A Generic Model and Hybrid Algorithm for Hoist Scheduling Problems

Robert Rodošek and Mark Wallace

IC-Parc, Imperial College, SW7-2AZ London
E-mail: {rr5, mgw}@icparc.ic.ac.uk

Abstract. This paper presents a robust approach to solve Hoist Scheduling Problems (HSPs) based on an integration of Constraint Logic Programming (CLP) and Mixed Integer Programming (MIP). By contrast with previous dedicated models and algorithms for solving classes of HSPs, we define only one model and run different solvers.

The robust approach is achieved by using a CLP formalism. We show that our models for different classes of industrial HSPs are all based on the same generic model. In our hybrid algorithm search is separated from the handling of constraints. Constraint handling is performed by constraint propagation and linear constraint solving. Search is applied by labelling of boolean and integer variables.

Computational experience shows that the hybrid algorithm, combining CLP and MIP solvers, solves classes of HSPs which cannot be handled by previous dedicated algorithms. For example, the hybrid algorithm derives an optimal solution, and proves its optimality, for multiple-hoists scheduling problems.

1 Introduction

1.1 The Hoist Scheduling Problem

Many industrial processes employ computer-controlled hoists for material handling [2, 4, 14, 16, 22]. The hoists are programmed to perform a fixed sequence of moves repeatedly. Each repetition of the sequence of moves is called a *cycle* and the total time required by the hoists to complete the cycle is called a *cycle time*. A typical application is an automated electroplating line for processing printed circuit boards (jobs). The importance of minimising the cyclic time is evident by the fact that the lot sizes for electroplating jobs are usually large and a production run may require weeks between changeovers [22]. Even a small reduction in the cycle time can result in a significant saving of time and cost.

Due to the nature of different industrial processes, specific approaches for different classes of HSPs have been developed. There are two drawbacks of the proposed approaches. First, the models and solution algorithms are dedicated to each class of problems, and second, an optimal solution and the proof of optimality has been shown only for restricted classes of HSPs [10].

1.2 A Generic Model and Solver for HSPs

This paper addresses all the different classes of HSPs using a single generic approach. This approach uses Constraint Logic Programming (CLP) as the modelling language, and models the different classes of HSPs by extending a single underlying model with extra constraints. The models are handled by a generic algorithm which uses a combination of constraint propagation and linear solving.

The HSP problem involves both linear constraints and logical constraints, and CLP is a powerful language for modelling such problems. The language used in this paper to model HSPs is CLP(R)[1], an instance of the CLP scheme [13]. However while the CLP Scheme envisages a single constraint solver for all the constraints, this paper follows [3] and passes constraints to either (or both) of two different solvers. In this setup, we can separate the definition and the behaviour of constraints. A practical consequence is that the programmer can concentrate on modelling of the problem and any problems with the performance of the default behaviour can be ironed out afterwards.

The hybrid algorithms for solving HSPs combines a CLP solver and a MIP solver such that both solvers share the variables and constraints to cooperate in finding an optimal solution. The experiments on HSPs have shown that there are problem classes which can be solved neither by constraint propagation nor MIP alone, but which succumb to this combination of the two.

Moreover the combination is not an exclusive one, where some constraints are handled by one solver and the remainder by the other solver. Indeed every constraint is passed to both solvers. This application, therefore, shows the value of allowing a single constraint to be handled by more than one solver.

1.3 Outline of the Paper

In this paper two contributions are presented. First, models for classes of HSPs can be defined independently of any solver which will be used during the search for an optimal solution. Second, the proposed hybrid algorithm derives a schedule with the minimal cyclic time and proves its optimality for classes of HSPs which cannot be handled by previous dedicated CLP and MIP solvers.

The remainder of this paper is organised as follows. Section 2 presents related work. Section 3 models different classes of HSPs. Section 4 demonstrates how to use a CLP formalism to apply different solvers on the same model. Section 5 presents the computational experience with the proposed hybrid approach. Finally, Section 6 concludes the paper.

[1] By CLP(R) we mean constraint logic programming over numerical equations and inequations. The implementation we use is ECLiPSe [9], which supports not only linear constraint solving, but also finite domain propagation, and various other constraint handling facilities.

2 Related Work

2.1 Hoist Scheduling Models and Programs

Previous approaches to minimise the cycle time of HSPs are mostly mathematical programming-based approaches [15, 16, 18, 22]. Recently, several constraint programming-based approaches have been developed [2, 4]. In the following we present classes of HSPs, the models, and the solution algorithms which have been used in the proposed approaches.

