Scheduling Optional Tasks with Explanation

Andreas Schutt[1], Thibaut Feydy[1], and Peter J. Stuckey[1,2]

[1] Optimisation Research Group, National ICT Australia
[2] Department of Computing and Information Systems,
The University of Melbourne, Victoria 3010, Australia
{andreas.schutt,thibaut.feydy,peter.stuckey}@nicta.com.au

Abstract. Many scheduling problems involve reasoning about tasks which may or may not actually occur, so called optional tasks. The state-of-the-art approach to modelling and solving such problems makes use of interval variables which allow a start time of $\perp$ indicating the task does not run. In this paper we show we can model interval variables in a lazy clause generation solver, and create explaining propagators for scheduling constraints using these interval variables. Given the success of lazy clause generation on many scheduling problems, this combination appears to give a powerful new solving approach to scheduling problems with optional tasks. We demonstrate the new solving technology on well-studied flexible job-shop scheduling problems where we are able to close 36 open problems.

1 Introduction

Many resource-constrained scheduling problems involve reasoning about tasks which may or may not actually occur, so called optional tasks. The state-of-the-art approach in Constraint Programming (CP) to modelling and solving such problems makes use of so-called *interval variables* [12] which represent a start time, end time, and duration of a task, or $\perp$ indicating the task does not run. Propagation algorithms can update the possible start and end times of a task, without knowing whether the task actually runs or not.

In 2008, Laborie and Rogerie [12] introduce interval variables for resource-constrained scheduling to IBM ILOG CP Optimizer [11] as a "first-class citizen" variable type for CP systems. In that work and later follow up work [13,14], they show how to handle these variables in the context of planning and scheduling. The benefits of interval variables are not only in giving a neat conceptual model for representing optional tasks, but also the additional propagation obtained that is possible by reasoning on start and end times even without knowing whether a task executes. However, interval variables do not come for free, they may introduce additional variables into a model, and their propagation is more complex.

Standard CP systems that do not support interval variables are still able to model and solve problems with optional tasks, but suffer from the weaker propagation. For example, each optional task can be associated with a Boolean

variable representing whether it executes or not [1,2] or a non-optional task composed of exclusive optional tasks can be associated with an index variable representing which one of the optional tasks runs [16]. In order to strengthen the propagation, special global constraints have been introduced (see, *e.g.*, [1,2,16]).

For CP systems that support interval variables, propagation algorithms have been proposed for the resource constraints `disjunctive` and `cumulative` with optional tasks (see, *e.g.*, [26,31,30,29]). These algorithms record tentative start and end times of optional task and once the optional task is known to execute these become the actual start and end time variables.

In this paper, we not only show how to mimic interval variables with integer variables, but also how propagators defined for constraints on optional tasks can be extended to explain their propagation, which is required for CP solvers with learning. One of those solvers is a lazy clause generation (Lcg) (Lcg) [19] solver which has proven to be remarkably effective on many scheduling problems defining the state-of-the-art in Rcpsp [24,23], Rcpsp/max [25], and RcpspDc [22] problems. We implement the handling of optional tasks in the re-engineered version of Lcg solver [7], and then demonstrate the combination on the well-studied flexible job shop scheduling problem, where we are able to close a number of open instances.

2 Preliminaries

At first, we introduce lazy clause generation and then scheduling with optional tasks.

2.1 Lazy Clause Generation

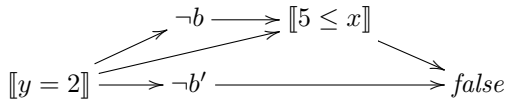
CP solves constraint satisfaction problems by interleaving propagation, which remove impossible values of variables from the domain, with search, which guesses values. All propagators are repeatedly executed until no change in domain is possible, then a new search decision is made. If propagation determines there is no solution then search undoes the last decision and replaces it with the opposite choice. If all variables are fixed then the system has found a solution to the problem. For more details see, *e.g.*, [21].

We assume we are solving a constraint satisfaction problem over set of variables $x \in \mathcal{V}$, each of which takes values from a given initial finite set of values or *domain* $D^0(x)$. The domain D keeps track of the current set of possible values $D(x)$ for a variable x. Define $D \sqsubseteq D'$ iff $D(x) \subseteq D'(x), \forall x \in \mathcal{V}$. The constraints of the problem are represented by propagators f which are functions from domains to domains which are monotonically decreasing $f(D) \sqsubseteq f(D')$ whenever $D \sqsubseteq D'$, and contracting $f(D) \sqsubseteq D$. If all values are removed from one domain of a variable x, *i.e.*, $D(x) = \emptyset$ then the constraints cannot be satisfied with the search decisions made and a failure is triggered. Given a domain D then $lb_D(x) = \min D(x)$ and $ub_D(x) = \max D(x)$. We will omit the subscript D when the domain is clear from the context.

We make use of CP with learning using the LCG [19] approach. Learning keeps track of what caused changes in domain to occur, and on failure computes a *nogood* which records the reason for failure. The nogood prevents search making the same incorrect set of decisions again.

In an LCG solver integer domains are also represented using Boolean variables. Each variable x with initial domain $D^0(x) = [l..u]$ is represented by two sets of Boolean variables $[\![x = d]\!], l \leq d \leq u$ and $[\![x \leq d]\!], l \leq d < u$ which define which values are in $D(x)$. We use $[\![x \neq d]\!]$ as shorthand for $\neg[\![x = d]\!]$, and $[\![d \leq x]\!]$ as shorthand for $\neg[\![x \leq d-1]\!]$. An LCG solver keeps the two representations of the domain in sync. For example if variable x has initial domain [0..5] and at some later stage $D(x) = \{1, 3\}$ then the literals $[\![x \leq 3]\!], [\![x \leq 4]\!], \neg[\![x \leq 0]\!], \neg[\![x = 0]\!]$, $\neg[\![x = 2]\!], \neg[\![x = 4]\!], \neg[\![x = 5]\!]$ will hold. Explanations are defined by clauses over this Boolean representation of the variables.

Example 1. Consider a simple constraint satisfaction problem with constraints $b \rightarrow x + 3 \leq y$, $\neg b \rightarrow y + 3 \leq x$, $b' \rightarrow y \leq 3$, $\neg b' \rightarrow x \leq 3$, with initial domains $D^0(b) = D^0(b') = \{0, 1\}$, and $D^0(x) = D^0(y) = \{0, 1, 2, 3, 4, 5, 6\}$. There is no initial propagation. Setting $[\![y = 2]\!]$ makes the first constraint propagate $D(b) = \{0\}$ with explanation $[\![y = 2]\!] \rightarrow \neg b$, then the second constraint propagates $D(x) = \{5, 6\}$ with explanation $\neg b \wedge [\![y = 2]\!] \rightarrow [\![5 \leq x]\!]$. The third constraint propagates $D(b') = \{0\}$ with explanation $[\![y = 2]\!] \rightarrow \neg b'$ and the last constraint sets $D(x) = \emptyset$, with explanation $[\![5 \leq x]\!] \wedge \neg b' \rightarrow \mathit{false}$. The graph of the implications is



Any cut separating the decision $[\![y = 2]\!]$ from *false* gives a nogood. The simplest one is $[\![y = 2]\!] \rightarrow \mathit{false}$. □

2.2 Scheduling and Optional Tasks

Scheduling applications deal with non-optional and optional tasks. A typical task is specified by a *start time* variable S_i and a *processing time/duration* d_i (which may also be variable). For simplicity we assume durations are fixed, it is easy to extend the results of the paper to variable durations. Given a task and current domain D we define the *earliest start time* $ect_i = lb_D(S_i)$, *earliest completion time* $ect_i = lb_D(S_i) + d_i$, *latest start time* $lst_i = ub_D(S_i)$, and *latest completion time* $lst_i = ub_D(S_i) + d_i$.

