Constraints (2020) 25:319–337
https://doi.org/10.1007/s10601-020-09316-z

A new constraint programming model and solving for the cyclic hoist scheduling problem

Mark Wallace[1] · **Neil Yorke-Smith**[2]

Accepted: 15 October 2020 / Published online: 9 November 2020

Abstract
The cyclic hoist scheduling problem (CHSP) is a well-studied optimisation problem due to its importance in industry. Despite the wide range of solving techniques applied to the CHSP and its variants, the models have remained complicated and inflexible, or have failed to scale up with larger problem instances. This article re-examines modelling of the CHSP and proposes a new simple, flexible constraint programming formulation. We compare current state-of-the-art solving technologies on this formulation, and show that modelling in a high-level constraint language, MiniZinc, leads to both a simple, generic model and to computational results that outperform the state of the art. We further demonstrate that combining integer programming and lazy clause generation, using the multiple cores of modern processors, has potential to improve over either solving approach alone.

Keywords Constraint programming · Hoist scheduling · Modelling · MiniZinc

1 Introduction

Hoist scheduling is an abstraction of a real-life scheduling problem. In the hoist scheduling problem we must specify the operation of an industrial hoist which moves along a linear track above a set of tanks. The problem originally arose in automated electroplating lines, but applies to any process where robots transfer items between the process tasks.

Mathematically, hoist scheduling is interesting because at its heart is a complex disjunctive constraint involving three temporal variables, as well as resource variables. Variants of the problem have features such as time windows for how long an object may dwell in each tank, optional tanks, multi-capacity tanks, classes of objects, multiple hoists, multiple tracks, slippage in movement, etc. [3, 7]. We use the term *hoist* rather than robot, and

✉ Neil Yorke-Smith
n.yorke-smith@tudelft.nl

Mark Wallace
mark.wallace@monash.edu

[1] Monash University, Melbourne, Australia

[2] Delft University of Technology, Delft, Netherlands

treatment rather than task, to remain consistent with the literature. We refer to an object that undergoes a series of treatments as a *job*.

The *cyclic hoist scheduling problem* (CHSP) assumes a fixed sequence of jobs to be processed. The challenge is to allow the processing of successive jobs to overlap, so that (different) treatments on different jobs may be carried out at the same time. The schedule of treatments for one job is repeated for subsequent jobs, giving the problem its cyclic nature. The length of a cycle is the time between the start of processing for a job and that of its successor. (Observe that in the absence of overlap the cycle time is the whole process duration for one job.) The usual aim is to minimise this cycle time by maximising the overlap – and for a large overlap there may be many treatments in process simultaneously.

When many treatments are simultaneously in process, the hoists have to be available to complete all the moves between treatments. Moreover, each hoist must itself travel from the end of the last move that it performed to the start of the next move. Thus hoist availability is a complex resource constraint. The challenge is to find a feasible schedule that minimises the cycle time, which is termed its *period*. The central disjunctive constraint in the CHSP connects the period with the temporal decisions about hoists [18].

Given its practical and mathematical importance, an impressive role call of solving techniques have been brought to bear on the CHSP. As surveyed by [3, 8], the techniques include (mixed) integer programming (IP), constraint programming (CP), evolutionary algorithms, bespoke branch-and-bound and heuristics, and many meta-heuristics. With a few exceptions, the literature develops complex, custom models for the particular CHSP variant under study. Despite the wide range of solving techniques applied to the CHSP and its variants, the models have remained complicated and inflexible and have failed to scale up with larger problem instances. To our knowledge no methods have been applied to problems with more than 20 tanks.

This article presents two contributions. First, we re-examine modelling of the CHSP and propose a new constraint programming formulation. This new model is significantly simpler than those in the literature while being able to accommodate several variants of CHSP with minimal modification.

Our second contribution is to solve our model with different optimisation approaches using single and multiple cores: integer programming, lazy clause generation, and an example hybrid which combines both approaches. To support this flexibility our model is implemented in a generic modelling language, MiniZinc [14]. We compare its performance on standard and new benchmark problems against results reported in the literature. We show that, with our model, current state-of-the-art IP and LCG solvers can be used off the shelf to obtain computational results that outperform models from the literature.

The next section introduces the CHSP. Section 3 explains our basic model and Section 4 discusses how it differs from previous models. Section 5 extends the model to more complex CHSP instances. Section 6 performs an empirical study. Section 7 discusses related work and Section 8 concludes the article.

2 Problem formulation

The basic form of the cyclic hoist scheduling problem is as follows. A single hoist operates on a single track above a sequential line of N tanks. A number of identical jobs are placed at the initial stage (0) of the line. Each job is to be processed through the tanks according to a fixed scheme of treatments, and finally be placed at the far end of the line (stage $N + 1$).

Hoist scheduling is distinguished from classical scheduling problems, such as flowshop or jobshop, in that a job under process is either in a tank or held by a hoist. It cannot be left idle until a resource is available. Second, travel times of an operation are not negligible. In addition, to reflect the industrial situation, it is typical for the prescribed processing times in the tanks to be bounded in a time window.

Let the number of treatments be T, and the sequence of treatments be $i, i = 1, \ldots, T$. For the ith treatment, a job must be placed into tank $S(i)$ and removed within the time window of the treatment. Let the minimum and maximum times in the tank $S(i)$ for treatment i be $m(i)$ and $M(i)$ respectively.

The time the hoist needs to move a job from tank $S(i)$ to tank $S(i+1)$ is denoted $F(i)$[1] while the time the hoist needs to move unloaded (empty) from tank n to m is denoted $E(n, m)$.[2]

Note that an earlier job can be still under process in a later treatment, when a new job is picked up by the hoist from stage 0 for its first treatment. However the number of jobs being treated in the same tank at the same time is limited by the tank capacity. Let the capacity of the tanks be $C(n), n = 1, \ldots, N$. In the unit-capacity case, $C(n) = 1, \forall n$.

It suffices to have as decision variables the removal time upon completion of each treatment, which we denote $R(i), i = 0, \ldots, T$, a variable $B(i), i = 1, \ldots, T$ for the number of jobs in each treatment i at the end of the cycle, and the cycle period P.

This definition exploits an important fact about a cycle. During a single cycle exactly one new job is introduced, and one job is completed. Therefore in each cycle each treatment i is completed for just one job. If no job is removed from treatment i, then the tank $S(i)$ would collect all the jobs, and if more than one job was removed then eventually the tank would starve – because only one job arrives in each cycle.

We observe however that almost all previous models have used many additional variables: often a variable for the completion of each treatment for each job under process during the cycle, as well as multiple binary variables. For this reason most models require the maximum number of jobs under treatment in a single cycle, J, to be given as an input. Our model does not require such a maximum.

The CHSP objective is to maximise throughput, i.e., to minimise P while satisfying constraints about hoist movement, tank capacity, treatment sequence and processing time windows.