Phillips and Unger [18] used a mixed integer programming model to determine a schedule with the minimum cyclic time for a real *one-hoist* scheduling problem with 12 chemical treatment tanks. The Phillips and Unger's (P&U's) HSP has become a benchmark problem in several follow-up studies.

Shapiro and Nuttle [22] introduced a branch-and-bound procedure and used linear programming on different subproblems to bound the search space.

Lei and Wang [15] introduced a heuristic algorithm for *two-hoists* scheduling problems with both hoists on the same track. The algorithm uses a *partitioning* approach by which the production line is partitioned into two sets of contiguous tanks and each hoist is assigned to a set. Lei et. al. [14] introduced also another heuristic algorithm for the class of *two-hoists* scheduling problems with both hoists on the same track. In contrast to the algorithm in [15], the movements of both hoists must be scheduled to *avoid traffic collisions*. The algorithm is not able to guarantee the optimal solution.

Baptiste et. al. [2] presented advantages and drawbacks of different kinds of constraint programming-based approaches. All approaches demonstrated that the good versatility of CLP language allows one to develop very rapidly computational models for different classes of HSPs. The empirical results show that CLP with a linear solver (Prolog III) is more effective than constraint propagation over finite domains in dealing with the HSPs. Prolog III was able to produce an optimal one-hoist schedule for the P&U's problem in 30 minutes and the finite domain solver in 306 minutes on SUN station Sparc 4/60. To increase the power of consistency control within the constraints solver, the disjunctive constraints have been used in an active way to reduce the domain of each variable. This modification of the CLP approach helps the rational solver to derive the optimal schedule in 40 seconds.

All HSPs introduced above have been shown by the authors to belong to the class of NP-hard problems [10]. Existing approaches which derive the minimal cycle time are limited to the single-hoist cases and use branch-and-bound procedures, whose efficiency quickly diminishes as the number of tanks in the system increases. Scheduling two or more hoists further expands the search space, and makes the task of searching for the global optimal solution extremely difficult.

2.2 MIP, CLP and Hybrid Algorithms

Hoist scheduling problems involve both logical and linear constraints. A pioneer in the combination of logical constraint solving and OR techniques is

Hooker [11]. An implemented system which combines logic and linear programming is 2LP [17]. A mathematical modelling language and implementation that interfaces to both CLP and linear solvers is presented in [1].

One of the first CLP platforms to support both constraint propagation and linear solving was CHIP [5]. However the solvers were not designed to handle constraints with shared variables. A CLP implementation supporting the combination of constraint propagation and linear programming with shared constraints and variables was described in [3].

The linear constraint solvers of these two systems were internal ones, and lacked the scaleability of commercial matrix-based implementations such as CPLEX [6] and XPRESS-MP [23]. Nevertheless several researchers identified problems and problem classes that could not be handled by the major MIP packages, but could be solved using constraint propagation. Examples were party planning [20] and machine allocation [8].

In 1995 an integration was developed between the CLP platform ECLiPSe and the commercial packages CPLEX and XPRESS-MP [21]. This was used to build a hybrid algorithm solving a fleet scheduling problem [12]. Subsequently an automatic translator was built to map models expressed in ECLiPSe to MIP models [19]. The resulting MIP model can be solved using constraint propagation and search in ECLiPSe; linear solving and search in an external MIP package; or linear solving in an external package, and propagation and search in ECLiPSe. The paper [19] describes how this system was used to improve on the CLP results of [8, 20], and to solve a number of other problems.

The current paper uses the same implementation to solve to optimality some problem classes that have never previously been solved, neither using CLP nor MIP techniques.

3 Modelling the Different HSP Classes

In this section we present natural models for several classes of HSPs. The models are not tailored for any solver, and models for different classes are obtained by simply adding or changing the relevant constraints: no remodelling is attempted.

The modelling language syntax is based on Edinburgh Prolog, with some extensions to make the models easier to read.[2]

- `for(El,Min:Max) do` **Goal**, applies the goal to each number in the range.
- A functional syntax can be used where an integer is expected. For example the goal `plus(2,1) < 5` is expanded into the goal `plus(2,1,N), N<5`.
- Finally an array syntax is defined such that `Array[N]` picks out the `N`th member of the array `Array`, for example: `Var = f(5,3,1), X is Var[2]` which instantiates `X` to 3.

