, -A^{\xi}_{\alpha_{mid}})$. If a negative circuit is detected, then the interval search is reduced to $[\alpha_{mid}, \alpha_{ub}]$, otherwise, the next interval search is $[\alpha_{lb}, \alpha_{mid}]$. These iterations are repeated until the interval becomes small enough. A pseudo-code of the algorithm is given in Algorithm 3.

Proposition 4 *The binary search algorithm algorithm runs in $\mathcal{O}(n^3\Gamma^2 \log(n\bar{L}_{max} + \Gamma\hat{L}_{max}))$ time.*

Proof The complexity of the binary search algorithm is equal to the complexity of the negative circuit detection algorithm, which is equal to $\mathcal{O}(n^3\Gamma^2)$, multiplied by the number of passes over the interval search with is bounded by $\mathcal{O}(\log(n\bar{L}_{max} + \Gamma\hat{L}_{max}))$ Consequently, it runs in $\mathcal{O}(n^3\Gamma^2 \log(n\bar{L}_{max} + \Gamma\hat{L}_{max}))$. □

Algorithm 3 Binary search algorithm (BS)

Compute a lower bound α_{lb} and an upper bound α_{ub} on the optimal cycle time α_{opt}
while $ub - lb > \epsilon$ **do**
  $\alpha_{mid} \leftarrow \frac{\alpha_{lb}+\alpha_{ub}}{2}$
  Run negative circuit detection algorithm with G'_{neg}

  if *no negative circuit* **then**
    $\alpha_{ub} \leftarrow \alpha_{mid}$
  else
    $\alpha_{lb} \leftarrow \alpha_{mid}$
  end
end
if $\alpha_{lb} > \alpha_{ub}$ **then**
  the problem is infeasible
else
  $\alpha_{opt} \leftarrow \alpha_{lb}$
end

4.3 Howard's algorithm adaptation

The Howard's algorithm, proposed in [25], is originally designed for Markov decision processes. The algorithm was adapted for maximum circuit ratio computation in [26].

Algorithm 4 Robust Howard's algorithm (R-HOW)

for $i \leftarrow 1, ..., n$ **do**
  for $\gamma \leftarrow 0, ..., \Gamma$ **do**
    $d_i^{\gamma} \leftarrow \bar{L}_i$;
  end
  $\sigma^+(i) \leftarrow i$;
end
while *True* **do**
  $< \alpha_{lb}$,h$>\leftarrow$ compute maximum circuit ratio(G_{σ^+}); $\triangleright$ `h is a given node that belongs to the critical circuit`
  if $h \neq$ *Nil* **then**
    if $\exists$ *a path from i to h in* G_{σ^+} **then**
      $d_i^{\gamma} \leftarrow \max\{d_{\sigma^+(i)}^{\gamma-1} + \bar{L}_{i\sigma^+(i)} + \hat{L}_{i\sigma^+(i)} - \alpha_{lb}H_{i\sigma^+(i)};\ d_{\sigma^+(i)}^{\gamma} + \bar{L}_{i\sigma^+(i)} - \alpha_{lb}H_{i\sigma^+(i)}\}$;
    end
  end
  changed $\leftarrow$ False;
  for *each arc* $(i,j) \in E$ **do**
    for $\gamma \leftarrow 0, ..., \Gamma$ **do**
      if $d_i^{\gamma} < \max\{d_j^{\gamma-1} + \bar{L}_{ij} + \hat{L}_{ij} - \alpha_{lb}H_{ij} - \epsilon;\ d_j^{\gamma} + \bar{L}_{ij} - \alpha_{lb}H_{ij} - \epsilon\}$ **then**
        $d_i^{\gamma} \leftarrow \max\{d_j^{\gamma-1} + \bar{p}_i + \hat{L}_{ij} - \alpha_{lb}H_{ij} - \epsilon;\ d_j^{\gamma} + \bar{p}_i - \alpha_{lb}H_{ij} - \epsilon\}$;
        $\sigma^+(i) \leftarrow j$; changed $\leftarrow$ True;
      end
    end
  end
  if *not changed* **then**
    return α_{lb};
  end
end

In the following, we adapt the Howard's algorithm to take into account the uncertainty set Ξ^{Γ} presented in Section 3. A pseudo-code of the robust Howard's algorithm is given in Algorithm 4.