For simplicity, we will assume sequential treatment. This means $S(i) = i, i = 0, \ldots, N$. Thus the sum

$$F(i) + E(S(i+1), S(j))$$

(which is the time required for a hoist to move a job from $S(i)$ to $S(i+1)$ plus the time to move empty from $S(i+1)$ to the tank $S(j)$ for treatment j) can be shortened to

$$F(i) + E(i, j).$$

Figure 1 gives a numerical example. A straightforward schedule has a cycle of period 50 minutes, whereas the optimal schedule has a cycle of period 20 minutes.

[1] $F(0)$ is the time to move a new job from stage 0 to tank $S(1)$, and $F(N)$ is the time to move a completed job from tank $S(N)$ to stage $N+1$

[2] $E(n, 0)$ is the time for the hoist to travel from tank n to stage 0, and $E(N+1, m)$ is the time for the hoist to travel from stage $N+1$ to tank m

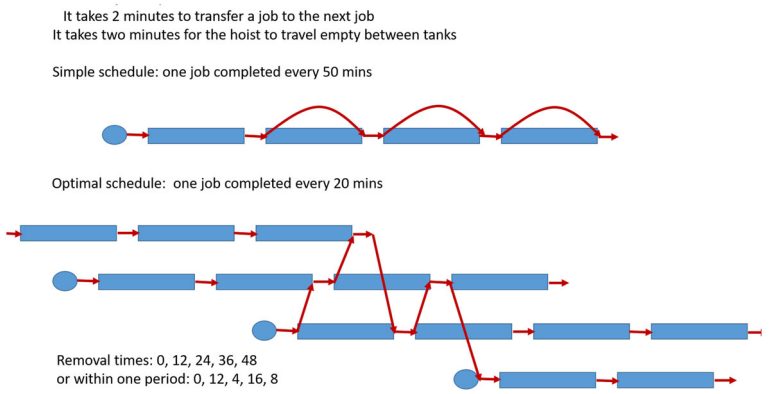

Fig. 1 Example of a simple cyclic hoist scheduling problem

3 A compact model for hoist scheduling

We first present a simple model for the problem with unit capacities and one hoist. We call this the *CUC1H* model. In the following sections we generalise the model and compare it to existing models.

Decision variables

$R(i)\ \ i = 0, \ldots, N$	removal times
$B(i)\ \ i = 1, \ldots, N$	$B(i) = 1$ iff there is a job in tank i at the end of the cycle
P	cycle length or *period*

Initial bounds

$$\begin{aligned} 0 \le R(i) \le P_{ub} \quad & i = 0, \ldots, N \\ B(i) \in \{0, 1\} \quad & i = 0, \ldots, N \\ P_{lb} \le P \le P_{ub} & \end{aligned}$$

Objective

$$\min P$$

Required constraints

$$R(i) \le P \qquad \forall i \in [0, N] \tag{1}$$

$$\begin{aligned} R(i-1) + F(i-1) + m(i) &\le R(i) + P \cdot B(i) \\ &\le R(i-1) + F(i-1) + M(i) \\ &\forall i \in [1, N] \end{aligned} \tag{2}$$

$$\begin{aligned} R(i) + F(i) + E(i+1, j) \le R(j) \ \vee\ R(j) + F(j) + E(j+1, i) \le R(i) \\ \forall i \in [0, N-1],\ j \in [i+1, N] \end{aligned} \tag{3}$$

$$R(i) + F(i) + E(i+1, j) \leq R(j) + P \;\forall i \in [1, N],\, j \in [0, N],\, i \neq j \tag{4}$$

$$R(i) \leq R(i-1) + F(i-1) + P \cdot (1 - B(i)) \qquad \forall i \in [1, N] \tag{5}$$

$$P \geq R(N) + F(N) \tag{6}$$

Constraint for fixed number of jobs, *J*

$$\sum_i B \leq J \tag{7}$$

Symmetry breaking constraint

$$R(0) = 0 \tag{8}$$

Bounds computation We compute static initial lower (P_{lb}) and upper (P_{ub}) bounds on the period variable P thus:

$$P_{lb} = F(N) + \sum_{k=0}^{N-1} F(k) + \min(\{m(k+1)\} \cup \{E(k+1, j) \mid j \neq k \in 0..N\})$$

$$P_{ub} = F(N) + \sum_{k=0}^{N-1} F(k) + m(k+1)$$

The upper bound covers the worst case where the hoist simply follows one job from start to end. The lower bound sums all the necessary full hoist movements and the subsequent minimum empty movements (except if the empty move is to wait at the next tank, necessarily for the minimum tank processing time).

In this model the time of every movement is relative to the start of the current period. The constraint (1) requires all removals to occur within the cycle period.

The constraint (2) enforces the minimum and maximum soak times in each tank. The expression $R(i-1) + F(i-1)$ represents the time the job is placed in tank i. If the job is removed from tank i at a time $R(i)$ later in the cycle than it was put in, then $R(i)$ lies between $R(i-1) + F(i-1) + m(i)$ and $R(i-1) + F(i-1) + M(i)$. In this case there will be no job in the tank at the end of the cycle, so $B(i) = 0$. However if the job is in the tank i at the end of the cycle, then $B(i) = 1$, so $R(i) \leq R(i-1) + F(i-1)$ (by constraint 5) which means that after a job is entered in tank i (at time $R(i-1) + F(i-1)$) it stays in the tank until removed (at time $R(i)$) in the *next* cycle.

Another important fact about hoist movements underlies constraint (3). The time required to go directly from tank i to tank j is always less than the time required to go from tank i to some other tank and from there on to tank j.[3] This the usual triangular inequality saying a straight line is the shortest distance between two points.

Figure 2 shows a feasible cycle for a hoist to remove jobs from tank 0 at time $t0$, tank 2 at time $t1$ and tank 1 at time $t2$. The empty movements ma, mb and mc need enough time for the hoist to be ready for the next removal.

The constraint (3) ensures that the hoist is never scheduled to do two things at once. Consider any two removals $R(i)$ and $R(j)$. Assuming $R(j)$ occurs after $R(i)$, then if these are

[3] In fact this constraint holds under the weaker assumption that, for any three tanks i, j, k, $E(i, j) \leq E(i, k) + F(k) + E(k+1, j)$.

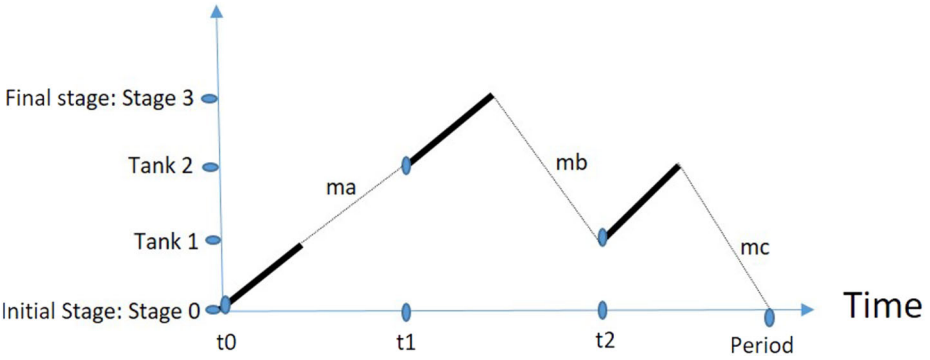

Fig. 2 One hoist and two tanks

successive removals then $R(j) \geq R(i)+F(i)+E(i+1, j)$. If the hoist does other removals (from tanks k_1, k_2, etc.) in between, then the triangular inequality and the constraints on $R(i)$ and $R(k_1)$, on $R(k_1)$ and $R(k_2)$ etc. will ensure $R(j) > R(i)+F(i)+E(i+1, j)$. The other alternative is that $R(j)$ occurs before $R(i)$ and the same argument applies. Constraint (4) extends constraint (3) to preclude an overlap in the case where $R(i)+F(i)+E(i+1, j) \geq P$. In this case potential clash would be between $R(i)$ and $R(j) + P$.