Some tasks need resources, such as *e.g.*, labour, space, or particular machinery, from a limited pool for their execution. A schedule of those tasks must ensure that the demand on a resource does not exceed the resource capacity in any time period. In this work, we consider *renewable resources* characterised by the constant *resource capacity* R over time. Such a resource can be modelled by

the *cumulative constraint*

$$\texttt{cumulative}([S_1,\ldots,S_n],[d_1,\ldots,d_n],[r_1,\ldots,r_n],R) \equiv \left(\forall\tau : \sum_{i=1}^{n} runs_{i\tau}\cdot r_i \leq R\right) \ ,$$

where τ is a time period, r_i is the *resource usage* of task i, and $runs_{i\tau}$ expresses whether task i runs at time period τ.

The disjunctive constraint `disjunctive`, requiring that no two tasks are executing at the same time, encodes the special case of cumulative when the resource capacity is 1, and the resource usage for each task is 1.

$$\texttt{disjunctive}([S_1,\ldots,S_n],[d_1,\ldots,d_n]) \equiv \texttt{cumulative}([S_1,\ldots,S_n],[d_1,\ldots,d_n],[1,\ldots,1],1)$$

Specialised propagation algorithms [26,31] are available for the `disjunctive` constraint.

Laborie and Rogerie [12] introduce interval variables to represent optional tasks. The domain of an interval variable ranges over $\bot \cup \{[s,e) \mid s,e \in \mathbb{Z}, e \geq s\}$. A fixed interval variable represents either an *absent* interval $\bot$ or a *present* interval $[s,e)$. Accordingly, an optional task is absent or present if its interval is absent or present respectively. If the interval $[s,e)$ is present then s and e respectively represent its start and end time and $e-s$ its length and it must be that $s \leq e$. If a is an interval variable then let s^a, e^a, and x^a denote the start time, end time, and presence state, respectively.

A task 0 can be composed of other tasks $1,\ldots,n$ and modelled with interval variables a_0 and $a_1,\ldots,a_n$, respectively. Then the relation between the tasks is described via a *span* constraint [14]:

$$span(a_0,\{a_1,\ldots,a_n\}) \equiv \begin{cases} & (x_0^a \leftrightarrow \bigvee_{i=1}^{n} x_i^a) \\ \wedge & (s_0^a = \min_{1\leq i\leq n : x_i^a} s_i^a) \\ \wedge & (\leftrightarrow e_0^a = \max_{1\leq i\leq n : x_i^a} e_i^a) \end{cases} \tag{1}$$

That is, task 0 starts when the earliest task in $\{1,\ldots,n\}$ that is present starts, and ends when the latest task that is present ends. It is present iff at least one of tasks $1,\ldots,n$ is present.

An important specialisation of the span constraint is the *alternative* constraint [14] which allows only one task $1,\ldots,n$ to be present (thus representing a choice for task 0).

$$alternative(a_0,\{a_1,\ldots,a_n\}) \equiv \begin{cases} & \sum_{i=1}^{n} x_i^a \leq 1 \\ \wedge & span(a_0,\{a_1,\ldots,a_n\}) \end{cases} \tag{2}$$

Note if the task 0 is present then exactly one task in $\{1,\ldots,n\}$ is present too; otherwise all are absent.

3 Modelling Optional Tasks

The crucial requirement for effective modelling of optional tasks is to be able to reason about finite domain integer variables which have an additional value $\bot$, which we will call $\texttt{int}_\bot$ variables. These variables can then represent start times of optional tasks. They can also be useful for other reasoning, for example reasoning about databases with null values. In this section we show how to model $\texttt{int}_\bot$ variables using integer and Boolean variables. We then discuss how to model compositional constraints such as *span* and *alternative*. Finally we discuss how tracking implications between presence of tasks can be modelled, to help improve propagation.

3.1 Integers with Bottom

LCG solvers do not currently support $\texttt{int}_\bot$ variables. But we can make use of existing integer and Boolean variables to model an $\texttt{int}_\bot$ variable and thus interval variables.

We model an $\texttt{int}_\bot$ variable S with initial domain $D^0(S) = [l^S..u^S]$ and $\bot$ as $S = (\underline{S}, \overline{S}, x^S)$ using two integer variables $\underline{S}$, $\overline{S}$, and a Boolean variable x^S: $\underline{S}$ holds the lower bound of the $\texttt{int}_\bot$ variable S; while $\overline{S}$ holds the upper bound of the $\texttt{int}_\bot$ variable S; and x^S holds the presence state of the $\texttt{int}_\bot$ variable. The initial domains are $D^0(\underline{S}) = l^S..u^S + 1$ are $D^0(\overline{S}) = l^S - 1..u^S$. The representatives $(\underline{S}, \overline{S}, x^S)$ are constrained by

$$intbot(S) \quad \equiv \quad x^S \leftrightarrow (\underline{S} = \overline{S}) \quad \wedge \quad \neg x^S \leftrightarrow \underline{S} > u^S \quad \wedge \quad \neg x^S \leftrightarrow \overline{S} < l^S$$

Thus if the $\texttt{int}_\bot$ variable S is present, *i.e.*, $S \neq \bot$, the lower and upper bound are identical. If the lower and upper bound are not compatible, *i.e.*, $\underline{S} > \overline{S}$, then the $\texttt{int}_\bot$ variable must be absent, *i.e.*, $S = \bot$, and if the $\texttt{int}_\bot$ variable is absent we set the lower bound to $u^S + 1$ and the upper bound to $l^S - 1$. Note $\underline{S} < \overline{S}$ never holds.

The constraint $\underline{S} \geq v$ represents that $S \geq v \vee S = \bot$. The constraint $\overline{S} \leq v$ represents that $S \leq v \vee S = \bot$.

Propagation on the $\texttt{int}_\bot$ variable is enforced using the appropriate bound. Hence a new (tentative) lower bound $S \geq v$ is enforced by $\underline{S} \geq v$, and a new (tentative) upper bound $S \leq v$ is enforced as $\overline{S} \leq v$. Asserting that $S \neq v$ is enforced by $\underline{S} \neq v \wedge \overline{S} \neq v$. Asserting $S = v$ if S is present is enforced by $\underline{S} \geq v \wedge \overline{S} \leq v$. Two integer variables are required to model an $\texttt{int}_\bot$ variable so that if the bounds cross we do not get a domain wipe-out, which would incorrectly trigger a failure.

Care must be taken in using the tripartite representation of $\texttt{int}_\bot$ variables, because of the special role taken by the *sentinel values* $u^S + 1$ for $\underline{S}$ and $l^S - 1$ for $\overline{S}$. If a propagator ever tries to set $\underline{S} \geq k$ where $k > u^S + 1$, this should be replaced by setting $\underline{S} \geq u^S + 1$. Similarly if a propagator ever tries to set $\overline{S} \leq k$ where $k < l^S - 1$, we should instead set $\overline{S} \leq l^S - 1$. Since propagators are aware that they are dealing with $\texttt{int}_\bot$ variables, they can be modified to act accordingly, without changing the integer variables used to represent $\underline{S}$ and $\overline{S}$.