The principle of the algorithm is the same as the negative circuits cancelling algorithm. It starts with a lower bound α_{lb} and improves this bound until the optimality.

The algorithm uses a graph $G_{\sigma^+} = (\mathcal{T}, E_{\sigma^+}, A^{\xi}_{\alpha_{lb}})$ called *policy graph*, where $E_{\sigma^+} = \{(i, \sigma^+(i)), i \in \mathcal{T}\}$ and $\sigma^+(i)$ denotes the successor of node i. Each node of this graph has only one out-degree. The algorithm starts by initializing the policy graph by n disjunctive circuits. Then, the algorithm determines the circuit c with a maximum ratio and sets α_{lb} to:

$$\alpha_{lb} = \frac{1}{\sum_{(i,j)\in c} H_{ij}} \left(\sum_{(i,j)\in c} \bar{L}_{ij} + \max_{\xi \in \Xi^{\Gamma}} \left\{ \sum_{(i,j)\in c} \hat{L}_{ij}(\xi) \right\} \right).$$

Afterwards, the policy graph G_{σ^+} is changed such that it contains only one circuit c and paths from each node $i \notin c$ to a chosen node $h \in c$ and computes the longest path d_i^{γ}, from each node $i \in \mathcal{T}$ to h, under the uncertainty set Ξ^{Γ}. The final step is to check if the labels d_u^{γ} can be improved by adding some arc $e \in E$ to the graph G_{σ^+}. These operations are repeated until there is no possible improvement.

Proposition 5 *The robust Howard's algorithm runs in $\mathcal{O}(n^2 m \Gamma^2 |\mathcal{C}|)$ time, where $|\mathcal{C}|$ is the number of simple circuits in the graph G.*

Proof Determining the maximum ratio of the policy graph can be achieved in $\mathcal{O}(n + nlog(n))$. Then, the step of changing the policy graph G_{σ^+} such that it contains only the circuit c and paths from each node $i \notin c$ to h can be performed in $\mathcal{O}(mn\Gamma)$. The computation of the longest path can be done in $\mathcal{O}(mn\Gamma)$ in backward breadth first search by using the following formula:

$$\forall\, i \in \mathcal{T}, \gamma \in [1, \Gamma] :$$

$$d_i^{\gamma} = \max\{d^{\gamma-1}_{\sigma^+(i)} + \bar{L}_{i\sigma^+(i)} + \hat{L}_{i\sigma^+(i)} - \alpha_{lb} H_{i\sigma^+(i)},\ d^{\gamma}_{\sigma^+(i)} + \bar{L}_{i\sigma^+(i)} - \alpha_{lb} H_{i\sigma^+(i)}\ \}. \tag{14}$$

Finally, the last step of the while loop can be achieved in $\mathcal{O}(m\Gamma)$. So, an iteration of the while loop can be achieved in $\mathcal{O}(mn\Gamma)$. Note that, in the robust Howard's algorithm, the cycle time α is improved at most every $n\Gamma$ iterations of the while loop (this can be proved as in Dasdan et al. [27]). Finally, the while loop is repeated until there is no possible improvement of the distance labels d_i^{γ}, hence, the robust Howard's algorithm turns in $\mathcal{O}(n^2 m \Gamma^2 |\mathcal{C}|)$, where $|\mathcal{C}|$ is the number of simple circuits in the initial uniform graph G. □

Example 5 We reconsider the Example 4 and we apply the robust version the Howard's algorithm.