The next constraint (5) is the one that enforces $B(i) = 0$ in case a job is removed from tank i after one has been loaded into the same tank earlier in the cycle, i.e., $R(i) \geq R(i - 1) + F(i - 1)$. In this case, constraint (2) enforces that $R(i - 1) + F(i - 1) + m(i) \leq R(i)$. On the other hand, if the removal occurs earlier in the cycle than the loading of tank i, i.e., $R(i) \leq R(i - 1) + F(i - 1)$, then constraint (2) entails that $B(i) = 1$.

The equality can only happen if the job is removed from the tank at the same time it is entered. Informally there are three ways this could happen: (1) If the same hoist dips the job in the tank for zero time which makes no sense (2) The same hoist removes a job which started one (or more) period(s) before at the same time as dropping the original job in the tank (3) Two hoists handle two jobs exactly one period apart (2) and (3) involve two jobs being in the tank (for zero seconds) while the hoist drops the first and picks up the second.

The constraint (6) ensures that the end of the cycle (and the entry of the next job in tank 1) occurs after the previous job has been removed from the last tank, N. The constraint on the number of jobs concurrently in process, in case it is specified in the input, is enforced by constraint (7). (For models that do not need to constrain the number of concurrent jobs, this can be omitted.)

The final constraint is redundant, but since we can always set the time to zero when a new job enters the system, constraint (8) does so, and removes all the symmetrical cycles which begin at other times.

In short this model *CUC1H* comprises just six essential constraints.

4 Previous models

There are over 20 research papers on the basic unit capacity single hoist problem, and it is surprising that none have devised as simple a model as presented here.

4.1 IP models

The most recently-published IP model for the basic hoist scheduling problem is by [5]. This builds on the previously-published models repairing shortcomings in earlier work with IP, and eliminating redundant constraints.

The model requires eight constraints to impose the soak times, expressed in our model by constraint (2). The first two of these are:

$$R(i) - (R(i-1) + F(i-1)) \leq M(i) + BigM \cdot (B(i) + W(i-1)) \quad \forall i \in [1, N]$$
$$R(i) - (R(i-1) + F(i-1)) \geq m(i) - BigM \cdot (B(i) + W(i-1)) \quad \forall i \in [1, N]$$

where $BigM$ is some sufficiently large number. The binary variable $W(i)$ indicates whether move i goes across the cycle. The remaining six constraints enforcing soak times capture different combinations of minimum and maximum, and combinations of the job remaining in the tank over a cycle or not, and the move $i-1$ crossing the cycle or not.

Dealing with this additional array W of variables requires additional constraints on the period P:

$$R(i) \leq P \qquad \forall i \in [1, N]$$
$$R(i) + F(i) \leq P + BigM \cdot W(i) \qquad \forall i \in [1, N]$$
$$R(i) + F(i) \geq P - BigM \cdot (1 - W(i)) \qquad \forall i \in [1, N]$$

Ultimately this 'improved' model deploys over 20 constraints to express the basic hoist scheduling problem.

4.2 CP models

The previous models of hoist scheduling in CP [1, 17, 18] all consider the whole period that a job is in process. If there are, for example, 5 jobs in process at any time, this period is 5 times the duration of a cycle (implying the number of jobs must be an input in such models). As a result the constraints become larger the more jobs in process – hence the constraint (7) which typically occurs in CP models but not in IP models. Indeed the central constraint in [18] is equivalent to our constraint (3) but must be applied to every $k \in 1 \ldots J$:

$$R(i-1) + F(i-1) + E(i, j) + k \cdot P \leq R(j) \ \vee$$
$$R(j-1) + F(j-1) + E(j, i) \leq R(j) + k \cdot P$$
$$\forall i, j \in [1, N], k \in [1, J]$$

4.3 Comparison

The IP models in the literature are complicated to understand, and complicated to adapt for new constraints as will become clear in the next section. Their performance when handled by an IP solver could prove to be better than the *CUC1H* model, since these models are tailored for the IP solver. We examine this empirically in Section 6. By contrast, the previous CP models are easier to read and extend than IP models, but suffer from performance problems when there are multiple jobs in process at any time.

5 Multiple hoists and larger capacity tanks

The extension to the *CUC1H* model to cope with the extended CHSP problem is straightforward. We call this the *CECnH* model.

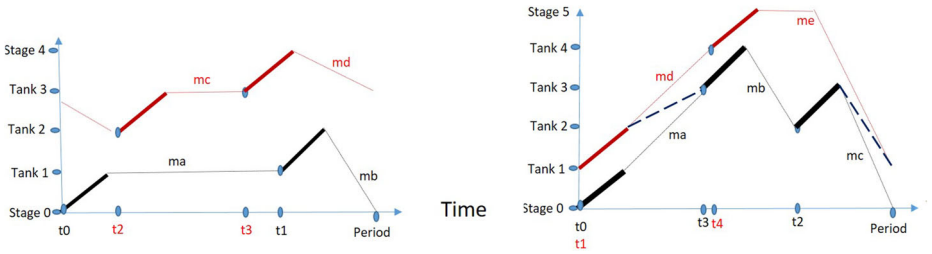

Fig. 3 Left: Two hoists and three tanks. Right: Two hoists and four tanks: avoiding collision on a single track

The constants and data are as described in Section 2. In addition, H denotes the number of hoists, A the number of tracks, and $C(i) \geq 1$ the capacity of tank i. Either $A = H$ in which case each hoist runs on its own track, or $A = 1$ and all the hoists run on one track.[4]

Additional decision variables

$$H(i) \quad i = 1, \ldots, N \text{hoist that performs treatment } i$$

Initial bounds

$$\begin{aligned} 1 \leq H(i) \leq H \qquad & i = 1, \ldots, N \\ 0 \leq B(i) \leq C(i) \qquad & i = 1, \ldots, N \end{aligned}$$

The handling of capacities is simply done by increasing the bound on $B(i)$, so that it is an integer-valued rather than 0, 1. If, for example, $B(i) = C(i)$ then there are $C(i)$ jobs in tank i at the end of the cycle, which is the maximum possible. Constraint (9) replaces (5), which it generalises.

