De-Cycling Cyclic Scheduling Problems

Alessio Bonfietti, Michele Lombardi, Michela Milano
DISI, University of Bologna
{alessio.bonfietti,michele.lombardi2,michela.milano}@unibo.it

Abstract

An elegant way to tackle a problem that you cannot solve is to cast it to a problem that you can solve very well. *Cyclic* Scheduling problems are very similar to Resource Constrained Project Scheduling Problems (RCPSP), except that the project activities are repeated over time. Due to the similarity, reducing Cyclic Scheduling problems to RCPSPs seems an appealing approach. In this paper we discuss four methods to perform the reduction. The first two are existing techniques. The remaining ones are novel and include the first (to the best of our knowledge) equivalent RCPSP formulation of a cyclic problem. We compare the presented approaches in an experimental evaluation.

Introduction

The classical Resource Constrained Project Scheduling Problem (RCPSP) consists in ordering a set A of activities a_i, connected by a set E of precedence constraints (a_i, a_j). The two sets form the so-called Project Graph. Each activity has a fixed duration d_i and requires some amount of one or more resources with limited capacity. For sake of simplicity we consider here a single resource with capacity c and refer to the requirements as rq_i. The objective is to minimize the makespan. Cyclic scheduling problems (see e.g. [Draper et al., 1999; Hanen, 1994]) are very similar, with the main difference that activities are repeated indefinitely over time.

We refer as $\langle i, \omega \rangle$ to the ω-th execution of activity a_i, where $\omega \in \mathbb{Z}$ is called execution number. A schedule is an assignment of a start time $s(i, \omega)$ to each $\langle i, \omega \rangle$ and has in principle infinite size. From a practical perspective, one is typically interested in finding a *periodic* schedule, where:

$$s(i, \omega) = s(i, 0) + \omega \cdot \lambda \tag{1}$$

where $\lambda \geq 0$ is the *period* and measures the solution quality: the lower the period, the better the schedule.

As a second difference w.r.t. the RCPSP, precedence constraints may link activity executions with different ω. In detail, each arc $(a_i, a_j) \in E$ is labeled with a delay $\delta_{i,j}$ and means that $\langle j, \omega \rangle$ must wait for the end of $\langle i, \omega - \delta_{i,j} \rangle$. In

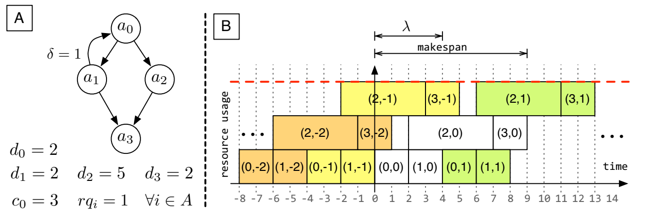

Figure 1: A) A Cyclic Scheduling Problem. B) A Periodic Schedule.

other words, the execution 0 of a_j must wait for at least $\delta_{i,j}$ executions of a_i before it can start. Formally:

$$s(j, \omega) \geq s(i, \omega - \delta_{i,j}) + d_i \tag{2}$$

In the case of a periodic schedule, Equation (2) becomes:

$$s(j, 0) + \omega \cdot \lambda \geq s(i, 0) + d_i + (\omega - \delta_{i,j}) \cdot \lambda \tag{3}$$

$$\text{i.e. } s(j, 0) \geq s(i, 0) + d_i - \delta_{i,j} \cdot \lambda \tag{4}$$

which is not dependent on ω. Hence, if the constraints are satisfied for $\omega = 0$, then they are satisfied in general.

Figure 1A shows a problem instance. Note that arcs with $\delta_{i,j} > 0$ may lead to feasible cycles in the Project Graph. Figure 1B shows the corresponding optimal periodic schedule. We refer as *makespan* to the length of the interval spanned by all the executions having the same ω value. The makespan may be considerably larger than the period (9 time units vs 4 in the figure). As a consequence, different schedule executions may require the resource concurrently.