Given we have $\texttt{int}_\perp$ variables, we can model an interval variable a as a pair (S, d) of an $\texttt{int}_\perp$ variable $S = (\underline{S}, \overline{S}, x^S)$ and an integer d by $x^a = x^S$, $s^a = lb(\underline{S})$, and $e^a = ub(\overline{S}) + d$. Note that [12,13,14] introduce interval variables as an abstract type for tasks and here we consider tasks with fixed duration, thus an end time variable is not required.

3.2 Compositional Constraints

The *span* constraint can be modelled using $\texttt{int}_\perp$ variables and constraints supported by most CP solvers as follows:

$$
\begin{aligned}
&\mathit{span}((S_0, d_0), [(S_1, d_1), \ldots, (S_n, d_n)]) \\
&\equiv \begin{cases}
& \underline{S}_0 \geq \min\{\underline{S}_i + (1 - x_i^S)(u_0^S - u_i^S) \mid 1 \leq i \leq n\} \cup \{u_0^S + 1\} \\
\wedge & \overline{S}_0 \leq \max\{\overline{S}_i + (1 - x_i^S)(l_0^S - l_i^S) \mid 1 \leq i \leq n\} \cup \{l_0^S - 1\} \\
\wedge & \underline{d}_0 \geq \min\{\underline{S}_i + d_i - \overline{S}_0 + (1 - x_i^S)(u_0^d + 1 - d_i) \mid 1 \leq i \leq n\} \cup \{u_0^d + 1\} \\
\wedge & \overline{d}_0 \leq \max\{\overline{S}_i + d_i - \underline{S}_0 + (1 - x_i^S)(l_0^d - 1 - d_i) \mid 1 \leq i \leq n\} \cup \{l_0^d - 1\} \\
\wedge & x_0^S \geq \sum_{i=1}^n x_i^S \quad ,
\end{cases}
\end{aligned}
$$

The interval S_0 is constrained to be lie around the S_i that are present. The duration interval d_0 is constrained to be large enough to reach the minimal end time of tasks that is present, and small enough not to reach beyond the last possible end time of a task which is present. Note the last element in each line ensures that none of the upper or lower bound variables is every bound too strongly to remove the sentinel value.

The *alternative* constraint can be modelled similarly. It propagates more strongly if it is modelled directly rather than making use of *span*. The model is:

$$
\begin{aligned}
&\mathit{alternative}((S_0, d_0), [(S_1, d_1), \ldots, (S_n, d_n)]) \\
&\equiv \begin{cases}
& \underline{S}_0 \geq \min\{\underline{S}_i + (1 - x_i^S)(u_0^S - u_i^S) \mid 1 \leq i \leq n\} \cup \{u_0^S + 1\} \\
\wedge & \overline{S}_0 \leq \max\{\overline{S}_i + (1 - x_i^S)(l_0^S - l_i^S) \mid 1 \leq i \leq n\} \cup \{l_0^S - 1\} \\
\wedge & \underline{d}_0 \geq \min\{d_i + (1 - x_i^S)(u_0^d + 1 - d_i) \mid 1 \leq i \leq n\} \cup \{u_0^d + 1\} \\
\wedge & \overline{d}_0 \leq \max\{d_i + (1 - x_i^S)(l_0^d - 1 - d_i) \mid 1 \leq i \leq n\} \cup \{l_0^d - 1\} \\
\wedge & x_0^S = \sum_{i=1}^n x_i^S \quad ,
\end{cases}
\end{aligned}
$$

The duration d_0 is easier to model since it must be one of the durations of the alternatives. The last constraint enforces that exactly one optional task is actually present if the task 0 is present.

3.3 Presence Implications

Laborie and Rogerie [12] illustrate how reasoning about the presence of optional tasks can substantially improve propagation. The key knowledge is, given two tasks, i and j, does the presence of i imply the presence of j, *i.e.*, $x_i^S \rightarrow x_j^S$.

Such knowledge allows one to perform propagation on i using the information of j even when the presence of both tasks is still unknown. This relationship might initially be available in the modelling stage or might dynamically become available during the solving stage.

Define $impl(i, j)$ as the representation of $x_i^S \rightarrow x_j^S$ we shall use in explanation. For models where there is no information about relative presence we just use $impl(i, j) = x_j^S$. If presence implications can be statically determined from the model we can define the representation statically, hence $impl(i, j) = true$ if task i is present then so must be j, and x_j^S otherwise. We also add the constraint $x_i^S \rightarrow x_j^S$ to enforce the presence relationship.

For models where the relative execution information is dynamically determined we introduce new Boolean variables $I_{i,j}$ to represent the information and let $impl(i, j) = I_{i,j}$. We also add a transitivity constraint $transitive(I, [x_1^S, \ldots, x_n^S])$ which ensures that $I_{i,j} \wedge I_{j,k} \rightarrow I_{i,k}$ and $I_{i,j} \leftrightarrow (\neg x_i^S \vee x_j^S)$. In practice the Boolean variables $I_{i,j}$ can be created as required during the execution, they do not all need to be created initially. Our use of *transitive* corresponds to the logical network of [12].

Example 2. Suppose we have a model with tasks i, j, and k and variable sum where we know that $x_i^S \rightarrow x_j^S$, and if $sum \geq 0$ then $x_i^S \rightarrow x_k^S$, but nothing else about presence implications. For this model we have that $impl(i, j) = true$, $impl(i, k) = I_{i,k}$ where $sum \geq 0 \rightarrow I_{i,k}$ and $I_{i,k} \leftrightarrow (\neg x_i^S \vee x_k^S)$. Since we can never determine any presence implications between j and k, $impl(j, k) = x_k^S$, and similarly $impl(k, j) = x_j^S$, $impl(k, i) = impl(j, i) = x_i^S$. □

4 Explanations for Propagation with Optional Tasks

Propagation with optional tasks requires the generation of explanations for the use in a CP solver with nogood learning. Here, we present explanations for pruning on lower bounds of the start time variables making use of generalised precedences, detectable precedences, and time-table, and energetic reasoning propagation. Pruning on corresponding upper bounds is symmetric and thus omitted. These explanations are extensions of the explanation presented in [24,23] and the same generalisation steps apply for optional tasks for creating a strongest explanation as possible. However, we omit consideration of generalisation here, since it works equivalently to the non-optional tasks case.

For the remainder of this paper, we only consider optional tasks. A non-optional task with a start time variable S and duration d can be represented as an optional task with start time $\underline{S} = \overline{S} = S$ and $x^S = true$ and duration d. While we only consider fixed durations, the explanations can all be extended to use variable durations by replacing d with $lb(d)$ and adding literals $[\![lb(d) \leq d]\!]$ to explanations.

We assume a given domain D, for which we are defining explanations. We lift the definitions of lst_i and ect_i to optional tasks.

$$lst_i := ub(\overline{S}_i) \qquad ect_i := lb(\underline{S}_i) + d_i \qquad lct_i := ub(\overline{S}_i) + d_i \qquad est_i := lb(\underline{S}_i)$$

If $lst_i < ect_i$ then we say the task i has a *compulsory part* $[lst_i, ect_i)$.