Additional constraints

$$R(i) \leq R(i-1) + F(i-1) + P_{ub} \cdot (C(i) - B(i)) \qquad \forall i \in [1, N] \tag{9}$$

$$\begin{aligned} & H(i) > H(j) \vee \\ & (R(i) + F(i) + E(i+1, j) \leq R(j) + P \wedge R(j) + F(j) + E(j+1, i) \leq R(i) + P) \\ & \quad \forall i \in [1, N],\ j \in [0, i-1] \end{aligned} \tag{10}$$

$$\begin{aligned} & H(i) > H(j) \vee \\ & R(i) + F(i) + E(i+1, j) \leq R(j) \ \vee \ R(j) + F(j) + E(j+1, i) \leq R(i) \\ & \quad \forall i \in [1, N],\ j \in [0, i-1] \end{aligned} \tag{11}$$

If there is only one hoist, it starts the first action (removing a job from tank 0) at the start of a cycle, so no action continues across the end of a cycle. However, as evident from Fig. 3 (left), only one hoist has a treatment starting at time $t0$. For the other hoist a constraint is needed to ensure time for the empty movement md over the end of the cycle. The constraint (10) extends constraint (4) to handle this case.

[4] Other values of A are handled by specifying the set of hoists running on each track.

The constraint (11) is an extension of constraint (3) in the *CUC1H* model. This is the model for $A = 1$, where all the hoists are on the same track.

The interesting case illustrated in Fig. 3 (right) has the thick hoist 'below' the thin one on the track, but the thin hoist removes from tank 1 which is 'below' tanks 2 and 3. In this case the time needed for the dotted 'virtual' empty moves are enforced by (11) giving time for the thin hoist to get out of the way of the thick one.

If each hoist has its own track ($H = A$), we replace $H(i) > H(j)$ by $H(i) \neq H(j)$ in (11) and (10).

By contrast to the *CECnH* model, many new constraints and variables are required to extend the IP model. We give one of the four 'collision avoidance' constraints:

$$\begin{aligned} R(i) + F(i) + E(i+1, j) - R(j) \quad &\leq \\ BigM \cdot (3 - Y(i, j) - Z(i, k) - \sum_{h=k}^{K} Z(j, h)) \quad & \\ \forall i, j \in [1, N], j < i, k \in [1..K] \quad & \end{aligned}$$

Here, $Y(i, j)$ is a binary variable indicating which removal is earlier, $R(i)$ or $R(j)$. $Z(i, k)$ is a further binary variable indicating whether hoist k removes jobs from tank i. Although easier to extend than IP models, previous CP models are also more complex to extend than our model – for example requiring various auxiliary boolean variables.

Researchers tackle many minor variations of a particular CHSP problem. One difference between the *CECnH* model and [5]'s IP model is accounting for the time $p(i)$ required to unload a part from a tank – while the hoist stays at the same location – and the time $q(i)$ required to load the tank. This prevents one hoist loading a tank when another is unloading it. It is handled by three further IP constraints:

$$\begin{aligned} &R(i) + p(i) - (R(i-1) + F(i-1) - q(I)) \leq BigM \cdot (1 - B(i) + W(i-1)) \\ &\quad \forall i \in [1, N] \\ &R(i) + p(i) - (R(i-1) + F(i-1) - q(I)) \leq BigM \cdot (1 - W(i-1) + B(i)) \\ &\quad \forall i \in [1, N] \\ &R(i) + p(i) - (R(i-1) + F(i-1) - q(I) - P) \leq BigM \cdot (2 - B(i) - W(i-1)) \\ &\quad \forall i \in [1, N] \end{aligned}$$

The modification required to the *CUC1H* model to ensure it returns precisely the same solutions is simply to extend constraint (5) thus:

$$R(i) \cdot B(i) \leq R(i-1) + F(i-1) - (p(i) + q(i)) \quad \forall i \in [1, N]$$

5.1 Global constraints

CECnH does not use global constraints. An option is to introduce the cumulative constraint to the model. However, for each of the hoist tasks, its duration depends on where is the next task. This is because there is no variable in the model representing the successor task. Thus cumulative only has a lower bound on the duration and is unlikely to achieve substantial additional pruning to justify its overhead. In order to assess the value of cumulative, we study two variants of *CECnH* with cumulative constraints in Section 6.

6 Empirical analysis

We implemented the extended model of Section 5 in the high-level constraint language MiniZinc [14]. In this section we report empirical results on benchmark CHSP instances. To be compatible with previous benchmarks we adopt the one-track option ($A = 1$), and set $p = q = 0$.

As solvers we first compare the constraint programming ('CP') solver Gecode version 6.2, the integer programming ('IP') solver Gurobi version 9.0.1, and the lazy clause generation ('LCG') solver Chuffed version 0.10.4. We later introduce a straightforward hybrid IP–LCG approach. Default settings were used for each solver , i.e., solvers could ignore any model-specified search heuristics if they wished. We used no specific search or symmetry-breaking annotations.

Experiments were performed on a machine with a 12-core Intel Core i7 processor at 3.2GHz, and 16GB memory. The model was implemented in MiniZinc version 2.4.3 and compiled for each of the solvers using the standard MiniZinc settings and optimisation flag 'O1'. The models and datasets are available at: https://doi.org/10.4121/12912413.[5]

Linearisation for the IP solver was done automatically by MiniZinc [2]. Research on MiniZinc model linearization has ensured even CP MiniZinc models generate good IP models – as proven on MiniZinc benchmark set. Note that IP models in MiniZinc simply map to the same IP model in the underlying solver, so solving time is unaffected.

6.1 Benchmarks

First we take two standard benchmarks from the literature, 'P&U' [16] and 'BO1' [12]. Both datasets originate from real-life industrial lines. Since these benchmarks are not considered difficult for the single-part single-degree CHSP [5], we study extended benchmarks in which the line and treatments are multiplied, as follows.

For each benchmark with tanks $0, \ldots, N$ and multiplier $\mathbf{M} \geq 1$, we create a benchmark with tanks $0, \ldots, N \cdot \mathbf{M}$. For $1 \leq k \leq N$, the properties of the k^{th} tank are copied on to $\mathbf{M}$ further tanks, and similarly the value of $F(k)$. For the empty movements, we want a non-zero time to travel between different copies of the same tank. We do this by adding 5 between the first and second copy, 10 between the first and third copy, etc. These numbers were chosen to be of the same order of magnitude as the other empty movements $E(i, j)$ in the benchmarks. Specifically, let i_{m1} be the $m1^{th}$ copy of tank i, and let j_{m2} be the $m2^{th}$ copy of tank j. Then $E(i_{m1}, j_{m2})$ is set to $E(i, j) + 5 \cdot |m1 - m2|$.

In addition to these extended benchmarks, we also study random instances, generated according to the specification of [5]. These instances have $H \in [2, 3, 4]$ and $N \in \{8, 10, 12, 14\}$, and the time windows uniformly $[0, 0]$, $[0, 50]$ or $[0, 100]$. We further generated larger random instances with Multiplier $\mathbf{M} \in [1, 5]$ and $C(i) \in [1, 3]$, $\forall i$.

We emphasise that no previous model has been benchmarked on instances with a multiplier $\mathbf{M} > 1$.

6.2 Results and discussion

Global constraints with Chuffed First, in a preliminary experiment, we study our *CECnH* model with the Chuffed LCG solver. As noted earlier, the global constraint cumulative can be added to the model as a redundant constraint as follows in MiniZinc:

[5] A version without supplementary files is at the earlier DOI: https://doi.org/10.4121/uuid:211d5c86-ee65-455c-a8ab-9265aab1e289.