Cyclic Scheduling problems can be solved via ad hoc approaches, based on Integer Linear Programming (e.g. [Artigues et al., 2012]), Constraint Programming (e.g. [Bonfietti et al., 2012]) or specialized heuristics (e.g. [Blachot, Dupont de Dinechin, and Huard, 2006]). It seems however reasonable to tackle Cyclic Scheduling problems by casting them to a RCPSP, since this allows the application of all the constraint techniques developed for traditional scheduling problems [Baptiste, Le Pape, and Nuijten, 2001]. In the following, we discuss four methods to perform the reduction. The first two are existing techniques based on simplifying assumptions, that can therefore incur a loss of optimality. The third method is novel and consists in the first (to the best of our knowledge) equivalent RCPSP formulation of a cyclic

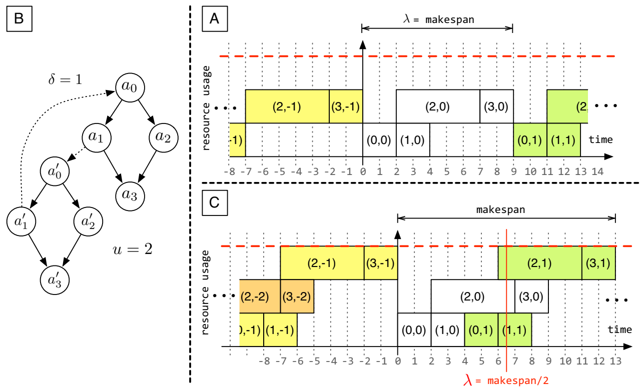

Figure 2: A) An optimal blocked schedule. B) Unfolded graph with $u = 2$. C) An unfolded schedule.

problem. The fourth method leads to a quasi-RCPSP formulation, which can however be modeled by many constraint solvers without ad hoc extensions.

Existing Reduction Approaches

Blocked Scheduling: Blocked Scheduling is the most straightforward reduction approach and consists in forcing the period to be equal to the makespan. Formally:

$$\lambda = \max_{a_i \in A} \{s(i, 0) + d_i\} \tag{5}$$

As a consequence, all precedence constraints with $\delta_{i,j} > 0$ are trivially satisfied, since Equations (4) are in the form $s(i, 0) \geq \theta$, with $\theta \leq 0$. Moreover, executions with different ω do not compete for the resource. Hence, a periodic schedule can be obtained by: 1) removing the arcs with $\delta_{i,j} > 0$. 2) Solving the problem as a classical RCPSP. The solution becomes the schedule for execution 0. An optimal blocked schedule for our example is shown in Figure 2A. The approach is easy to implement, but may incur a substantial loss of solution quality (e.g. in the figure λ changes from 4 to 9).

Unfolded Scheduling: The Unfolded Scheduling technique [Parhi and Messerschmitt, 1991] improves over Blocked Scheduling by adding a preprocessing step. This consists in obtaining a new graph, representing a number u (a.k.a. *unfolding factor*) of consecutive executions. Figure 2B shows the unfolded graph with $u = 2$ for the example problem, which contains two replicas of each activity.

Then, a solution is obtained via Blocked Scheduling. Figure 2C shows an optimal schedule for the unfolded example graph. The schedule follows a periodic pattern every u repetitions. The corresponding macro-period in this case is 13, corresponding to an 'average λ' of 6.5. Unfolded Scheduling achieves better period values by allowing different executions to overlap, at the price of solving a larger RCPSP. Determining the optimal unfolding factor is not trivial, since increasing u does not necessarily decrease the period.

Novel Formulation

In the following we outline a novel technique to solve a Cyclic Scheduling problem as a RCPSP that incurs no loss of optimality. We focus on the feasibility version of the problem, i.e. finding a feasible periodic schedule with period equal to a fixed value λ_0. Optimization can be done by performing (e.g.) binary search on the possible λ_0 values.

Dealing with Precedence Constraints: Since the period is fixed, the precedence constraints from Equation (4) become:

$$s(j, 0) \geq s(i, 0) + d_i - \delta_{i,j} \cdot \lambda_0 \tag{6}$$

If $d_i - \delta_{i,j} \cdot \lambda_0 \geq 0$, then we have a start-to-start precedence constraint with a minimal time lag (an end-to-start constraint if $\delta_{i,j} = 0$). If instead $d_i - \delta_{i,j} \cdot \lambda_0 < 0$, then we get:

$$s(i, 0) - s(j, 0) \leq |d_i - \delta_{i,j} \cdot \lambda_0| \tag{7}$$

that is, a start-to-start precedence constraint between a_j and a_i with a maximal time lag. Therefore we can model the temporal constraints by building a RCPSP with an activity for each execution $\langle i, 0\rangle$ and by adding arcs as from Equation (6). Figure 3A shows the RCPSP for the problem from Figure 1A, with $\lambda_0 = 4$. Note that the arc (a_1, a_0) with $\delta_{1,0} = 1$ has become a start-to-start arc between a_0 and a_1, with maximal time lag equal to $|d_1 - \delta_{1,0} \cdot \lambda_0| = 3$.