Fig. 6 displays the initial policy graph G_{σ^+}. The maximum circuit ratio of the graph is $\alpha = 5$, the associated circuit is $(1, 1)$ and the scenario giving this value is $\xi = (1, 0, 0, 0)$. The longest paths from node 1 to the other nodes, under the uncertainty set Ξ^{Γ}, are computed and the policy graph is updated. Both of the label values and the policy graph G_{σ^+} are given respectively in Fig. 7 and Fig. 8.

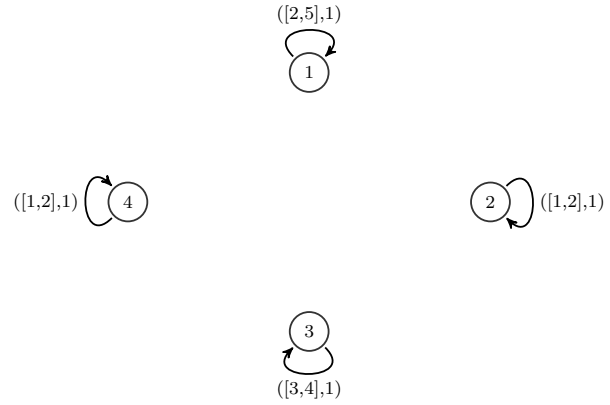

Fig. 6: The initial policy graph G_{σ^+}.

Then, other arcs of the original graph G are checked for possible improvements. The label values and the policy graph G_{σ^+} are given in Fig. 9 and Fig. 10.

The maximum circuit ratio of the graph is $\alpha = 7$, the associated critical circuit is $(1, 2, 4, 1)$ and the scenario giving this value is $\xi = (1, 0, 0, 0)$.

Node	1	2	3	4
$\gamma = 0$	0	-3	0	6
$\gamma = 1$	0	-2	1	7

Fig. 7: The label values d_i^γ of the iteration 1.

Note that, once the optimal cycle time α_{opt} is computed by one of the three above algorithms, the worst case scenario ξ can be obtained. Hence, one can compute the vector of the starting times $(t_i(\xi))_{i\in\mathcal{T}}$, in the worst case scenario, by computing the longest path in the graph $G^{'} = (G, p_e(\xi) - \alpha_{opt} H_e)$.

5 Experimental study

In order to validate the proposed approaches, we present numerical experiments that we performed on randomly generated instances. We first describe how the instances are built, then we discuss the numerical results.

Since there is no benchmark available in the literature, even for the deterministic Basic Cyclic Scheduling Problem, we randomly generate the instances. We consider instances where the number of tasks varies from 10 to 200. In order to build the instances, we proceed as follows. First, we generate a precedence graph using the library *GGen* (see [28]). We vary the probability of having an arc from i to j from 0.5 until 0.9. Once the precedence graph is obtained, we add two dummy

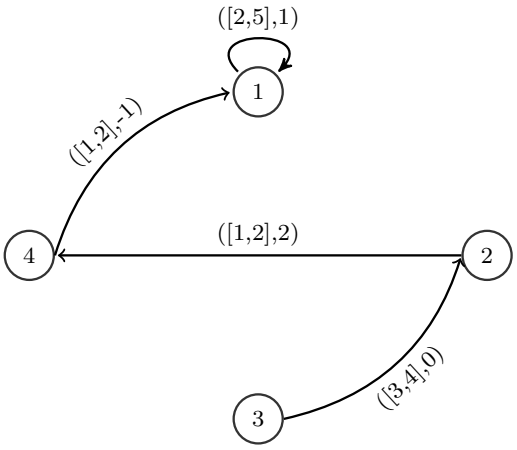

Fig. 8: The policy graph G_p of the iteration 1.

tasks s and e. For each node i of the graph having zero in-degree (rep. zero out-degree), we add an arc (s, i) (resp. (i, e)) and finally, we add a return arc from e to s. Next, we generate the arc valuation uniformly. The nominal processing times take values from 1 to 10 and the deviations from 0% to 30%.

Node	1	2	3	4
$\gamma = 0$	0	-3	0	6
$\gamma = 1$	2	-2	1	7

Fig. 9: The label values d_i^γ of the iteration 2.