Generalised Precedences Given the constraint $S_j + v \leq S_i$ where S_i and S_j are $\texttt{int}_\perp$ variables and v is an integer, then we can propagate on the lower bound of S_i if $impl(i, j)$ is currently known to be *true*. The lower bound is $est_j + v$. In order to prevent the wipe out of all values in $\underline{S}_i$ if the new bound is greater than $u_i^S + 1$ we reduce it to this. Consequently, only an update to $\min(est_j + v, u_i^S + 1)$ is permissible. The corresponding explanation is

$$impl(i, j) \wedge [\![est_j \leq \underline{S}_j]\!] \rightarrow [\![\min(est_j + v, u_i^S + 1) \leq \underline{S}_i]\!]$$

Note that the explanation holds regardless of whether i or j executes.

We can extend this reasoning to half-reified [6] precedences of the form $b \rightarrow S_j + v \leq S_i$ by simply adding b to the left hand of the explanation.

Example 3. Suppose that $S_k + 3 \leq S_i$ for the tasks described in Example 2. Suppose $I_{i,k}$ is currently true, and $D(\underline{S}_i) = [2..5]$ and $D(\underline{S}_k) = [6..10]$. The we propagate $[\![9 \leq \underline{S}_i]\!]$ assuming $u_i^S \geq 8$ with an explanation $I_{i,k} + [\![3 \leq \underline{S}_k]\!] \rightarrow [\![9 \leq \underline{S}_i]\!]$. Suppose instead that $u_i^S = 7$, then we propagate with explanation $I_{i,k} + [\![3 \leq \underline{S}_k]\!] \rightarrow [\![8 \leq \underline{S}_i]\!]$ which will cause $x_i^S = \mathit{false}$. □

Detectable Precedences Given the constraint $\texttt{disjunctive}([S_1, \ldots, S_n], [d_1, \ldots, d_n])$ over n tasks with start time $\texttt{int}_\perp$ variables S_i and fixed duration d_i, $1 \leq i \leq n$. Then two tasks i, j can not be run concurrently if $lst_j < ect_i$ and we can conclude that j must finish before i ($j \ll i$) if they are both present. If we detect that currently $lst_j < ect_i$ holds and also $impl(i, j)$ then we can propagate as in the case above. The new bound is $\min(ect_j, u_i^S + 1)$ with explanation:

$$impl(i, j) \wedge [\![t + 1 - d_i \leq \underline{S}_i]\!] \wedge [\![\overline{S}_j \leq t]\!] \rightarrow [\![\min(ect_j, u_i^S + 1) \leq \underline{S}_i]\!]$$

where t can be any integer in $[lst_j, ect_i)$.

Time-Table Propagation Given n tasks which are competing for a resource with capacity R. Then $\texttt{cumulative}([S_1, \ldots, S_n], [d_1, \ldots, d_n], [r_1, \ldots, r_n], R)$ must hold. Let i be a task for which we want to propagate the lower bound and Ω be subset of tasks $\{j \mid 1 \leq j \neq i \leq n\}$ which are known to be present if i is present, *i.e.*, $impl(i, j), j \in \Omega$ are known to be true currently. If the tasks $j \in \Omega$ create a compulsory part overlapping the interval $[begin, end)$, *i.e.*, $lst_j \leq begin$ and $end \leq ect_j$, and it holds that $begin < ect_i$ and $r_i + \sum_{j \in \Omega} r_i > R$ then the lower bound of S_i can be updated to $\min(end, u_i^S + 1)$. If $ect_i < end$ then LCG solvers break down the propagation in several steps, so that $ect_i \geq end$ holds for the interval considered (see [24] for details). Then, the point-wise explanation [24] is

$$[\![end - d_i \leq \underline{S}_j]\!] \wedge \bigwedge_{j \in \Omega} impl(i, j) \wedge [\![end - d_j \leq \underline{S}_j]\!] \wedge [\![\overline{S}_j \leq end - 1]\!] \rightarrow [\![\min(end, u_i^S + 1) \leq \underline{S}_i]\!]$$

Explaining conditional task overload requires a set of tasks $\Omega \subseteq \{1, \ldots, n\}$ that are all either present together or none present, that is all of $impl(i, j)$ currently hold for $\{i, j\} \in \Omega$, and all have a compulsory part overlapping $[begin, end)$ where $\sum_{i \in \Omega} r_i > R$. Then none of the tasks in Ω can be present, which can be explained as:

$$\bigwedge_{\{i,j\} \in \Omega} impl(i, j) \wedge \bigwedge_{j \in \Omega} [\![t - d_j \leq \underline{S}_j]\!] \wedge [\![\overline{S}_j \leq t - 1]\!] \rightarrow \bigwedge_{j \in \Omega} \neg x_j^S \ ,$$

where t can be any value in $[begin, end)$. Note that this explanation creates $|\Omega|$ clauses due to the conjunction on the right hand side.

Energetic Reasoning Propagation Given n tasks which are competing for a resource with capacity R. Then $\texttt{cumulative}([S_1, \ldots, S_n], [d_1, \ldots, d_n], [r_1, \ldots, r_n], R)$ must hold. Let i be a task for which we want to propagate the lower bound and Ω be subset of tasks $\{j \mid 1 \leq j \neq i \leq n\}$ which are known to be present if i is present, *i.e.*, $impl(i, j), j \in \Omega$ are known to be true currently. If the tasks $j \in \Omega$ are partially processed in the interval $[begin, end)$, *i.e.*, $begin < ect_j$ and $lst_j < end$ for $j \in \Omega$, then the lower bound of S_i can be updated to $min(begin + \lceil rest/r_i \rceil, u_i^S + 1)$ if $begin < ect_i$, $rest > 0$, and $min(d_i, end - begin) + \sum_{j \in \Omega} r_j \cdot p_j(begin, end) > R \cdot (end - begin)$ where

$$rest = \sum_{j \in \Omega} r_j \cdot p_j(begin, end) - (R - d_i) \cdot (end - begin) \qquad \text{and}$$

$$p_j(begin, end) = \max(0, \min(ect_j - begin, end - lst_j, end - begin)) \quad j \in \Omega \ .$$

Thus, the explanation is as follows with $t = min(begin + \lceil rest/r_i \rceil, u_i)$.

$$[\![begin - d_i < \underline{S}_i]\!] \wedge \bigwedge_{j \in \Omega} impl(i, j) \wedge [\![\overline{S}_j \leq end - p_j(begin, end)]\!] \wedge \bigwedge_{j \in \Omega} [\![begin + p_j(begin, end) - d_j \leq \underline{S}_j]\!] \rightarrow [\![t \leq \underline{S}_i]\!]$$

Note that t might not be the largest lower bound for this update, but just as for time-table propagation, for LCG solvers using energetic reasoning it is preferable to perform a step-wise update (see [25] for details). Moreover, since energetic reasoning generalises (extended) edge-finding and time-tabling edge-finding propagation, the explanation presented covers these cases too.