Dealing with Resource Constraints: We take into account cyclic resource restrictions by modeling multiple executions as additional RCPSP activities. We also constrain the activities representing consecutive executions so that:

$$s(i, \omega + 1) = s(i, \omega) + \lambda_0 \tag{8}$$

Hence we retain the periodicity, while in Unfolded Scheduling the solver is free to assign non-periodic start times. As a consequence, the precedence constraints for execution 0 hold for every ω and there is no need to compute an unfolded graph. Now, we call a *modular interval* a time interval in the form $[t, t + \lambda_0[$ with $t \in \mathbb{R}$. For a periodic schedule:

Statement 1. *If the resource constraints are satisfied on a modular interval I, they are satisfied on the whole schedule.*

Statement 2. *Any modular interval I contains the start time of exactly one execution of each activity, i.e.:*

$$\forall a_i \in A\ ,\ \exists! \omega \in \mathbb{Z}\ : s(i, \omega) \in I \tag{9}$$

Both the statements follow directly from Equation (1). Hence, our goal is to use additional activities and constraints to provide a correct model of at least one modular interval.

Now, let us assume an upper bound M on the makespan of a periodic schedule is available. Due to the periodicity constraints, no more than $\rho = \lceil M/_{\lambda_0} \rceil$ executions of the periodic schedule can overlap in a single modular interval. Hence, it is enough to build ρ replicas of each activity a_i, corresponding to executions $\langle i, 0\rangle \ldots \langle i, \rho - 1\rangle$. For the example graph, assuming $M = 11$, we have to build 3 replicas of each activity (see Figure 3B). The start times of the replicas representing consecutive execution obey the periodicity constraints, depicted as dashed double arrows.

We now need to make sure that there exists a modular interval which is guaranteed to contain the start time of exactly one of the replicas of each a_i (see Statement 2). Due to the periodicity constraints (8), this is equivalent to proving that

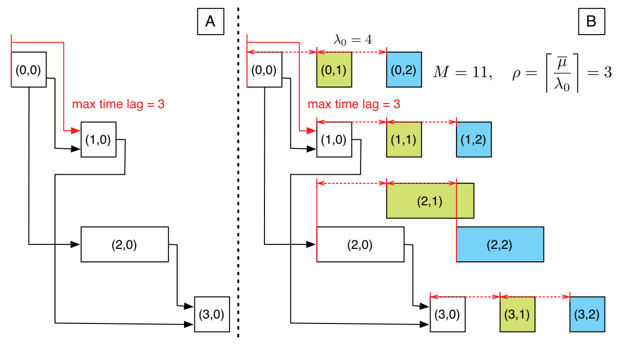

Figure 3: A) The equivalent RCPSP for the precedence constraints. B) The equivalent RCPSP for the full problem.

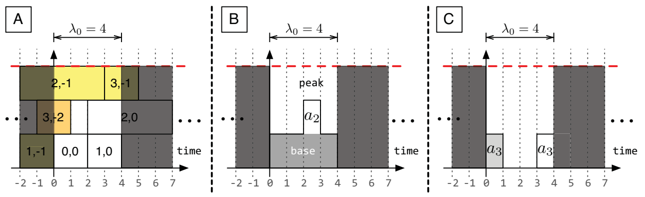

Figure 4: A) Part of a periodic schedule. B) Base and peak functions. C) Peak function crossing the interval boundaries.

at least one replica starts in the interval. Now, since M is a makespan bound, with no loss of generality we can post:

$$s(i,0) \geq 0 \text{ and } s(i,0) + d_i \leq M, \ \forall a_i \in A \qquad (10)$$

Now, consider the modular interval $I^* = [M - \lambda_0, M[$. Let us assume by contradiction that no replica of a_i starts in I^*. This means that either $s(i, \rho - 1) < M - \lambda_0$ or $s(i, 0) \geq M$. However, assuming $d_i > 0$:

$$s(i, \rho - 1) < M - \lambda_0 \Rightarrow s(i,0) < 0 \qquad (11)$$

$$s(i,0) \geq M \Rightarrow s(i,0) + d_i > M \qquad (12)$$

Since $s(i,\omega) = s(i,0) + \omega \cdot \lambda_0$ and $\rho = \lceil {}^{M}/_{\lambda_0} \rceil$. If $d_i = 0$ there is no resource usage at all. Therefore, Constraints (10) guarantee that the modular interval I^* contains the start time of at least one replica of each activity.