Table 1 Results on P&U instances with *CECnH* and the Chuffed LCG solver

Instance	Optimum	*CECnH*	+cumulative	+annotation
1-1-1	521	1.1	3.3	**0.70**
1-2-1	251	1.4	**1.0**	1.4
1-2-2	221	2.5	**1.5**	2.4
1-3-1	170	1.0	**0.98**	1.02
1-3-2	168	1.9	**1.6**	1.9
1-4-1	150	0.36	0.36	**0.35**
1-5-1	150	0.25	**0.23**	0.25
2-1-1	1076	10.4	10.4	**10.3**
2-2-1	448	1590	1400	**1390**
2-2-2	448	**1140**	1390	1470
2-2-3	448	1020	1050	**852**
2-3-1	334	432	449	**371**
2-3-2	334	381	393	**317**
2-3-3	334	344	356	**287**
2-4-1	295	24.3	25.0	**21.0**
3-1-1	1438	**112**	121	125

Time limit 1 hour. 1 core. Times are shown in seconds

```
constraint cumulative( [r[i]|i in 1..N],
                       [f[i] + min([e[i,j]|j in 1..N
                            where j != i])|i in 1..N],
                       [1|i in 1..N],
                       Hoists );
```

Schutt et al. [19] defined specific LCG explanations for cumulative: both for cumulative's time-tabling and time-tabling edge finding.[6] Thus the following MiniZinc annotations can yield more powerful propagation:

```
constraint cumulative(...) :: ttef_filt(true) :: tt_filt(true)
```

For the problem P&U, Table 1 reports the results of Chuffed on *CECnH* with no cumulative constraint (column 3). Column 'Instance' shows the tuple Multiplier – Hoists – Capacity; column 'Optimum' shows the optimal period P. Times are shown in seconds.

Adding the cumulative constraint (column 4) brings a little gain on smaller instances but is a little worse on larger instances, whereas also using the time-tabling annotations (column 5) does bring a consistent gain.

In order to fairly compare across models and solvers, we use the base version of *CECnH*, with no cumulative constraint, in the remainder of the experiments below.

Models compared on real-world instances For the problem P&U, Table 2 shows the results of the three models: our model *CECnH*, the IP model of [5] (re-implemented in MiniZinc), and the CP model of [17] (also re-implemented). Times are shown in seconds. We solve all models with an IP solver, and the two CP models also with a CP solver.

[6] We are grateful to an anonymous reviewer for this suggestion.

Table 2 Results on P&U instances with different models

Instance	Optimum	[17] (CP)	[17] (IP)	[5]'s IP	*CECnH* (CP)	*CECnH* (IP)
1-1-1	521	≤ *731*	2.3	1.1	4.1	**0.89**
1-2-1	251	≤ *731*	**4.4**	12.6	102	4.8
1-2-2	221	≤ *731*	24.4	13.0*	78.8	**7.9**
1-3-1	170	≤ *731*	7.6	30.5	60.9	**3.1**
1-3-2	168	≤ *731*	9.8	26.9*	217	**3.6**
1-4-1	150	≤ *731*	1.7	1.2	18.7	**0.48**
1-5-1	150	≤ *731*	1.4	2.5	12.6	**0.51**
2-1-1	1076	≤ *1182*	209	12.7	454	**10.3**
2-2-1	448	≤ *1182*	≤ *1070*	≤ *513*	≤ *778*	***≤ 455***
2-2-2	448	≤ *1313*	≤ *903*	≤ *513**	≤ *823*	***≤ 448***
2-2-3	448	≤ *1313*	≤ *584*	≤ *513**	≤ *772*	***≤ 450***
2-3-1	334	≤ *1182*	≤ *337*	–	≤ *672*	**2523**
2-3-2	334	≤ *1313*	≤ *339*	–*	≤ *648*	***≤ 334***
2-3-3	334	≤ *1313*	≤ *342*	–*	≤ *648*	***≤ 334***
2-4-1	295	≤ *1152*	146	–	≤ *603*	**59.2**
3-1-1	1438	≤ *2599*	≤ *2869*	220	≤ *1537*	**68.9**

Time limit 1 hour. 1 core. Times are shown in seconds. If the solver could not find and prove the optimum within the time limit, the upper bound obtained is shown in italic. '*' indicates the model proves a sub-optimal solution for the instance. '–' indicates the model cannot solve the instance

Note that [5]'s model does not take account of non-unit tank capacities, and so, while it finds solutions for such instances, the period is sub-optimal. These entries are marked by an asterisk.

From the table we see that our model is faster, overall, than the IP model from the literature by a factor of 1–2, and is faster than the CP model from the literature by up to an order of magnitude when the problem size grows ($\mathbf{M} > 1$) – whether the CP or IP solver is used as the solving backend. Indeed, without search guidance the previous CP model cannot find any optimal solution within the timeout when a CP solver is used on it.

Solvers compared on real-world instances For problems P&U and BO1 respectively, Tables 3 and 4 show the results of the three solvers on our model. Overall, the best performance of solvers using 1 core is by LCG. Below we discuss use of multiple cores, but already from the tables we see that the best performance overall is either from IP with 4 or 8 cores, or from LCG. For P&U, for example, IP is best for the small (Multiplier 1) instances, but struggles in comparison with LCG for the medium instances (Multiplier 2); for the largest instances IP again has the edge. As also seen in Table 2, we observe that the pure CP solver has uniformly inferior performance.

Solvers compared on random instances Table 5 shows results on 360 random instances (10 for each parameter setting), generated with the same parameters as [5]. In this table we show [5]'s IP model in addition to our model *CECnH*. Table 6 shows results on 100 random instances of larger size, generated as described earlier.

As before we compare CP, IP and LCG solvers. The instances of [5] are not difficult for any solver. When using IP, our model (when linearised) is also faster by an order of