The three algorithms have been coded using the C++ language and compiled with GNU G++ 5.4.

We present in Table 1 the average running times of each algorithm, the binary search algorithm (BS), the negative circuits cancelling algorithm (NCC) and the robust Howard's algorithm (R-HOW), according to the number of tasks and different values of the budget of uncertainty.

Table 1 shows that the R-HOW algorithm outperforms the NCC algorithm and the BS algorithm. The running times are comparable on small-size instances, but at least 100 times smaller for the most of the other cases. Note that in this table we report only results on instances having at most 70 tasks. The robust Howard's algorithm solves all the instances (from 10 to 200 tasks), contrary to the binary search algorithm and the negative circuits cancelling algorithm that exceed the time limits with instances having respectively 70 tasks and 80 tasks. We also remark that, for instances with 70 tasks, the average running time of the negative

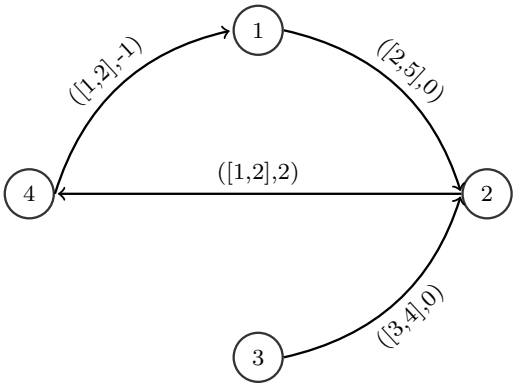

Fig. 10: The policy graph G_{σ^+} of the iteration 2.

cycle cancelling algorithm is smaller than the one of robust Howard's algorithm when $\Gamma = 0$. However, this is not the case when the level of the deviations increases.

Fig. 11 displays some plots, each one representing, for a fixed budget of the uncertainty varying from 0% to 100%, the average running time of the robust Howard's algorithm according to the number of the tasks. The figure shows also that the robust Howard's algorithm is insensitive to the variation of the level of the deviations, since the running times with different budget of the uncertainty are quite stable. This behaviour is not present in the case of the two other algorithms. Table 1 shows that the running times for these two algorithms are very sensitive to the variation of the budget of the uncertainty and increase significantly.

6 Conclusions and perspectives

In this paper, we consider the Basic Cyclic Scheduling Problem where processing times are affected by uncertainties. We model the problem as a robust optimization problem with polyhedral uncertainty set. We derive a separation procedure and propose three algorithms to solve the problem. The two first are based on negative circuits detection and the last one is an adaptation of the Howard's algorithm. In order to validate the three algorithms, we perform numerical experiments on randomly generated instance. The NCC and BS algorithms solves instances having less than 70 tasks and the Howard's algorithm adaptation solve all the instances. Numerical experiments show that the robust Howard's algorithm is the most efficient, in terms of the running times, among the three algorithms.

The next step is considering additional constraints such as disjunctive constraints or deadlines. More precisely, a relevant perspective is to tackle the robust version of the cyclic job shop problem. The main difference with the R-BCSP is that the disjunctions occur since the tasks are mapped into limited resources. Note that when the disjunctions are fixed, the problem can be solved as Robust