3.1 Single Cyclic Scheduling

Informal Description The simplest cyclic HSP contains a single hoist and the number of sequential chemical treatment tanks in the production line. Each

[2] These extensions have been implemented in ECLiPSe.

tank applies chemical or plating treatments, such as H_2SO_4 activating or Nickel plating, to the jobs. A large number of identical jobs is placed at the initial stage of the production line and these jobs have to be processed in the order that tanks are sequenced. The hoist is programmed to handle the inter-tank moves of the jobs, where each move consists of three simple hoist operations: (i) lift a job from a tank; (ii) move to the next tank; and (iii) submerge the job in that tank. Upon completion of a move, a hoist travels to another tank for the next scheduled move. Both the hoist travelling times and the times to perform moves are given constants. The time of each move is independent of the move direction. A hoist can carry one job at a time, and no buffer exists between tanks. A job must remain in each tank for a certain amount of time, between a minimum and a maximum: this is the tank's *time window*. The fixed sequence of moves that the hoist performs in each cycle is defined by a *one-hoist* cyclic schedule [18]. Exactly one job is removed from each tank in a cycle, and therefore, one job enters and one job leaves the production line in a cycle. Figure 1 represents a one-hoist scheduling problem with 6 tanks and three jobs present in the system simultaneously.

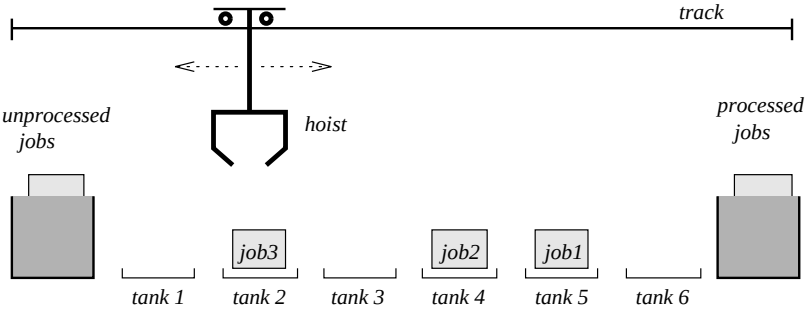

Fig. 1. A one-hoist scheduling problem

CLP Model This class of one-hoist scheduling problems can be captured by the following model. The model is expressed in CLP syntax: input data is expressed as facts; constraints as clauses; variables start with an upper-case letter. We use a bold font for types; for example `numTanks(Integer)` means that the predicate `numTanks` takes a single argument which is an integer.

- `numTanks(Integer).` The number (=12) of chemical treatment tanks in the production line.
- `empty(Tank, Tank, Time).` **Tank** is an integer denoting a tank. **Time** is also an integer. The predicate `empty` is used to record the times of travel from tank to tank for the hoist when empty. It records times for every pair of tanks.
- `full(Tank, Time).` The transport times of jobs from a tank to the next tank on the production line.
- `numJobs(Integer).` The number (=3) of jobs in the production line.

- `minTime(Tank, Time)`. The minimum time a job can stay in each tank.
- `maxTime(Tank, Time)`. The maximum time a job can stay in each tank.

The problem variables are those whose value is found during the search for the shortest possible cycle:

- `Entry`. An array of the times at which the jobs are put into the tanks.
- `Removal`. An array of the times at which the jobs are removed from the tanks.
- `Period`. The time of a cycle.

A single cyclic HSP with time windows can be defined by four types of constraints.

First, we relate the array of variables `Entry` to the array of decision variables `Removal`. For each tank, the entry time is the removal time from the previous tank, plus the transportation time between the two tanks. The "0th" tank is the stack of unprocessed jobs.

```
lin1(Removal,Entry):-
   for(Tank,1:numTanks) do
      Removal[Tank-1] + full(Tank-1) = Entry[Tank].
```

Second, the entry time of a job to a tank is related straightforwardly to its removal time from this tank: the difference between them is equal to the treatment time in the tank. Due to the nature of chemical treatments, the processing of a job in a tank must be completed within a given time window. These intervals impose *time window constraints* on the hoist movements. Scheduling with time window has been studied by Phillips and Unger [18].

```
lin2(Removal,Entry) :-
   for(Tank,1:numTanks) do
      Entry[Tank] + minTime(Tank) ≤ Removal[Tank],
      Entry[Tank] + maxTime(Tank) ≥ Removal[Tank].
```

Third, since one job is removed from the production line during each cycle the time of any job in the system cannot be longer than the time of $NumJobs$ cycles.

```
lin3(Removal,Period) :-
   Removal[numTanks] + full(numTanks) ≤ numJobs * Period.
```

Fourth, the hoist can only do one thing at a time. Thus a constraint is required to prevent the hoist transporting a job from tank T_1 to $T_1 + 1$ at the same time as it is transporting the same, or another, job from tank T_2 to $T_2 + 1$. A clash will obviously occur if a task is being performed on a job at the same time as any other task on the same job. Less obviously a clash will occur if a task is being performed on a job at time `Time`, and another task is being performed on the same job at time `Time+Period`. In this case the clash will be between the first task on one job and the second task on the following job. In fact a clash will occur if two tasks are being performed on a job at any pair of times `Time` and `Time+N*Period`, for N up to but not including the number of jobs at the same time on the production line.