Table 3 Results on P&U instances

Instance	Optimum	CP	IP	IP-2core	IP-4core	IP-8core	LCG
1-1-1	521	4.1	0.89	1.1	**0.73**	1.2	1.1
1-2-1	251	102	4.8	2.6	3.1	3.0	**1.4**
1-3-1	170	60.9	3.1	1.5	2.9	2.3	**1.0**
1-3-2	168	217	3.6	2.3	5.0	2.4	**1.9**
1-4-1	150	18.7	0.48	0.37	0.40	**0.34**	0.48
1-5-1	150	12.6	0.51	0.34	0.36	0.37	**0.25**
2-1-1	1076	454	10.3	5.1	5.5	**4.9**	10.4
2-2-1	455	*≤ 778*	*≤ 455*	*≤ 468*	2910	3030	**1590**
2-2-2	448	*≤ 823*	*≤ 448*	*≤ 450*	*≤ 448*	*≤ 457*	**1140**
2-2-3	448	*≤ 772*	*≤ 450*	*≤ 462*	*≤ 450*	*≤ 448*	**1020**
2-2-4	448	*≤ 776*	*≤ 465*	*≤ 462*	2990	2100	**884**
2-2-5	448	*≤ 776*	*≤ 448*	*≤ 476*	*≤ 468*	*≤ 448*	**510**
2-3-1	334	*≤ 672*	2523	3420	1560	1100	**432**
2-3-2	334	*≤ 648*	*≤ 334*	2950	1210	1410	**381**
2-3-3	334	*≤ 648*	*≤ 334*	*≤ 334*	1290	1050	**344**
2-4-1	295	*≤ 603*	59.2	41.9	27.0	120.7	**24.3**
2-4-2	295	*≤ 381*	93.2	43.3	68.3	44.8	**20.7**
2-4-3	295	*≤ 545*	100	85.2	32.6	27.4	**21.2**
2-4-4	295	*≤ 545*	104	66.9	52.3	24.9	**19.5**
2-5-1	278	3420	10.0	10.3	9.8	9.5	**2.5**
2-5-2	278	*≤ 314*	14.7	14.4	14.6	11.7	**2.5**
2-5-5	278	*≤ 341*	14.0	12.7	11.0	9.1	**4.7**
2-6-1	273	1710	2.1	7.0	7.3	**2.1**	2.2
2-7-1	269	510	1.7	2.3	2.3	1.6	**1.3**
2-8-1	269	625	1.9	2.2	2.1	**1.6**	1.8
2-8-8	269	831	3.2	3.5	3.3	2.8	**1.5**
3-1-1	1438	*≤ 1537*	68.9	47.6	45.3	**38.9**	112
3-1-2	1430	*≤ 1526*	51.2	48.3	37.2	**26.0**	109
3-2-1	≤ 751	*≤ 1474*	*≤ 796*	*≤ 826*	*≤ 768*	*≤ 765*	***≤ 751***
3-2-2	≤ 760	*≤ 1526*	*≤ 788*	*≤ 776*	*≤ 768*	***≤ 760***	*≤ 773*
3-2-3	≤ 724	*≤ 1526*	*≤ 772*	*≤ 770*	*≤ 758*	*≤ 743*	***≤ 724***
3-3-1	≤ 506	*≤ 1474*	*≤ 519*	*≤ 488*	*≤ 517*	***≤ 506***	*≤ 514*
3-3-2	≤ 503	*≤ 1526*	*≤ 517*	*≤ 513*	*≤ 517*	***≤ 503***	*≤ 505*
3-4-1	428	*≤ 1474*	322	247	**75.1**	91.6	216
4-1-1	2196	*≤ 2472*	267	332	**162**	513	575
4-1-2	2195	*≤ 2400*	709	605	**192**	264	538
4-1-3	2195	*≤ 2644*	1640	1100	226	**205**	609
4-2-1	≤ 1057	*≤ 2472*	*≤ 1123*	–	***≤ 1032***	*≤ 1109*	*≤ 1057*
4-2-2	≤ 1027	*≤ 2679*	***≤ 1027***	*≤ 1032*	*≤ 1082*	*≤ 1035*	*≤ 1075*
4-3-1	≤ 687	*≤ 2400*	*≤ 709*	*≤ 699*	*≤ 731*	*≤ 694*	***≤ 687***
4-3-2	≤ 688	*≤ 2400*	*≤ 743*	*≤ 719*	*≤ 709*	*≤ 713*	***≤ 688***
4-4-1	≤ 577	–	*≤ 578*	*≤ 583*	***≤ 577***	***≤ 577***	*≤ 582*

Time limit 1 hour. If the solver could not find and prove the optimum within the time limit, the bounds obtained are shown in italic. '–' indicates no (sub-optimal) solution or bounds information returned by the solver

Table 4 Results on BO1 instances

Instance	Optimum	CP	IP	IP-2core	IP-4core	IP-8core	LCG
1-1-1	2819	128	0.82	1.13	**0.66**	1.11	4.5
1-2-1	2400	21.1	0.50	**0.32**	0.36	0.33	10.4
1-3-1	2400	10.6	0.40	0.33	0.33	**0.29**	4.3
1-3-2	1565	*≤ 2173*	1.90	2.96	0.82	**0.64**	2.3
2-1-1	4947	*≤ 5716*	1810	1350	1020	**798**	2720
2-2-1	3282	*≤ 5634*	34.2	15.6	**15.4**	17.7	38.6
2-2-2	3282	*≤ 5206*	89.3	88.5	**20.5**	22.1	62.1
2-2-3	3282	*≤ 5369*	59.0	30.5	**24.1**	30.3	102
2-3-1	5731	*≤ 5731*	19.0	**10.0**	10.6	11.3	32.4
2-3-2	5218	*≤ 5218*	26.4	14.7	16.1	**13.9**	26.2
2-3-3	5335	*≤ 5335*	16.3	16.0	19.0	**15.1**	19.3
3-1-1	≤ 7929	*≤ 8422*	*≤ 8959*	*≤ 8134*	*≤ 8009*	***≤ 7929***	*≤ 8133*
3-1-2	≤ 7792	*≤ 9261*	*≤ 7756*	*≤ 7764*	*≤ 7884*	***≤ 7792***	*≤ 36572*
3-2-1	4807	*≤ 8936*	369	624	**245**	392	727
3-2-2	4807	*≤ 12124*	*≤ 4807*	1190	417	**346**	769
3-2-3	4807	*≤ 8064*	*≤ 4807*	*≤ 4807*	**605**	1200	664
3-3-1	4699	*≤ 8456*	67.0	88.7	**64.3**	65.0	272
3-3-2	4699	*≤ 9250*	254	31.4	58.7	129	522
4-1-1	≤ 10913	*≤ 12091*	*≤ 12809*	*≤ 12674*	*≤ 29198*	*≤ 12983*	***≤ 10913***
4-1-2	≤ 11718	*≤ 16220*	*≤ 61984*	*≤ 49162*	*≤ 48050*	*≤ 46905*	***≤ 11718***
4-1-3	≤ 12060	***≤ 12060***	*≤ 42846*	*≤ 11286*	*≤ 36371*	*≤ 21402*	*≤ 12931*
4-2-1	≤ 6567	*≤ 11829*	*≤ 6677*	***≤ 6567***	*≤ 6574*	*≤ 6615*	*≤ 6919*
4-2-2	≤ 6570	*≤ 12443*	*≤ 6657*	*≤ 6821*	***≤ 6570***	*≤ 6614*	*≤ 6858*
4-3-1	6306	*≤ 13126*	517	506	**387**	406	*≤ 6744*
4-3-2	6306	*≤ 15697*	1880	704	**243**	409	*≤ 6729*
4-4-1	6248	*≤ 14527*	127	266	194	**83.0**	3550

Time limit 1 hour

magnitude than [5]'s IP model. The larger random instances are more difficult, especially for higher values of the Muliplier parameter. The most robust performance is the LCG solver, although for the instances it solves, the IP solver is the fastest.