5 Experiments on Flexible Job Shop Scheduling

Experiments were carried out on challenging flexible job-shop scheduling problems (FJSP) [5] where we seek a minimal makespan. FJSP consists of a set of jobs J to be executed on a set of machines M. Each job $j \in J$ is made up of a sequence of tasks $T_{j1}, \ldots T_{jn_j}$, and the tasks can be executed on different machines which may cause them to have different duration. Executing a task T_{jk} on machine $m \in M$ requires d_{jkm} time. The aim is to complete all the tasks in the minimum amount of time.

5.1 Model

For FJSP instance, we model each task T_{jk} using a integer start time variable S_{jk} and duration variable d_{jk} (if the processing time of the task differs on different machines), as well as $\texttt{int}_\perp$ start time variables S_{jkm} and fixed durations d_{jkm} for the optional task of execution task T_{jk} on machine m. The constraints of the model are

$$\begin{array}{l}
\bigwedge_{m\in M} .\texttt{disjunctive}([S_{jkm} \mid j \in J, k \in [1..n_j]], [d_{jkm} \mid j \in J, k \in [1..n_j]]) \wedge \\
\bigwedge_{j\in J, k\in[1..n_j]} . \mathit{alternative}(S_{jk}, d_{jk}, [S_{jkm} \mid m \in M], [d_{jkm} \mid m \in M]) \wedge \\
\bigwedge_{j\in J, k\in[1..n_j-1]} .S_{jkm} + d_{jkm} \leq S_{jk+1m} \wedge \\
\bigwedge_{j\in J, k\in[1..n_j], m\in M} \mathit{intbot}(S_{jkm})
\end{array}$$

We can add a redundant `cumulative` constraint to improve propagation

$$\begin{array}{l}
\texttt{cumulative}([S_{jk} \mid j \in J, k \in [1..n_j]], [d_{jk} \mid j \in J, k \in [1..n_j]], \\
\qquad\qquad\qquad\qquad\qquad\qquad [1 \mid j \in J, k \in [1..n_j]], |M|) \ .
\end{array}$$

In this model there are no presence implications and $\mathit{impl}(a_i, a_j) = x_j^a$ and similarly for b.

Example 4. Consider a FJSP problem with 2 machines (a, b) and 5 jobs each made up of a single task where the durations (d_a, d_b) of each task if it is executed on machine a,b respectively are given by (12,9), (5,11), (6,7), (9,6), (7,8). We aim to schedule the tasks on the two machines with no two tasks on the same machine overlapping within a makespan of at most 22. This is modelled with 5 (non-optional) tasks with start times S_1, S_2, S_3, S_4, S_5 and (variable) durations $d_1 \in [9..12], d_2 \in 5..11, d_3 \in 6..7, d_4 \in 6..9, d_5 \in 7..8$. And 5 optional tasks with time-intervals a_1, a_2, a_3, a_4, a_5 and fixed durations $da = [12, 5, 6, 9, 7]$ representing that task i runs on machine a. And 5 optional tasks with time-intervals b_1, b_2, b_3, b_4, b_5 with fixed durations $db = [9, 11, 7, 6, 8]$ representing that task i runs on machine b. This constraints of the model are:

$$\begin{array}{lll}
 & \texttt{disjunctive}([a_1, a_2, a_3, a_4, a_5], [12, 5, 6, 9, 6]) \\
\wedge & \texttt{disjunctive}([b_1, b_2, b_3, b_4, b_5], [9, 11, 7, 6, 8]) \\
\wedge & \texttt{cumulative}([S_1, S_2, S_3, S_4, S_5], [d_1, d_2, d_3, d_4, d_5], [1, 1, 1, 1, 1], 2) \\
\wedge & \bigwedge_{i=1}^{5} \mathit{alternative}(S_i, d_i, [a_i, b_i], [da_i, db_i]) \\
\wedge & \bigwedge_{i=1}^{5} \mathit{intbot}(a_i) \quad \wedge \quad \bigwedge_{i=1}^{5} \mathit{intbot}(b_i) \quad \wedge \quad \bigwedge_{i=1}^{5} S_i + d_i \leq 22
\end{array}$$

The first `disjunctive` ensures that no tasks that run on machine a overlap, while the second ensures the same for machine b. The `cumulative` is a redundant constraint that ensures that at most two tasks run at any time. The *alternative* constraints model the relationship between each (non-optional) task the two alternatives running on machines a and b. Finally the *intbot* constraints ensure that the interval variables are accurately modelled by triples.

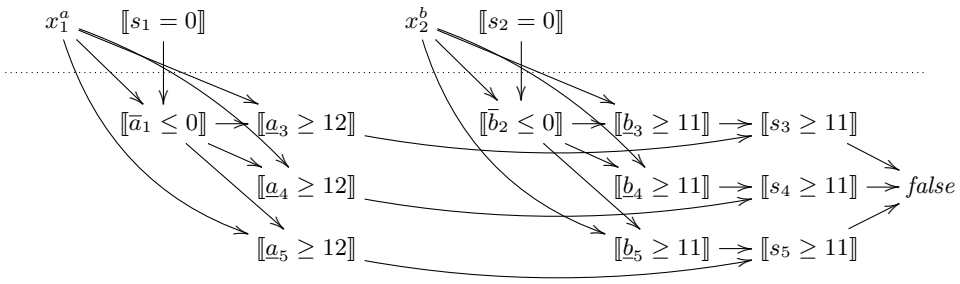


Fig. 1: Implication graph for the search of Example 4. Literals above the dotted lines are decisions.

Suppose search first schedules task 1 on machine a, setting $x_1^a = \mathit{true}$ and $S_1 = 0$. This forces $\underline{a}_1 = \overline{a}_1 = 0$. The first `disjunctive` constraint then imposes that $\underline{a}_2 \geq 12$, $\underline{a}_3 \geq 12$, $\underline{a}_4 \geq 12$, $\underline{a}_5 \geq 12$. Suppose search next schedules task 2 on machine b, setting $x_2^b = \mathit{true}$ and $S_2 = 0$. This forces $\underline{b}_2 = \overline{b}_2 = 0$. The second `disjunctive` constraint then imposes that $\underline{b}_3 \geq 11$, $\underline{b}_3 \geq 11$, $\underline{b}_3 \geq 11$. The *alternative* constraints enforce that $S_3 \geq 11$, $S_4 \geq 11$ and $S_5 \geq 11$. The `cumulative` constraint discovers that task 3 has a compulsory part in $[16, 17)$, task 4 has a compulsory part in $[16, 17)$ and task 5 has a compulsory part in $[15, 18)$. This leads to a resource overload at time 16 and failure is detected. The (relevant part) of the implication graph is shown in Figure 1. The 1UIP nogood is: $[\![\underline{a}_3 \geq 12]\!] \wedge [\![\underline{a}_4 \geq 12]\!] \wedge [\![\underline{a}_5 \geq 12]\!] \wedge x_2^b \wedge [\![S_2 = 0]\!] \rightarrow \mathit{false}$. Note how the interval variables play an important role in propagation and in the final nogood. □

5.2 Experiments

The experiments were run on a X86-64 architecture running GNU/Linux and an Intel(R) Core(TM) i7 CPU processor at 2.8 GHz. The code was written using the G12 Constraint Programming Platform [28]. The model was written in MiniZinc [18] and executed by mzn-g12lazy, the Lcg solver described in [7]. The `disjunctive` propagator in the Lcg solver performs the time-table and edge-finding consistency check before filtering the bounds on the start times via detectable precedences and edge-finding (denoted disjDPEF). We also ran the experiments with filtering via detectable precedences disabled (denoted disjEF). Since there were no significant differences between both cases, we only present disjEF. Appendix A shows the full results for both cases. We compare our results with the current best known lower and upper bounds of the makespan.