To ensure no clash between the transportation from T_1 to $T_1 + 1$ and T_2 to $T_2 + 1$, either one job must be removed from tank T_1 after the other was placed

in tank $T_2 + 1$, leaving time for the hoist to travel empty from tank $T_2 + 1$ to tank T_1; or the job must be placed in tank $T_1 + 1$ before being removed from tank T_2, leaving time for the hoist to travel empty from $T_1 + 1$ to T_2.

This constraint is the core of the generic HSP model. It is expressed in terms of the following clauses:

```
disj(Removal,Entry,Period) :-
   for(T1,1:numTanks-1) do  for(T2,T1+1:numTanks) do
      for(K,1:numJobs-1) do
         disj1(T1,T2,K,Removal,Entry,Period).

disj1(T1,T2,K,Removal,Entry,Period) :-
   Entry[T1+1] + empty(T1+1,T2) + K * Period ≤ Removal[T2].
disj1(T1,T2,K,Removal,Entry,Period) :-
   Entry[T2+1] + empty(T2+1,T1) ≤ Removal[T1] + K * Period.
```

This disjunctive constraint is superficially similar to the resource constraints encountered in disjunctive scheduling, which enforce that one task is performed either before or after another task [7]. However the disjunctive constraints in hoist scheduling involve not just the two task variables, `Entry[T1]` and `Removal[T2]` for example, but they also involve a third variable `Period`. The occurrence of a third variable makes the handling of the hoist scheduling disjunctive constraints quite different. In case there are only two variables, choosing one alternative for each disjunctive constraint, together with propagation, suffices to decide the global consistency of the constraints. In case three variable are involved, by contrast, choosing disjuncts and propagating without failing, no longer suffices to guarantee global consistency.

The whole single cyclic scheduling problem with time windows, denoted by `hsp1`, is defined as follows:

```
problem hsp1:
   minimize  Period
   subject to lin1(Removal,Entry), lin2(Removal,Entry),
              lin3(Removal,Period), disj(Removal,Entry,Period).
```

Other classes of HSPs can be defined by an extension or a small modification of this generic HSP model.

3.2 Scheduling with Tank Capacity

Each tank has a finite capacity. There is a limit to the number of jobs it can treat at any one time. The papers on one-hoist scheduling problems consider usually the maximum capacity of each tank equal to one.

- `tankCapacity(Tank, Integer).` The capacity of each tank.

The tanks must never contain more jobs than their capacities. So for tank T_1, a job must be removed before the arrival of the job which is `tankCapacity(I)` jobs (=cycles) behind it on the production line. We introduce the following constraint:

```
lin4(Removal,Entry,Period) :-
   for(Tank,1:numTanks) do
      Removal[Tank] - Entry[Tank] ≤ tankCapacity(Tank) * Period.
```

The P&U's problem with tank capacity is denoted by `hsp2` and it contains constraints `lin1`, `lin2`, `lin3`, `lin4`, `disj`.

3.3 Scheduling with Multiple Hoists on One Track

HSP can use two or more hoists on the same track. In the multiple hoist problem we need a variable which associates a hoist to each activity - transporting a job from one tank to the next. Data:

- `numHoists(Integer)`. The number of hoists (=2) in the system.

Variable:

- **Hoist** An array recording which hoist is assigned to each activity. `Hoist[T]` is the hoist which transports a job from tank `T` to `T+1`.

The multiple hoist problem can be modelled by changing only the disjunctive constraint `disj` from the previous model. The new disjunctive constraint only differs by allowing an extra alternative: the case where the two activities are performed by different hoists.