Equivalent RCPSP Formulation: In summary, our equivalent RCPSP formulation consists in the following Constraint Satisfaction Problem (CSP):

$$s(i, \omega + 1) = s(i, \omega) \qquad \forall a_i \in A, \omega \in 0..\rho - 2 \quad (13)$$

$$s(j, 0) \geq s(i, 0) + d_i - \delta_{i,j} \cdot \lambda_0 \qquad \forall (a_i, a_j) \in E \quad (14)$$

$$\text{cumulative}([s(i,\omega)], [d_i], [rq_i], c) \quad (15)$$

$$s(i,\omega) \in [0..M - d_i[\qquad \forall a_i \in A, \omega \in 0..\rho - 1 \quad (16)$$

where in Equation (15) we use the classical cumulative constraint [Baptiste, Le Pape, and Nuijten, 2001] to model resource capacity restrictions. With $[s(i,\omega)]$ we refer to the vector of all execution start variables. We assume the $[d_i]$ and $[rq_{i,k}]$ vectors are adjusted to match the size of $[s(i,\omega)]$.

Computing a Makespan Bound: By makespan bound we mean an upper bound on the makespan of a feasible periodic schedule. Here we propose two partial bounds (M^0 and M^1), which respectively guarantee the satisfaction of precedence and resource constraints. A valid makespan bound is given by $\max(M^0, M^1)$. The bounds require to have $\delta_{i,j} \geq 0$ for all arcs, which is true in all known Cyclic Scheduling applications. We also recall that the original cyclic problem has no time window and no time lag. Then:

Theorem 1. *Assuming no resource restriction is enforced, than a feasible schedule exists iff there is a feasible schedule with makespan* $\leq M^0 = \sum_{a_i \in A} d_i$.

Proof. We need to show that there exists a schedule with makespan $\leq M^0$ that satisfies all the precedence constraints. If $\delta_{i,j} = 0$ for each arc, then Constraints (6) are classical end-to-start precedence relations and M^0 is a well-known valid bound. Any $\delta_{i,j} > 0$ leads to a reduction of the temporal distance between a_i and a_j, possibly making it negative. This may render the problem infeasible, but it cannot increase the length of any feasible schedule. □

We can obtain a second bound, by reasoning on the resource constraints alone. The bound builds on the following theorem (from [Lombardi et al., 2011]):

Theorem 2. *The resource usage due to the overlapping executions of* a_i *in a modular interval* I *is the sum of a* base *and a* peak *function. The* base *function has height* $rq_i \cdot \lfloor {}^{d_i}/_{\lambda_0} \rfloor$ *and spans the whole interval. The* peak *function has height* rq_i*, length* $d_i \mod \lambda_0$*, starts (within* I*) at* $s(0,0) \mod \lambda_0$*, and can "cross" the interval boundary.*

Figures 4B and C show the cumulative usage of activities a_2 and a_3 in the modular interval from Figure 4A. Note that the peak function of a_3 crosses the interval boundary.

Theorem 3. *Assuming no precedence restriction is enforced, then a feasible schedule exists iff there is a feasible schedule with makespan* $\leq M^1 = \max_{a_i \in A} (\lambda_0 + d_i)$.

Proof. As a consequence of Theorem 2, the existence of a resource-feasible schedule depends only on the start time of the peak functions. If $s(i,0) + d_i < \lambda_0 + d_i$, then the start times of the peak functions, i.e. $s(i,0) \mod \lambda_0$, are free to take any value in the modular interval $[0, \lambda_0[$. □

Performing Period Optimization: We can find the optimal period for a Cyclic Scheduling problem by performing binary or linear search on λ. A valid lower bound on λ (based on the resource constraints) is ${}^{\sum_{a_i} rq_i \cdot d_i}/_{c}$. Possibly better bounds can be computed via more advanced techniques (see [Artigues et al., 2012]) . A valid upper bound for the period is M^0, which corresponds to the blocked scheduling case.

Quasi-RCPSP Formulation: Here we describe another reformulation for the Cyclic Scheduling problem. This is technically not a RCPSP, but the resulting model can be handled without custom extensions by several constraint solvers (such as IBM ILOG CP Optimizer [IBM Corp., 2012]).