Effect of multiple cores for IP solving Since the version of Chuffed available to us exploits only a single core, for fair comparison we focused on solvers running on a single core in the experiments so far. However we are interested in exploiting multiple cores, and so

Table 5 Results on random instances generated as [5]

	CP	IP	LCG	[5]'s IP
Number of instances solved (of 360)	360	360	360	360
Mean solving time (of instances solved)	35.9s	5.37s	1.12s	42.3s

Time limit 1 hour. 1 core. Note that all solvers/models could solve all instances with the time limit

Table 6 Results on harder random instances

	CP	IP	LCG
Number of instances solved (of 100)	24	33	77
Mean solving time (of instances solved)	603s	177s	387s
Mean solving time (of instances all solvers solved)	4.6s	0.80s	0.13s

Time limit 1 hour. 1 core.

we studied Gurobi's performance with 2, 4 and 8 cores. These results are also included in Tables 3 and 4, columns 'IP-*n*core'.

The literature suggests that IP "parallelism efficiency decreases for more than 4-8 threads" [9] (see also [4]). We find a similar pattern. The tables' columns 'IP', 'IP-2core', 'IP-4core' and 'IP-8core' indicate that increasing the number of cores for Gurobi up to 4 almost always decreases the runtime (or gives tighter bounds within the time limit). However, when we increase the number of cores further, performance sometimes improves and sometimes declines. Increasing the number of cores to 12 was only neutral or negative to performance.

6.3 Exploring IP–LCG hybridisation

That neither IP and LCG dominate each other on the real-world CHSP benchmarks points to the value of exploring hybrids of the two. We consider two hybrid approaches exploiting the multiple cores of modern processors. The first is a simple portfolio approach, while the second hybrid communicates lower bounds between the solvers.

Consider a machine with $n > 1$ cores. The simple hybrid ('SH') uses $n - 1$ cores for Gurobi and 1 core for Chuffed. Both solvers run on the problem instance, and whichever finishes first gives the solution. If neither prove the optimal solution within the time limit, we take the tightest bound information from the two. In the experiments below, we take $n = 4$ and use 3 cores for Gurobi.

The second hybrid approach runs Gurobi with $g < n$ cores and runs $n - g$ separate parallel Chuffed solver instances with (different) *fixed* period P, i.e., with the constraint $P = p$ for a given p which differs between the Chuffed instances; we called these fixed-chuffeds. We call this the parallel hybrid ('PH') approach. The idea is that the fixed-chuffeds improve the lower bound of the overall search. In particular, let ℓ be the current lower bound of Gurobi. Then in the parallel hybrid, the fixed-chuffeds have as their periods $P = \ell, \ell + 1, \ldots, \ell + n - g - 1$.

The Gurobi instance and the $n - g$ fixed-chuffed instances are coordinated through the current bounds on the period. We implemented this approach using Gurobi's callback functionality and lazy constraints. Whenever Gurobi increases its lower bound ℓ, any fixed-chuffed instance that is thus rendered obsolete (because its $p < \ell$) is terminated.[7] On the other hand, whenever a fixed-chuffed of period p completes, there are two possibilities. For simplicity, suppose the fixed-chuffed of period $p = \ell$ terminates. Either it has proved that $P = p$ is infeasible, in which case we can increment ℓ by 1; or it has found a solution of period $P = p$, which must be the optimal solution since p is the current lower bound.

[7] Similarly when Gurobi improves its upper bound.

Table 7 Results on P&U instances

Instance	Optimum	LCG (1 core)	IP (4 cores)	SH (1,3 cores)	PH (3,1 cores)	PH (2,2)	PH (1,3)
2-2-1	455	1590	2910	1590	**749**	1680	2020
2-2-2	448	1140	[*379*, *448*]	1140	**794**	1340	1550
2-2-3	448	1020	[*340*, *450*]	1020	**843**	1160	1620
2-3-1	334	432	1560	432	**321**	509	559
2-4-1	295	**24**	27	**24**	35	38	37
2-5-1	278	**2.5**	9.8	**2.5**	12	16	8
3-1-1	1438	112	45	38	72	35	**30**
3-2-1	$\leq$ 751	***$\leq$ 751***	[*491*, *768*]	**[*491*, *751*]**	[*567*, *838*]	[*549*, *780*]	[*552*, *815*]
3-2-2	$\leq$ 748	$\leq$ 773	[*491*, *768*]	**[*491*, *748*]**	[*561*, *820*]	[*546*, *781*]	[*549*, *765*]
3-3-1	$\leq$ 507	$\leq$ *514*	[*413*, *517*]	**[*414*, *507*]**	[*475*, *519*]	[*472*, *590*]	[*472*, *523*]
4-1-1	2196	575	**162**	257	458	300	280
4-2-1	$\leq$ 1032	$\leq$ 1057	**[*655*, *1032*]**	**[*655*, *1032*]**	[*670*, *1049*]	[*663*, *1077*]	[*662*, *1096*]
4-3-1	$\leq$ 687	***$\leq$ 687***	[*532*, *731*]	**[*532*, *687*]**	[*613*, *807*]	[*605*, *756*]	[*604*, *806*]

Time limit 1 hour. Times are shown in seconds. If the solver could not find and prove the optimum within the time limit, the bounds obtained are shown in italic. The 'SH' and 'PH' columns are parameterised by a pair (a, b), the number of cores for LCG (a) and IP (b) respectively

Should a fixed-chuffed of period $p > \ell + 1$ finish, a little more book-keeping tells us when we can push up the lower bound or declare the optimum found – or neither as yet.

Table 7 gives the results of the simple and parallel hybrids over 4 cores, compared to 1-core Chuffed and 4-core Gurobi. The table shows selected easy and hard instances from the first real-world dataset. It is worth emphasising that LCG's performance is on 1 core while the other solvers all use 4 cores.

As seen earlier in Table 3 for this problem, LCG and IP perform differently on different instances, while we now see that the hybrids can perform robustly. Only on very easy instances (i.e., $\mathbf{M} = 1$, not shown in the table) the overhead of the parallel hybrid dominates its performance.

The parallel hybrid performs well on the $\mathbf{M} = 2$ instances, and the simple hybrid on $\mathbf{M} = 3$ and $\mathbf{M} = 4$ instances. Note that instances 3–2–2 and 3–3–1 are examples of a problem where Gurobi does better with fewer cores than more (i.e., 3 instead of 4). The best combination of core usage, overall, for the parallel hybrid is 3 cores for fixed-chuffeds and 1 core for Gurobi. On instances where the parallel hybrid has a poorer optimum (upper bound) than the other solvers, i.e., $\mathbf{M} = 3$ and $\mathbf{M} = 4$, it has a better lower bound due to it putting more search effort there.

Taken together, the exploratory results suggest that there is potential in hybrid IP–LCG approaches, as we discuss further at the end of the article.

7 Related work

The problem addressed in this article is cyclic hoist scheduling with identical parts and single-degree schedules [16]. The most popular exact methods for this problem are (mixed) IP (i.e., branch-and-cut) and custom branch-and-bound, such as [22]. The first constraint-based approach was by [1]. Meta-heuristic approaches were proposed by, for instance, [13] who developed a genetic algorithm, and [21] who developed a tabu search and repair procedure.