We assume the hoists are numbered, with the hoists further along the track towards the higher-numbered tanks also having higher numbers. Because the hoists cannot pass each other, if two activities do overlap in time, the activity involving the higher-numbered tank must be performed by the higher-numbered hoist.

```
mhdisj(Removal,Entry,Hoist,Period) :-
   for(T1,1:numTanks-1) do  for(T2,T1+1:numTanks) do
      for(K,1:numJobs-1) do
         mhdisj1(T1,T2,K,Removal,Entry,Hoist,Period).

mhdisj1(T1,T2,K,Removal,Entry,_H,Period) :-
   Entry[T1+1] + empty(T1+1,T2) ≤ Removal[T2] + K * Period.
mhdisj1(T1,T2,K,Removal,Entry,_H,Period) :-
   Entry[T2+1] + empty(T2+1,T1) + K * Period ≤ Removal[T1].
mhdisj1(T1,T2,K,Removal,Entry,Hoist,Period) :-
   /* Comment: T2>T1 */
   Hoist[T2] ≥ Hoist[T1] + 1.
```

The whole P&U's problem with multi hoists on one track is denoted by `hsp3` and it contains constraints `lin1`, `lin2`, `lin3`, `lin4`, `mhdisj`.

Notice that Lei and others introduced two quite different models for the 2-hoist problem, the *partitioning* model [15], and the *traffic-collision* approach [14]. `hsp3` models the traffic-collision approach. The weaker partition approach could be modelled by adding the single constraint

```
partition(Hoist) :-
   for(I,1:numTanks-1) do  for(J,I+1:numTanks) do
      Hoist[J] ≥ Hoist[I].
```

3.4 Scheduling with Multiple Tracks

HSPs can contain hoists on more than one track. Previous approaches toward solving cyclic HSPs have been limited to single-track cases. Our model for that problem is very similar to the model for the previous class of HSPs. Since the hoists use different tracks it is enough to force that the hoists for each tank are different. We adapt the model of `hsp3` by adding a single extra clause to the procedure for `mhdisj1`, viz:

```
mhdisj1(T1,T2,K,Removal,Entry,Hoist,Period) :-
   Hoist[T1] ≥ Hoist[T2] + 1.
```

The whole P&U's problem with multi hoists on different tracks is denoted by `hsp4` and it contains `lin1`, `lin2`, `lin3`, `lin4`, and the modified `mhdisj`.

4 Deriving Models for Different Solvers

We use the CLP formalism for modelling and solving HSPs. Once a CLP program has been developed for a class of HSPs it is relatively easy, compared with the mathematical programming approach, to adapt the model for other classes of HSPs and run different solution algorithms.

4.1 Modelling

CLP has greater expressive power than traditional mathematical programming models in two ways:

- Constraints involving disjunction can be represented directly
- Constraints can be encapsulated (as predicates) and used in the definition of further constraints

However, a CLP model can be automatically translated into a traditional MIP model by

- Eliminating disjunctions in favour of auxiliary boolean variables
- Unfolding predicates into their definitions

This translation is only applicable on condition that any recursively defined constraints can be fully unfolded at the time of translation. This condition is satisfied in the HSP model, and all the other large scale industrial optimisation problems we have addressed. The translation is presented in detail in [19]. We briefly summarise the key steps, and present a toy example.

- For each possible top level goal, `p(X1, ..., Xn)` add to the program a single clause

  ```
  p(X1...Xn) :- p(X1...Xn,1).
  ```

  The extra final argument is an input boolean (1 imposes the constraint, whilst 0 would relax it).

– Each predicate definition

  `disj(X1...Xn)` **:- Body1.**
  ...
  `disj(X1...Xn)` **:- BodyN.**

  is translated into a single-clause predicate

  `disj(X1...Xn,B)` **:- Body1[B1], ..., BodyN[BN],** `B1+...+BN=B.`

  The bodies **Bodyi[Bi]** are produced by adding an extra argument `Bi` to every goal.
– Each linear constraint `X` $\leq$ `Y` is translated into another linear constraint `X + m * B` $\leq$ `Y + m` where `m` is the number $(hi(X) - lo(Y))$. Value $hi(X)$ is the upper bound of X and $lo(Y)$ is the lower bound of Y. Note, every translated linear constraint is equivalent to the original constraint if the auxiliary binary variable is instantiated to 1; and it is true for every value of X and Y within their ranges, if it is instantiated to 0.

After translation, the resulting CLP program has no choice points. When the input data of a given problem are supplied, the translated program is automatically unfolded into a conjunction of linear constraints.