The main idea is to to build a schedule for the peak functions appearing in the the modular interval $[0, \lambda_0[$. From Theorem 2 we know that the base function of each activity permanently consumes a portion of the resource capacity. This can be modeled by reducing the capacity c to:

$$\bar{c} = c - \sum_{a_i \in A} rq_i \cdot \lfloor {}^{d_i}/_{\lambda_0} \rfloor \qquad (17)$$

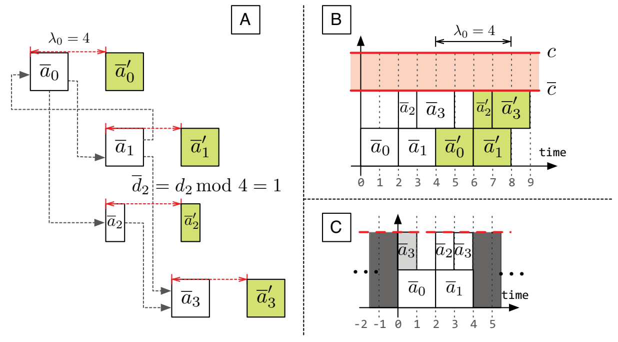

Figure 5: A) Quasi-RCPSP formulation. B,C) A solution for the Quasi-RCPSP formulation

The remaining resource usage is due to the peak functions, which we model via RCSP activities $\overline{a}_i$ (referred to as *reduced activities*). Each $\overline{a}_i$ has duration $\overline{d}_i = d_i \mod \lambda_0$ and resource requirement equal to rq_i. For each $\overline{a}_i$ we introduce a start time variable $\overline{s}(i)$, ranging in $[0..\lambda_0[$. Then, we need to model the precedence and the resource constraints.

The problem with the precedence constraints is that they are defined for the $s(i,0)$ variables rather than for the $\overline{s}(i)$ ones. However, from Theorem 2 we know that $\overline{s}(i) = s(i,0) \mod \lambda_0$ or, equivalently:

$$s(i,0) = \overline{s}(i) + \overline{b}(i) \cdot \lambda_0 \tag{18}$$

We use integer decision variables to represent the $\overline{b}(i)$ values. Their domain can be limited to $\{0..\rho - 1\}$, since we know that $s(i,0) \leq M$. By combining Equation (18) and (6), we can reformulate every arc (a_i, a_j) as:

$$\overline{s}(j) + \overline{b}(j) \cdot \lambda_0 \geq \overline{s}(i) + \overline{b}(i) \cdot \lambda_0 + d_i - \delta_{i,j} \cdot \lambda_0 \tag{19}$$

which is an end-to-start precedence constraint with a decision-dependent time lag, equal to $(\overline{b}(i) - \delta_{i,j} - \overline{b}(j)) \cdot \lambda_0$. This is fairly easy to model with modern constraint systems.

The resource constraints can be modeled exactly as in the previous formulation. As a considerable advantage, since $\overline{s}(i) < \lambda_0$ and $\overline{d} < \lambda_0$, the makespan for the schedule of peak functions is always strictly lower than $2 \cdot \lambda_0$. As a consequence, it is sufficient to introduce a single replica $\overline{a}'_i$ of each reduced activity $\overline{a}_i$. The start time of the replica will be $\overline{s}'(i) = \overline{s}(i) + \lambda_0$. Figure 5A shows the graph for the reformulation of the example problem. Figure 5B and C respectively show an optimal solution and the corresponding schedule of the peak functions.

Experiments

We finally compare the performance of the approaches discussed in the paper on 200 synthetically generated instances[1] with 5 to 25 activities and a resource with large capacity (the most interesting case for cyclic scheduling).

We solved the unfolded formulation with IBM ILOG CP 1.5 and the schedule-or-postpone search strategy, with unfolding factor u up to 5 ($u = 1$ is Blocked Scheduling). For the two novel formulations (referred to as RCPSP and Q-RCPSP) we employed the CP Optimizer from IBM ILOG CPLEX Studio 12.4, with default search. In all cases, optimization is performed via binary search, with a time limit of 2 seconds on each step: this makes the whole process heuristic, which is common for practical usage settings of Cyclic Scheduling. For each approach we report in Table 1 the average solution time and the average gap over Blocked Scheduling, computed as $(\lambda^*_{approach} - \lambda^*_{blocked}) / \lambda^*_{blocked}$. The results are grouped by the number N of activities.