# Tasks	Γ(%)	BS Time (s)	NCC Time (s)	R-HOW Time (s)
10	0	0.019	0.004	0.000
	10	0.038	0.006	0.000
	20	0.063	0.010	0.000
	30	0.083	0.013	0.000
	40	0.104	0.017	0.000
	50	0.125	0.021	0.000
	70	0.174	0.030	0.000
	90	0.232	0.039	0.000
	100	0.26	0.047	0.000
20	0	0.18	0.020	0.002
	10	0.544	0.088	0.002
	20	0.914	0.145	0.002
	30	1.305	0.222	0.002
	40	1.715	0.289	0.002
	50	2.157	0.359	0.002
	70	3.094	0.532	0.003
	90	4.144	0.696	0.003
	100	4.634	0.794	0.003
30	0	0.709	0.076	0.018
	10	2.902	0.464	0.018
	20	5.23	0.845	0.019
	30	7.75	1.260	0.019
	40	10.512	1.679	0.020
	50	13.424	2.142	0.021
	70	19.049	3.136	0.022
	90	25.789	4.280	0.024
	100	29.467	4.913	0.024
40	0	2.11	0.227	0.095
	10	10.735	1.726	0.097
	20	20.431	3.244	0.101
	30	30.067	4.825	0.101
	40	41.538	6.496	0.106
	50	52.875	8.301	0.110
	70	75.958	12.210	0.114
	90	104.761	16.324	0.119
	100	120.002	19.057	0.120
50	0	4.646	0.499	0.369
	10	29.588	4.717	0.375
	20	57.57	8.932	0.392
	30	86.651	13.539	0.399
	40	119.545	18.489	0.404
	50	151.41	23.592	0.419
	70	222.526	34.975	0.411
	90	299.486	47.818	0.425
	100	342.912	55.292	0.430
60	0	9.669	1.007	0.942
	10	71.619	10.912	0.959
	20	138.336	21.100	0.983
	30	206.963	31.967	0.983
	40	284.244	44.397	0.994
	50	364.813	55.376	1.016
	70	530.786	83.376	1.071
	90	719.544	112.423	1.085
	100	793.328	132.045	1.101
70	0	17.979	1.871	2.381
	10	149.431	22.453	2.353
	20	290.334	43.947	2.322
	30	444.957	66.400	2.374
	40	596.621	92.408	2.393
	50	747.239	119.380	2.509
	70	938.809	173.000	2.590
	90	-	237.221	2.580
	100	-	290.837	2.547

Table 1: Average solution times in seconds for BS, NCC and R-HOW.

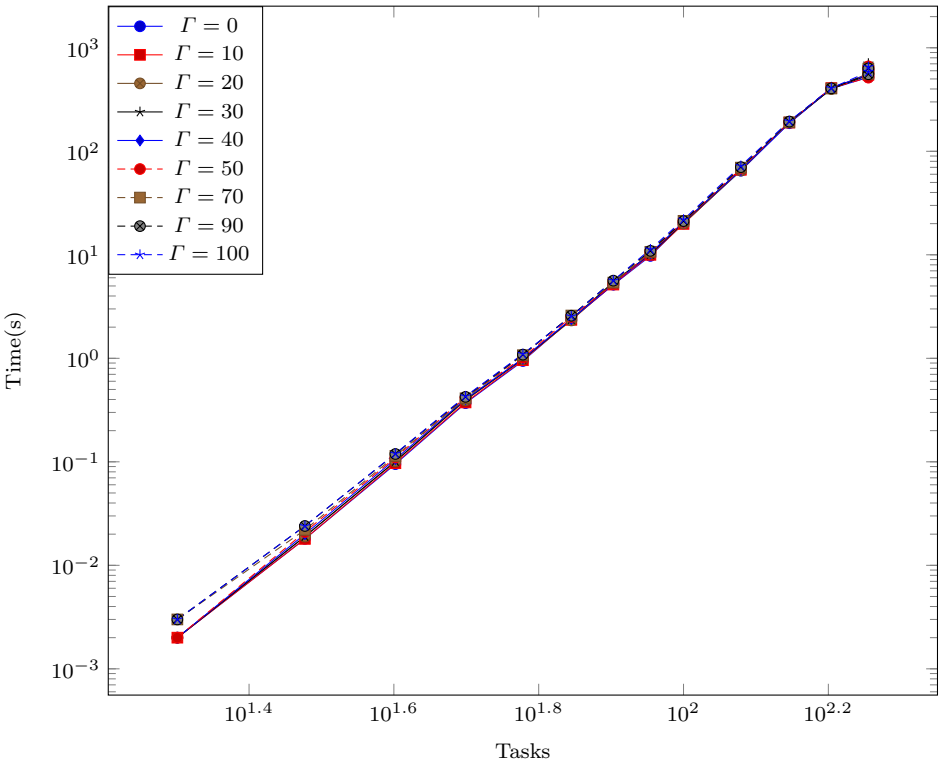

Fig. 11: Average running times for R-HOW with respect to different budget of uncertainty.