4.2 Example

The program

```
prog(X, Y) :- X::1..10, Y::1..10, diff(X, Y).
diff(X, Y) :- Y+2 ≤ X.
diff(X, Y) :- X+2 ≤ Y.
```

is translated into the program:

```
prog(X, Y) :- prog(X, Y, 1).
prog(X, Y, B) :- X::1..10, Y::1..10, diff(X, Y, B).
diff(X, Y, B) :- Y+2+ B1×11 ≤ X+11, X+2+ B2×11 ≤ Y+11, B1+B2 = B.
```

The goal `prog(3,Y)` is unfolded into the constraints:

```
1 ≤ Y ≤ 10, Y+2+ B1×11 ≤ 3+11, 3+2+ B2×11 ≤ Y+11, B1+B2 = 1.
```

4.3 Solving

The proposed evaluation algorithm allows an integration of MIP with CLP using a *unique model* for a problem. The derived linear constraints are treated either by the MIP solver, or the CLP solver, or by both solvers. Our hybrid algorithm combines both solvers such that search is separated from the handling of constraints. Search is applied by labelling of boolean and integer variables. Constraint handling is performed by constraint propagation of the CLP solver and linear constraint solving of the MIP solver.

We have implemented the integration of CLP with MIP by using the ECLiPSe constraint logic programming platform and the XPRESS-MP mathematical programming package [9, 23]. This allows XPRESS to be used to solve problems modelled in ECLiPSe. The control of the search process and the constraint propagation is handled by CLP while the linear constraint solving is handled by MIP. The constraint propagation is performed by a consistency algorithm on finite domains and it represents a component of the ECLiPSe package. On the other hand, the linear constraint solving is performed by the simplex algorithm which is a component of the XPRESS package.

Communication between the solvers is supported by the attributed variables of ECLiPSe. For the purposes of the hoist scheduling problem, the information communicated is just the upper and lower bounds of the variables.

Naturally the main performance benefit of the hybrid solver is due to the early detection of failure by the different solvers. Each solver detects certain failures which would not have been detected by the other solver until a later node in the search tree. For example constraint propagation fails immediately with constraints,

```
X::0..2, Y::0..2, X+2*Y = 3, X-Y = 1.
```

but cannot detect any inconsistency in the constraints:

```
X::1..10, Y::1..10, 2*X+2*Y ≥ 20, X+Y ≤ 11.
```

A linear solver has precisely the complementary behaviour, detecting the inconsistency of the second constraint set, but not the first.

On the HSP problem, we apply the following three solvers:

- **CLP solver.** The constraints are considered only by ECLiPSe. The search is done by labelling binary variables first. Constraint handling is performed by constraint propagation on finite domains.
- **MIP solver.** The constraints are considered only by XPRESS. The search is done by performing the default XPRESS branch-and-bound procedure. Constraint handling is performed by linear constraint solving (simplex). The optimal solution of the whole problem is returned to ECLiPSe.
- **CLP&MIP solver.** The constraints are considered by ECLiPSe and XPRESS. The search is done by labelling binary variables first. Constraint handling is performed by constraint propagation on finite domains and linear constraint solving.

5 Empirical Results

Let us discuss the empirical results of the hybrid CLP&MIP solver relative to the results of the CLP and MIP solvers on the following HSPs from Section 2:

The empirical results in Table 2 show that HSPs are hard for our CLP solver. However, the flexibility of CLP gives the programmer the choice of a variety

Table 1. Hoist scheduling problems

`hsp1`	the P&U's problem with 12 tanks and 4 jobs
`hsp2`	the P&U's problem with the tank capacity equal to 2
`hsp3`	the collision-based P&U's problem with 2 hoists on one track
`hsp4`	the P&U's problem with 2 hoists on two tracks

of constraint solvers and variable domains. Baptiste et al. [2] compared two domains and associated constraint solvers from different CLP languages: CHIP's finite domains and Prolog III's rational numbers. Their empirical results show when a constraint solver should be chosen in preference to another constraint solver and how to control the search towards an efficient running program.

The MIP solver has difficulties to derive an optimal solution to all HSPs. The solver is very efficient for problems `hsp1` and `hsp2`, and inefficient for problems `hsp3`, and `hsp4`.