	u=1	u=3		u=5		RCPSP		Q-RCPSP	
N	TIME	TIME	GAP	TIME	GAP	TIME	GAP	TIME	GAP
5	0.01	0.01	-55.18%	0.41	-68.99%	3.55	-72.43%	2.17	-72.57%
10	0.01	0.59	-37.55%	2.03	-49.43%	3.27	-57.88%	4.65	-58.33%
15	0.01	1.71	-33.75%	4.51	-39.58%	4.04	-44.35%	5.16	-45.01%
20	0.02	3.1	-36.76%	7.4	-41.31%	5.08	-43.17%	5.55	-44.70%
25	0.026	4.65	-31.58%	8.03	-33.84%	6.01	-34.51%	5.17	-36.18%
30	0.067	9.72	-22.39%	11.71	-22.1%	7.12	-21.96%	6.09	-24.12%
35	0.089	11.14	-18.14%	12.94	-10.02%	7.42	-15.33%	5.55	-17.47%
40	0.14	11.73	-13.28%	14.00	-13.02%	6.61	-13.14%	5.29	-15.15%
45	2.65	11.74	-12.54%	14.65	-12.11%	8.71	-12.19%	5.83	-13.99%
50	3.98	13.08	-8.46%	15.11	-8.06%	8.31	-8.56%	6.53	-10.49%

Table 1: Results of the comparison

Blocked Scheduling is the fastest approach, but yields the worst quality solutions. Increasing the unfolding factor leads to lower period values, at the cost of a higher solution time (although mitigated by the 2 sec. time limit). The novel formulations (in particular the Q-RCPSP one) are faster than Unfolded Scheduling and find higher quality solutions. This is a remarkable result, since both the approaches are easier to implement than Unfolded Scheduling. On larger graphs, the 2 sec. time limit becomes a bit too tight for the largest unfolding factors and the RCPSP formulation, causing a small decrease of performance. The Q-RCPSP method in unaffected, since it always contains only two replicas of each activity. Finally, our largest graphs have a higher number of possibly overlapping activities, which makes it easier for Blocked Scheduling to exploit the resource and explains the strong gap decrease from $N = 5$ to $N = 50$.

Conclusions

We have introduced the first (to be best of our knowledge) equivalent RCPSP reformulation and a more efficient, RCPSP based, CSP model. The novel formulations considerably improve over the existing ones in our experimentation and they are easy to implement. Possible developments include devising tighter makespan bounds. By combining the novel formulations with an unfolding step it is possible to obtain so-called K-periodic schedules [Hanen, 1994; Kampmeyer, 2006], which are known to dominate periodic schedules in the presence of limited capacity resources. We also plan to extend the comparison to include ad hoc approaches, such as those mentioned in the introduction.

Acknowledgements We thank Pierre Schaus for suggesting this line of research with a question at CPAIOR 2011.
This work has been partially funded by a Google Focused Grant.

[1]The instances are available for download at http://ai.unibo.it/data/de-cycling.

References

Artigues, C.; Ayala, M. A.; Benabid, A.; and Hanen, C. 2012. Lower and upper bounds for the resource-constrained modulo scheduling problem. In *13th International Conference on Project Management and Scheduling*, 82–85.

Baptiste, P.; Le Pape, C.; and Nuijten, W. 2001. *Constraint-Based Scheduling*. Springer.

Blachot, F.; Dupont de Dinechin, B.; and Huard, G. 2006. SCAN: A Heuristic for Near-Optimal Software Pipelining. *In Euro-Par 2006 Parallel Processing, Lecture Notes in Computer Science* 4128:289–298.

Bonfietti, A.; Lombardi, M.; Benini, L.; and Milano, M. 2012. Global cyclic cumulative constraint. In *CPAIOR*, 81–96.

Brucker, P.; Drexl, A.; Möhring, R. H.; Neumann, K.; and Pesch, E. 1999. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research* 112(1):3–41.

Draper, D. L.; Jonsson, A. K.; Clements, D. P.; and Joslin, D. E. 1999. Cyclic scheduling. In *Proc. of IJCAI*, 1016–1021. Morgan Kaufmann Publishers Inc.

Hanen, C. 1994. Study of a NP-hard cyclic scheduling problem: The recurrent job-shop. *European Journal of Operational Research* 72(1):82–101.

IBM Corp. 2012. IBM ILOG CP Optimizer Web Page.

Kampmeyer, T. 2006. *Cyclic Scheduling Problems*. Ph.D. Dissertation, Universität Osnabrück.

Lombardi, M.; Bonfietti, A.; Milano, M.; and Benini, L. 2011. Precedence constraint posting for cyclic scheduling problems. In *CPAIOR*, 137–153.

Parhi, K. K., and Messerschmitt, D. G. 1991. Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding. *IEEE Transactions on Computers* 40(2):178–195.