Basic Cyclic Scheduling Problem, therefore a Branch and Bound method can be combined with the presented algorithms to solve the problem.

References

1. M.W. Dawande, H.N. Geismar, S.P. Sethi, C. Sriskandarajah, *Throughput optimization in robotic cells*, vol. 101 (Springer Science & Business Media, 2007)
2. V. Kats, E. Levner, Journal of Scheduling **5**(1), 23 (2002)
3. G. Cavory, R. Dupas, G. Goncalves, European Journal of Operational Research **161**(1), 73 (2005)
4. H. Chen, C. Chu, J. Proth, IEEE Trans. Robotics and Automation **14**(1), 144 (1998)
5. P. Sucha, Z. Pohl, Z. Hanzálek, in *Real-Time and Embedded Technology and Applications Symposium, 2004. Proceedings. RTAS 2004. 10th IEEE* (IEEE, 2004), pp. 404–412
6. M. Ayala, A. Benabid, C. Artigues, C. Hanen, Computational Optimization and Applications **54**(3), 645 (2013)
7. R. Govindarajan, E.R. Altman, G.R. Gao, IEEE Transactions on Parallel and distributed systems **7**(11), 1133 (1996)
8. C. Hanen, A. Munier, *Scheduling Theory and Its Applications* (Wiley, 1995), chap. Cyclic Scheduling on Parallel Processors: An Overview
9. P. Brucker, T. Kampmeyer, Discrete Applied Mathematics **156**(13), 2561 (2008)
10. A. Che, J. Feng, H. Chen, C. Chu, European Journal of Operational Research **240**(3), 627 (2015)
11. P. Chretienne, Discrete Applied Mathematics **30**(2-3), 109 (1991)
12. A. Munier, Discrete applied mathematics **64**(3), 219 (1996)

13. A. Ben-Tal, A. Goryashko, E. Guslitzer, A. Nemirovski, Mathematical Programming **99**(2), 351 (2004)
14. A. Soyster, Operations Research pp. 1154–1157 (1973)
15. D. Bertsimas, M. Sim, Operations research **52**(1), 35 (2004)
16. A. Thiele, T. Terry, M. Epelman, Rapport technique pp. 4–37 (2009)
17. M. Minoux, Journal of Global Optimization **49**(3), 521 (2011)
18. A. Dasdan, ACM Transactions on Design Automation of Electronic Systems (TODAES) **9**(4), 385 (2004)
19. C. Hanen, European Journal of Operational Research **72**(1), 82 (1994)
20. M. Gondran, M. Minoux, *Graphs and algorithms* (Wiley, 1984)
21. V. Gabrel, M. Lacroix, C. Murat, N. Remli, Discrete Applied Mathematics **164**, 100 (2014)
22. M. Minoux, in *Encyclopedia of Optimization* (Springer, 2008), pp. 3317–3327
23. R.K. Ahuja, T.L. Magnanti, J.B. Orlin, (1993)
24. B.V. Cherkassky, A.V. Goldberg, Mathematical Programming **85**(2), 277 (1999)
25. R.A. Howard, (1960)
26. J. Cochet-Terrasson, G. Cohen, S. Gaubert, M.M. Gettrick, J.P. Quadrat, in *Proceedings of the IFAC Conference on System Structure and Control* (Nantes, 1998)
27. A. Dasdan, S. Irani, R.K. Gupta, An experimental study of minimum mean cycle algorithms. Tech. rep., Tech. rep. 98-32, Univ. of California, Irvine (1998)
28. D. Cordeiro, G. Mounié, S. Perarnau, D. Trystram, J.M. Vincent, F. Wagner, in *Proceedings of the 3rd international ICST conference on simulation tools and techniques* (ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010), p. 60