Table 2. Characteristics of the solvers on different HSPs

The CLP solver:

	Time (1st sol.)	Time (opt.sol.)	Time (proof opt.)	Min. cycle time	FD fails
hsp1	1204 sec	> 60 min	-	-	> 20000
hsp2	3371 sec	> 60 min	-	-	> 20000
hsp3	> 60 min	-	-	-	> 20000
hsp4	> 60 min	-	-	-	> 20000

The MIP solver:

	Time (1st sol.)	Time (opt.sol.)	Time (proof opt.)	Min. cycle time	Nodes processed
hsp1	4 sec	6 sec	7 sec	521	1200
hsp2	6 sec	8 sec	8 sec	521	1521
hsp3	7 sec	> 60 min	-	-	> 50000
hsp4	6 sec	> 60 min	-	-	> 50000

The CLP&MIP solver:

	Time (1st sol.)	Time (opt.sol.)	Time (proof opt.)	Min. cycle time	FD fails	LP fails
hsp1	19 sec	73 sec	105 sec	521	1338	502
hsp2	28 sec	76 sec	102 sec	521	1399	521
hsp3	218 sec	926 sec	961 sec	395	4179	1768
hsp4	36 sec	68 sec	185 sec	379	1300	92

By applying the CLP&MIP solver, simplex and the constraint propagation on finite domains helped to derive an optimal solution and to prove its optimality. It follows that the proposed constraint handling is very useful procedure by cutting

the solution space and deriving an optimal solution to the HSPs in reasonable time. There is a certain level of orthogonality between constraint propagation and linear constraint solving. The number of FD-failures and the number of LP-failures show that both constraint handling procedures are needed to prune the search space. Since constraint propagation is performed before linear constraint solving it is difficult to say which procedure is more important.

All timings in Table 2 are in CPU seconds running on a SUN-SPARC/20. "FD-fails" denotes the number of failures by performing constraint propagation and "LP-fails" denotes the number of failures by linear constraint solving.

We show the robustness of the hybrid approach by solving 100 randomly generated two-hoists HSPs with multiple tracks. The problems represent modifications of problem $hsp4$ in Section 2. The limits on processing times, $Min(i)$ and $Max(i)$, $i = 1, ..., NumTanks$, are determined by drawing values from two sampling functions, $f_{Min}(i) = Min(i) - 10 + 20 * r_1$ and $f_{Max}(i) = Max(i) - 10 + 20 * r_2$, where r_1 and r_2 are $(0, 1)$ uniform random numbers [15]. Once the values of these processing time limits are computed, they are used as constants. The hoist travelling times are also determined in a similar way using a sampling function $f_{Full}(i) = Empty(i, i + 1) + 15 + 10 * r_3$.

Table 3. The CLP&MIP solver on 100 randomly generated hsp4

Derived optimal solutions	100
Min. time	167 sec
Max. time	1146 sec
Avg. time	314 sec

Table 3 represents the minimum, maximum, and average computation times needed to derive optimal schedules of the generated HSPs. The results demonstrate that the hybrid solver is a robust algorithm and successfully derives an optimal solution and proves its optimality to all HSPs with two hoists on different tracks.

6 Conclusions

We have presented models for several classes of industrial HSPs and an efficient translation to a generic model for different solution techniques, i.e. the CLP solver, the MIP solver, or hybrid CLP&MIP solvers. These models and solvers have been benchmarked on some problems which have been the subject of previous research both using CLP, MIP and heuristic algorithms.

The proposed CLP&MIP solver can solve several classes of HSPs which have never previously been solved to optimality. Neither the CLP solver nor the MIP solver alone are able to solve them in reasonable time. The experimental results demonstrated that constraint propagation and linear constraint solving are orthogonal up to certain degree. An infeasibility of several HSPs is recognised by

only one of the procedures. The proposed integration of CLP with MIP allows comparisons between the two approaches and gives a clearer idea of when CLP should be chosen in preference to MIP, and when an integrated solver is quicker than the CLP solver or the MIP solver.

However the HSP experiments have revealed an unexpected, but very important, benefit of hybrid solvers. The experiments show that, if constraints are passed to both a constraint propagation engine and a linear solver, the robustness of the model may be dramatically enhanced. The same generic model can be easily and naturally adapted for all the different classes of HSPs, and they can all be solved.

Using either MIP or CLP solvers alone, problem modelling is made harder because models must be designed specifically for the solver, as for example in [14, 15, 16, 18, 22]. In fact, when the same generic model is solved by a CLP or an MIP solver alone, only a subset of the different problem classes can be effectively handled. Our hope is that hybrid solvers may make it possible to simplify problem modelling, by reducing the need to address issues of solver efficiency at the modelling stage.

Our work makes a contribution to the long-term objective of separating the modelling and solving of combinatorial problems. With the powerful modelling facilities of CLP, with multiple solvers and flexible search control, the encoding of a correct model of the problem can indeed be a guaranteed step towards an efficient running program. The consequences can be revolutionary - with programmers actually taking modelling seriously.

Acknowledgements Many thanks to the IC-PARC group who has contributed to the writing of this paper through helpful discussions and criticisms.