The same pattern of IP, custom branch-and-bound, or meta-heuristics is found in the literature for variants of the basic CHSP, including for multi-part lines with multi-degree cycles, or systems with multiple lines [3, 7, 20]; a typical example is [8].

Hybrid methods for the CHSP also deserve more attention than in the literature to date. [18] introduced the idea of modelling the generic CHSP with a single, parameterised CP model, and solved it using a loosely-coupled CP–IP hybrid. Other works adopting hybrid methods consider specific CHSP settings. For instance, [6] propose a two-stage approach for two hoists consisting of a custom heuristic followed by IP, while [23] hybridise two forms of evolutionary algorithm.

The HSP is related to but distinct from the problems of (un)loading ships in a container terminal with port and yard cranes [10, 11], and of scheduling cranes in a factory. For the latter problem, [15] present a dynamic decision tree approach, which would be interesting to consider for the HSP.

8 Conclusion and future work

This article re-examined modelling of the cyclic hoist scheduling problem and proposed a new and more simple model. At its heart this constraint programming model has only six constraints, and is easily extended to variants such as multiple hoists and tracks. We applied

state-of-the-art IP and LCG solvers with the new model on a range of benchmarks, including new larger benchmarks than before studied in the literature. Besides presenting the first application of LCG to the CHSP (to our knowledge), we showed that by using MiniZinc we can model the CHSP more simply, compactly and extensibly than previous modelling approaches, and can run IP and LCG solvers on any model instance.

Empirical results show, first, that our model has better computational properties than previous CP-based or IP-based models, and, second, that IP and LCG solvers have different strengths in solving effectiveness. We further presented a parallel hybrid of IP and LCG that coordinates between solvers using lower bounds on the period in a simple way. The results show that hybridisation has potential on more difficult problem instances.

The next step in our work is to extend to more variants of the CHSP, such as problems with multiple parts. Second, we would like to experiment further with modelling using interval constraints, since our models with these constraints (not reported here) are uncompetitive to date. It will also be interesting to explore the possibilities for global constraints in such a representation, especially for LCG solvers [19]. Third, we think there is ready potential to further explore forms of hybridisation that bring together the complementary performance of IP and LCG, for instance through communicating upper bounds, and logic-based Benders decomposition.

Acknowledgements This work was first presented as an abstract at CPAIOR'20. We thank the CPAIOR reviewers and the Constraints reviewers for their suggestions. Thanks also to C. Chu, K. Fleszar, S. van der Laan, W. Lei, K. Leo, G. Tack and F. Wimmenauer.

Data Availability https://doi.org/10.4121/12912413

References

1. Baptiste, P., Legeard, B., Varnier, C. (1992). Hoist scheduling problem: an approach based on constraint logic programming. In *Proc. of ICRA'92* (pp. 1139–1144).
2. Belov, G., Stuckey, P.J., Tack, G., Wallace, M. (2016). Improved linearization of constraint programming models. In *Proc. of CP'16* (pp. 49–65).
3. Boysen, N., Briskorn, D., Meisel, F. (2017). A generalized classification scheme for crane scheduling with interference. *European Journal of Operational Research*, *258*(1), 343–357.
4. Carle, M.A. (2012). Using more processors does not necessarily lead to reduced run times on CPLEX. http://www.thequestforoptimality.com/using-more-processors-does-not-necessarily-lead-to-reduced-run-times-on-cplex/. accessed: 2019-11-21.
5. Che, A., Lei, W., Feng, J., Chu, C. (2014). An improved mixed integer programming approach for multi-hoist cyclic scheduling problem. *IEEE Transaction Automation Science and Engineering*, *11*(1), 302–309.
6. Chtourou, S., Manier, M., Loukil, T. (2013). A hybrid algorithm for the cyclic hoist scheduling problem with two transportation resources. *Computers & Industrial Engineering*, *65*(3), 426–437.
7. Feng, J. (2017). Modélisation et optimisation des Hoist Scheduling Problems. Ph.D. thesis université Paris-Saclay / Northwestern Polytechnical University (China).

8. Feng, J., Chu, C., Che, A. (2018). Cyclic jobshop hoist scheduling with multi-capacity re-entrant tanks and time-window constraints. *Computers & Industrial Engineering*, *120*, 382–391.
9. IBM Support (2018). Effects of multithread execution in CPLEX Optimization Studio models. https://www.ibm.com/support/pages/effects-multithread-execution-cplex-optimization-studio-models. accessed: 2019-11-21.
10. Jonker, T., Duinkerken, M.B., Yorke-Smith, N., de Waal, A., Negenborn, R.R. (2019). Coordinated optimization of equipment operations in a container terminal. Flexible Services and Manufacturing Journal.
11. Kizilay, D., Eliiyi, D.T., Hentenryck, P.V. (2018). Constraint and mathematical programming models for integrated port container terminal operations. In *Proc. of CPAIOR'18* (pp. 344–360).
12. Leung, J.M., Zhang, G., Yang, X., Mak, R., Lam, K. (2004). Optimal cyclic multi-hoist scheduling: a mixed integer programming approach. *Operations Research*, *52*(6), 965–976.
13. Lim, J.M. (1997). A genetic algorithm for a single hoist scheduling in the printed-circuit-board electroplating line. *Computers & Industrial Engineering*, *33*(3-4), 789–792.
14. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G. (2007). Minizinc: Towards a standard CP modelling language. In *Proc. of CP'07* (pp. 529–543).
15. Peterson, B., Harjunkoski, I., Hoda, S., Hooker, J.N. (2014). Scheduling multiple factory cranes on a common track. *Computers & OR*, *48*, 102–112.
16. Phillips, L.W., & Unger, P.S. (1976). Mathematical programming solution of a hoist scheduling program. *AIIE Transactions*, *8*(2), 219–225.
17. Riera, D., & Yorke-Smith, N. (2002). An improved hybrid model for the generic hoist scheduling problem. *Annals of Operations Research*, *115*(1-4), 173–191.
18. Rodošek, R., & Wallace, M. (1998). A generic model and hybrid algorithm for hoist scheduling problems. In *Proc. of CP'98. pp. 385–399*.
19. Schutt, A., Feydy, T., Stuckey, P.J. (2013). Explaining time-table-edge-finding propagation for the cumulative resource constraint. In *Proc. of CPAIOR'13. pp. 234–250*.
20. Varnier, C., Bachelu, A., Baptiste, P. (1997). Resolution of the cyclic multi-hoists scheduling problem with overlapping partitions. *INFOR: Information Systems and Operational Research*, *35*(4), 309–324.
21. Yan, P., Che, A., Yang, N., Chu, C. (2012). A tabu search algorithm with solution space partition and repairing procedure for cyclic robotic cell scheduling problem. *International Journal of Production Research*, *50*(22), 6403–6418.
22. Yan, P., Chu, C., Yang, N., Che, A. (2010). A branch and bound algorithm for optimal cyclic scheduling in a robotic cell with flexible processing times. *International Journal of Production Research*, *48*(21), 6461–6480.
23. Yan, P., Wang, G., Che, A., Li, Y. (2016). Hybrid discrete differential evolution algorithm for biobjective cyclic hoist scheduling with re-entrance. *Computers & OR*, *76*, 155–166.