We used different benchmark suites for which a brief overview is given in Table 1 where #inst is the number of instances considered, #mach the range of the number of machines, #jobs the range of the number of jobs, #task the

Table 1: Overview of the benchmark suites used.

suite	sub-suite	#inst	#mach	#jobs	#task	#o-task
BC		21	11–18	10–15	100–225	110–270
Bm [4]		10	4–15	10–20	55–240	115–716
Hu [9]	edata	43	5–10	6–20	36–225	42–341
	rdata	43	5–10	6–20	36–225	74–592
	vdata	43	5–10	6–20	36–225	103–1507

range of the number of task per job, and #o-task the range of the total number of optional tasks.

5.3 Upper Bounds Computations

Upper bound computations approach the optimal solution by generating feasible solutions, potentially sub-optimal, and then restricting the objective correspondingly before continuing the search. Many methods (see, *e.g.*, [15,20,33]) have been proposed for finding feasible solutions. Most of them are incomplete, *i.e.*, they have no guarantee for finding the optimal solution and proving its optimality, but they are fast.

We use branch and bound for minimising the makespan and an activity-based search (an adaption of Vsids [17]) with restart. A geometric restart policy [32] on the number of node failures was used with a factor of 2.0 and a base of 256. The upper bound on the makespan was initialised to the rounded up value of the average makespan computed by [15], because [15] provides a method that quickly finds high quality solutions.

Tables 2–4 are organised as follow: the column Inst provides the instance names; the column LB-UB the best known lower and upper bound with respect to [10,15,20,3,33]; the column Initial presents the rounded up average UB obtained by [15] over several runs and its average run time in seconds;[3] the column disjEF shows the best obtained UB and the run time in seconds in which a bold UB indicates that disjEF could improve the best known bound or closed the instance, and an asterisk after UB indicates that the disjEF was able to find the optimal solution and prove it. An entry *n/a* in UB indicates that the Lcg solver was not able to find a solution with the given initial UB within the run time limit of 10 minutes.

Our method performed exceptionally well on instances from BC (see Table 2), all instances could be solved within the time limit given. In contrast, the instances from Bm were harder to solve for our methods (see Table 3), although the instance Mk04 was closed. Only five instances could be solved and a solution could be found for only one of the remaining five instances.

Table 4 show the result on a subset of instances from Hu. Due to space limits, we omit the instances mt06, mt10, mt20, and la01–la20 from the sub-

[3] The numbers were taken from the appendix of [15] provided at `http://www.idsia.ch/~monaldo/fjsp.html`

Table 2: Results on BC with initial UB from [15].

Inst	#o-task	LB-UB	Initial Sol		disjEF	
			UB	time	UB	time
mt10c1	110	655-927	928	2.33s	**927***	4.47s
mt10cc	120	655-908	910	10.04s	**908***	3.66s
mt10x	110	655-918	918	4.31s	**918***	2.45s
mt10xx	120	655-918	918	1.73s	**918***	2.21s
mt10xxx	130	655-918	918	1.10s	**918***	2.87s
mt10xy	120	655-905	906	4.02s	**905***	4.41s
mt10xyz	130	655-847	851	5.50s	**847***	2.98s
setb4c9	165	857-914	920	14.02s	**914***	12.45s
setb4cc	180	857-907	912	12.95s	**907***	8.60s
setb4x	165	846-925	925	7.45s	**925***	12.86s
setb4xx	180	846-925	927	14.87s	**925***	14.31s
setb4xxx	195	846-925	925	7.99s	**925***	15.02s
setb4xy	180	845-910	916	3.15s	**910***	8.40s
setb4xyz	195	838-903	909	7.35s	**902***	6.77s
seti5c12	240	1027-1171	1175	19.49s	**1169***	54.68s
seti5cc	255	955-1136	1137	11.91s	**1135***	95.27s
seti5x	240	955-1198	1204	15.85s	**1198***	28.34s
seti5xx	255	955-1197	1201	23.64s	**1194***	12.43s
seti5xxx	270	955-1197	1199	23.51s	**1194***	8.84s
seti5xy	255	955-1136	1137	11.91s	**1135***	95.20s
seti5xyz	270	955-1125	1127	17.13s	**1125***	337.98s

Table 3: Results on Bm with initial UB from [15].

Inst	#o-task	LB-UB	Initial Sol		disjEF	
			UB	time	UB	time
Mk01	115	40	40	0.01s	40*	0.25s
Mk02	238	24-26	26	0.73s	n/a	598.72s
Mk03	451	204	204	0.01s	204*	2.10s
Mk04	172	48-60	60	0.08s	**60***	0.45s
Mk05	181	168-172	173	0.96s	173	598.40s
Mk06	490	33-57	59	3.26s	59	599.28s
Mk07	283	133-139	147	8.91s	n/a	598.10s
Mk08	322	523	523	0.02s	523*	4.95s
Mk09	606	307	307	0.15s	307*	9.69s
Mk10	716	165-196	200	7.69s	n/a	599.96s

suites edata and rdata since they are easily solvable. We also omit the entire sub-suite vdata, because no new results could be obtained. The full results are presented in Appendix A. The Lcg solver disjEF solves all instances of the sub-suite edata except 7 and closes 9 of them. For the sub-suite rdata, the Lcg solver closes 5 instances, but for 13 instances it could not find a solution within the time limit.

Table 4: Results on Hu with initial UB from [15].