References

1. Barth, P., Bockmayr, A.: Modelling Mixed-Integer Optimisation Problems in Constraint Logic Programming. *MPI Report Nr. I-95-2-011* (1995)
2. Baptiste, P., Legeard, B., Manier, M.A., Varnier, C.: A Scheduling Problem Optimisation Solved with Constraint Logic Programming. *Proc. of the PACT Conf.* (1994) 47-66
3. Beringer, H., De Backer, B.: Combinatorial Problem Solving in Constraint Logic Programming with Cooperating Solvers. Chapter 8 in *Logic Programming: Formal Methods and Practical Applications* ed. C. Beierle and L. Pluemer Elsevier (1995)
4. Cheng, C.C., Smith, S.F.: A Constraint Satisfaction Approach to Makespan Scheduling. *Proc. of the AIPS Conf.* (1996) 45-52
5. Dincbas, M., Van Hentenryck, P., Simonis, H., Aggoun, A., Graf, T., Berthier, F.: The Constraint Logic Programming Language CHIP. *Proc. of the FGCS Conf.* (1988) 693-702
6. CPLEX. Using the CPLEX Callable Library. *CPLEX Optimization, Inc.* (1997)
7. Dincbas, M., Simonis, H., Van Hentenryck, P.: Solving Large Combinatorial Problems in Logic Programming. *Journal of Logic programming* **8** (1995) 75-93
8. Darby-Dowman, K., Little, J., Mitra, G., Zaffalon, M.: Constraint Logic Programming and Integer programming Approaches and their Collaboration in Solving an Assignment Scheduling Problem. *Constraints* **1(3)** (1997) 245-264

9. ECLiPSe User Manual Version 3.7.1. *IC-PARC, Imperial College, London* (1998)
10. Hanen. C.: Study of a NP-Hard Cyclic Scheduling Problem: The Recurrent Job-Shop. *European Journal of Operations Research* **72** (1994) 82-101
11. Hooker, J.N., Osorio, M.A.: Mixed Logical/Linear Programming. *Proc. of the INFORMS CSTS Conf.* Atlanta (1996)
12. Hajian, M., Sakkout, H.El, Wallace, M., Richards, E.: Towards a Closer Integration of Finite Domain Propagation and Simplex-Based Algorithms. *Proc. of the AI Maths Conf.* Florida (1995) `www.icparc.ic.ac.uk/papers.html`
13. Jaffar, J., Lassez, J.L.: Constraint Logic Programming. *Proc. of the ACM POPL Symposium* Munich (1997)
14. Lei, L., Armstrong, R., Gu, S.: Minimizing the Fleet Size with Dependent Time-Window and Single-Track Constraints. *Operations Res. Letters* **14** (1993) 91-98
15. Lei, L., Wang, T.J.: The Minimum Common-Cycle Algorithm for Cycle Scheduling of Two Material Handling Hoists with Time Window Constraints. *Management Science* **37(12)** (1991) 1629-1639
16. Lei, L., Wang, T.J.: Determining Optimal Cyclic Hoist Schedules in a Single Hoist Electroplating Line. *IEE Transactions* **26(2)** (1994) 25-33
17. McAloon, K., Tretkoff, C.: *Optimization and Computational Logic.* Wiley-Interscience (1996)
18. Phillips, L.W., Unger, P.S.: Mathematical Programming Solution of a Hoist Scheduling Problem. *AIIE Transactions* **8(2)** (1976) 219-255
19. Rodošek, R., Wallace, M.G., Hajian, M.T.: A New Approach to Integrating Mixed Integer Programming with Constraint Logic Programming. *Annals of Operational Research. Recent Advance in Combinatorial Optimization: Theory and Applications* (to appear) `www.icparc.ic.ac.uk/papers.html`
20. Smith, B.M., Brailsford, S.C., Hubbard, P.M., Williams, H.P.: The Progressive Party Problem: Integer Linear Programming and Constraint Programming Compared. *Constraints* **1(2)** (1996) 119-138
21. Schimpf. J.: ECLiPSe Approach to Solver Integration and Cooperation. *Proc. of the INFORMS CSTS Conf.* Monterey (1998)
22. Sharpio, G.W., Nuttle, H.: Hoist Scheduling for a PBC Electroplating Facility. *IIE Transactions* **20(2)** (1988) 157-167
23. Dash Associates. XPRESS-MP Reference Manual. *Dash Associates UK* (1993)

This article was processed using the LaTeX macro package with LLNCS style