Inst	#o-task	LB-UB	Initial Sol		disjEF	
			UB	time	UB	time
edata/la11	113	1087-1103	1103	1.91s	**1103***	0.45s
edata/la21	173	895-1009	1024	2.83s	1013	598.97s
edata/la22	173	832-880	883	4.29s	**880***	6.21s
edata/la23	171	950	950	2.97s	950*	17.26s
edata/la24	174	881-908	912	3.88s	**908***	80.21s
edata/la25	174	894-936	945	1.76s	**936***	21.89s
edata/la26	227	1089-1107	1127	5.48s	1127	598.72s
edata/la27	227	1181	1189	9.25s	1189	598.86s
edata/la28	226	1116-1142	1149	3.44s	1144	599.30s
edata/la29	227	1058-1111	1121	5.47s	1121	598.84s
edata/la30	227	1147-1195	1214	9.22s	1208	599.52s
edata/la31	341	1523-1533	1541	9.58s	1538	600.11s
edata/la32	341	1698	1698	1.85s	1698*	101.44s
edata/la33	339	1547	1547	1.40s	1547*	27.53s
edata/la34	339	1592-1599	1600	9.35s	**1599***	52.03s
edata/la35	339	1736	1736	0.41s	1736*	3.37s
edata/la36	258	1006-1160	1164	8.08s	**1160***	26.01s
edata/la37	258	1397	1397	3.48s	1397*	1.59s
edata/la38	257	1019-1143	1147	6.90s	**1141***	436.15s
edata/la39	257	1151-1184	1186	8.68s	**1184***	13.12s
edata/la40	258	1034-1144	1152	7.78s	**1144***	473.22s
rdata/la02	94	529-530	531	1.31s	**529***	431.20s
rdata/la19	196	647-700	702	1.90s	**700***	1.35s
rdata/la20	199	756	841	7.81s	n/a	598.79s
rdata/la22	306	737-758	764	5.14s	764	598.97s
rdata/la23	306	816-832	846	6.50s	n/a	598.88s
rdata/la24	297	775-801	814	4.06s	n/a	598.82s
rdata/la25	302	752-785	795	3.38s	793	599.34s
rdata/la26	391	1056-1061	1064	7.69s	n/a	599.00s
rdata/la27	392	1085-1090	1093	7.47s	n/a	598.96s
rdata/la28	402	1075-1080	1082	7.54s	n/a	599.00s
rdata/la29	399	993-997	999	4.03s	n/a	599.15s
rdata/la30	392	1068-1078	1081	7.78s	n/a	599.04s
rdata/la31	576	1520-1521	1522	8.61s	n/a	599.37s
rdata/la32	585	1657-1659	1660	12.67s	n/a	599.36s
rdata/la33	581	1497-1498	1500	11.48s	n/a	599.28s
rdata/la34	584	1535-1536	1537	7.28s	n/a	599.41s
rdata/la35	592	1549-1550	1551	15.28s	n/a	599.33s
rdata/la36	439	1016-1028	1032	4.90s	**1023***	38.98s
rdata/la37	437	989-1066	1081	9.52s	1077	600.50s
rdata/la38	444	943-960	968	9.32s	**954***	44.19s
rdata/la39	436	966-1018	1034	2.78s	**1011***	539.80s
rdata/la40	441	955-956	974	6.11s	968	601.03s

Overall we close 36 open instances and improve the best known upper bounds of 11 instances. Our approach is strongest on examples without too many tasks, we plan to investigate the combination with large neighbourhood search (as in [20]) to improve results on larger problems.

6 Conclusion and Outlook

Scheduling with optional tasks generalises the case of scheduling with tasks that must always execute. It provides considerable expressiveness for defining complex scheduling problems. In this paper we show how to extend LCG solvers to support scheduling with optional tasks. The resulting system combines the advantages of scheduling with optional tasks with learning. We demonstrate the power of the combination on hard flexible job-shop scheduling problems. In the future we plan to extend our implementation for optional tasks to more propagators, and probably to implement a native $\texttt{int}_\perp$ variable in our LCG solvers (although we do not expect the native implementation to be much more efficient than the tripartite model we use here).

Acknowledgements NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program. This work was partially supported by Asian Office of Aerospace Research and Development grant 10-4123.

References

1. Barták, R., Čepek, O.: Temporal networks with alternatives: Complexity and model. In: Proceedings of the Twentieth International Florida AI Research Society Conference (FLAIRS). pp. 641–646 (2007)
2. Beck, J.C., Fox, M.S.: Scheduling alternative activities. In: Proceedings of the National Conference on Artificial Intelligence. pp. 680–687. John Wiley & Sons, Ltd. (1999)
3. Behnke, D., Geiger, M.J.: Test instances for the flexible job shop scheduling problem with work centers. Research Report RR-12-01-01, Helmut-Schmidt-Universität, Hamburg, Germany (2012)
4. Brandimarte, P.: Routing and scheduling in a flexible job shop by tabu search. Annals of Operations Research 41(3), 157–183 (1993)
5. Brucker, P., Schlie, R.: Job-shop scheduling with multi-purpose machines. Computing 45(4), 369–375 (1990)
6. Feydy, T., Somogyi, Z., Stuckey, P.: Half-reification and flattening. In: Lee, J. (ed.) Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming. LNCS, vol. 6876, pp. 286–301. Springer (2011)
7. Feydy, T., Stuckey, P.J.: Lazy clause generation reengineered. In: Gent [8], pp. 352–366
8. Gent, I.P. (ed.): Proceedings of Principles and Practice of Constraint Programming – CP 2009, Lecture Notes in Computer Science, vol. 5732. Springer Berlin / Heidelberg (2009)

9. Hurink, J., Jurisch, B., Thole, M.: Tabu search for the job-shop scheduling problem with multi-purpose machines. Operations-Research-Spektrum 15(4), 205–215 (1994)
10. Jurisch, B.: Scheduling jobs in shops with multi-purpose machines. Ph.D. thesis, Universität Osnabrück (1992)
11. Laborie, P.: IBM ILOG CP Optimizer for detailed scheduling illustrated on three problems. In: Hoeve, W.J., Hooker, J.N. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Lecture Notes in Computer Science, vol. 5547, pp. 148–162. Springer Berlin Heidelberg (2009)
12. Laborie, P., Rogerie, J.: Reasoning with conditional time-intervals. In: Wilson, D.C., Lane, H.C. (eds.) Proceedings of the Twenty-First International Florida Artificial Intelligence Research Society Conference. pp. 555–560. AAAI Press (2008)
13. Laborie, P., Rogerie, J., Shaw, P., Vilím, P.: Reasoning with conditional time-intervals part II: An algebraical model for resources. In: Lane, H.C., Guesgen, H.W. (eds.) Proceedings of the Twenty-First International Florida Artificial Intelligence Research Society Conference. pp. 201–206. AAAI Press (2009)
14. Laborie, P., Rogerie, J., Shaw, P., Vilím, P., Katai, F.: Interval-based language for modeling scheduling problems: An extension to constraint programming. In: Kallrath, J. (ed.) Algebraic Modeling Systems, Applied Optimization, vol. 104, pp. 111–143. Springer Berlin Heidelberg (2012)
15. Mastrolilli, M., Gambardella, L.M.: Effective neighbourhood functions for the flexible job shop problem. Journal of Scheduling 3(1), 3–20 (2000)
16. Moffitt, M.D., Peintner, B., Pollack, M.E.: Augmenting disjunctive temporal problems with finite-domain constraints. In: Proceedings of the National Conference on Artificial Intelligence. pp. 1187–1192. AAAI Press (2005)
17. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proceedings of Design Automation Conference – DAC 2001. pp. 530–535. ACM, New York, NY, USA (2001)
18. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: MiniZinc: Towards a standard CP modelling language. In: Bessière, C. (ed.) Principles and Practice of Constraint Programming CP 2007. Lecture Notes in Computer Science, vol. 4741, pp. 529–543. Springer Berlin Heidelberg (2007)
19. Ohrimenko, O., Stuckey, P.J., Codish, M.: Propagation via lazy clause generation. Constraints 14(3), 357–391 (2009)
20. Pacino, D., Van Hentenryck, P.: Large neighborhood search and adaptive randomized decompositions for flexible jobshop scheduling. In: Proceedings of the Twenty-Second international joint conference on Artificial Intelligence. pp. 1997–2002. IJCAI'11, AAAI Press (2011)
21. Schulte, C., Stuckey, P.J.: Efficient constraint propagation engines. ACM Transactions on Programming Languages and Systems 31(1), Article No. 2 (2008)
22. Schutt, A., Chu, G., Stuckey, P.J., Wallace, M.G.: Maximising the net present value for resource-constrained project scheduling. In: Beldiceanu, N., Jussien, N., Pinson, E. (eds.) Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems, Lecture Notes in Computer Science, vol. 7298, pp. 362–378. Springer Berlin Heidelberg (2012)
23. Schutt, A., Feydy, T., Stuckey, P.J.: Explaining time-table-edge-finding propagation for the cumulative resource constraint. In: Gomes, C.P., Sellmann, M. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Lecture Notes in Computer Science, vol. 7874, pp. 234–250. Springer Berlin Heidelberg (2013)

24. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Explaining the cumulative propagator. Constraints 16(3), 250–282 (2011)
25. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Solving RCPSP/max by lazy clause generation. Journal of Scheduling pp. 1–17 (2012), online first
26. Seipel, D., Hanus, M., Wolf, A. (eds.): Efficient Edge-Finding on Unary Resources with Optional Activities, Lecture Notes in Computer Science, vol. 5437. Springer Berlin Heidelberg (2009)
27. Somogyi, Z., Henderson, F., Conway, T.: The execution algorithm of Mercury, an efficient purely declarative logic programming language. The Journal of Logic Programming 29(1–3), 17–64 (1996)
28. Stuckey, P.J., García de la Banda, M.J., Maher, M.J., Marriott, K., Slaney, J.K., Somogyi, Z., Wallace, M.G., Walsh, T.: The G12 project: Mapping solver independent models to efficient solutions. In: Gabbrielli, M., Gupta, G. (eds.) Proceedings of Logic Programming – ICLP 2005. Lecture Notes in Computer Science, vol. 3668, pp. 9–13. Springer Berlin / Heidelberg (Oct 2005)
29. Vilím, P.: Edge finding filtering algorithm for discrete cumulative resources in $\mathcal{O}(kn \log n)$. In: Gent [8], pp. 802–816
30. Vilím, P.: Max energy filtering algorithm for discrete cumulative resources. In: Hoeve, W.J., Hooker, J.N. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Lecture Notes in Computer Science, vol. 5547, pp. 294–308. Springer Berlin Heidelberg (2009)
31. Vilím, P., Barták, R., Čepek, O.: Extension of $O(n \log n)$ filtering algorithms for the unary resource constraint to optional activities. Contraints 10(4), 403–425 (October 2005)
32. Walsh, T.: Search in a small world. In: Proceedings of Artificial intelligence – IJCAI 1999. pp. 1172–1177. Morgan Kaufmann (1999)
33. Yuan, Y., Xu, H.: Flexible job shop scheduling using hybrid differential evolution algorithms. Computers & Industrial Engineering 65(2), 246–260 (2013)

A Detailed Upper Bound Results

Tables 5–9 show all results of the experiments conducted.

Table 5: Results on BC with initial UB and time taken from [15].

Inst	#o-task	LB-UB	Initial Sol		disjEF		disjDPEF	
			UB	time	UB	time	UB	time
mt10c1	110	655-927	928	2.33s	**927***	4.47s	**927***	5.84s
mt10cc	120	655-908	910	10.04s	**908***	3.66s	**908***	3.52s
mt10x	110	655-918	918	4.31s	**918***	2.45s	**918***	2.02s
mt10xx	120	655-918	918	1.73s	**918***	2.21s	**918***	2.38s
mt10xxx	130	655-918	918	1.10s	**918***	2.87s	**918***	3.27s
mt10xy	120	655-905	906	4.02s	**905***	4.41s	**905***	5.93s
mt10xyz	130	655-847	851	5.50s	**847***	2.98s	**847***	3.11s
setb4c9	165	857-914	920	14.02s	**914***	12.45s	**914***	13.49s
setb4cc	180	857-907	912	12.95s	**907***	8.60s	**907***	8.23s
setb4x	165	846-925	925	7.45s	**925***	12.86s	**925***	20.70s
setb4xx	180	846-925	927	14.87s	**925***	14.31s	**925***	18.37s
setb4xxx	195	846-925	925	7.99s	**925***	15.02s	**925***	17.55s
setb4xy	180	845-910	916	3.15s	**910***	8.40s	**910***	13.23s
setb4xyz	195	838-903	909	7.35s	**902***	6.77s	**902***	10.01s
seti5c12	240	1027-1171	1175	19.49s	**1169***	54.68s	**1169***	55.26s
seti5cc	255	955-1136	1137	11.91s	**1135***	95.27s	**1135***	202.41s
seti5x	240	955-1198	1204	15.85s	**1198***	28.34s	**1198***	32.85s
seti5xx	255	955-1197	1201	23.64s	**1194***	12.43s	**1194***	13.06s
seti5xxx	270	955-1197	1199	23.51s	**1194***	8.84s	**1194***	21.62s
seti5xy	255	955-1136	1137	11.91s	**1135***	95.20s	**1135***	202.23s
seti5xyz	270	955-1125	1127	17.13s	**1125***	337.98s	**1125***	566.15s

Table 6: Results on Bm with initial UB and time taken from [15].

Inst	#o-task	LB-UB	Initial Sol		disjEF		disjDPEF	
			UB	time	UB	time	UB	time
Mk01	115	40	40	0.01s	40*	0.25s	40*	0.25s
Mk02	238	24-26	26	0.73s	n/a	598.72s	n/a	598.64s
Mk03	451	204	204	0.01s	204*	2.10s	204*	2.48s
Mk04	172	48-60	60	0.08s	**60***	0.45s	**60***	0.44s
Mk05	181	168-172	173	0.96s	173	598.40s	173	598.36s
Mk06	490	33-57	59	3.26s	59	599.28s	n/a	599.09s
Mk07	283	133-139	147	8.91s	n/a	598.10s	n/a	598.54s
Mk08	322	523	523	0.02s	523*	4.95s	523*	3.59s
Mk09	606	307	307	0.15s	307*	9.69s	307*	30.45s
Mk10	716	165-196	200	7.69s	n/a	599.96s	n/a	600.06s

Table 7: Results on Hu **edata** with initial UB and time taken from [15].

Inst	#o-task	LB-UB	Initial Sol		disjEF		disjDPEF	
			UB	time	UB	time	UB	time
la01	60	609	609	0.01s	609*	0.19s	609*	0.16s
la02	60	655	655	0.04s	655*	0.28s	655*	0.30s
la03	59	550	550	1.00s	550*	0.17s	550*	0.15s
la04	59	568	568	0.36s	568*	0.18s	568*	0.17s
la05	60	503	503	0.01s	503*	0.15s	503*	0.13s
la06	86	833	833	0.00s	833*	0.20s	833*	0.27s
la07	86	762	762	0.35s	762*	0.28s	762*	0.26s
la08	86	845	845	0.02s	845*	0.27s	845*	0.45s
la09	86	878	878	0.04s	878*	0.18s	878*	0.26s
la10	87	866	866	0.01s	866*	0.32s	866*	0.26s
la11	113	1087-1103	1103	1.91s	**1103***	0.45s	**1103***	0.56s
la12	114	960	960	0.02s	960*	0.46s	960*	0.50s
la13	114	1053	1053	0.02s	1053*	0.78s	1053*	0.58s
la14	113	1123	1123	0.03s	1123*	0.44s	1123*	0.59s
la15	113	1111	1111	0.30s	1111*	0.50s	1111*	0.68s
la16	113	892	892	0.25s	892*	0.42s	892*	0.51s
la17	113	707	707	0.58s	707*	0.26s	707*	0.32s
la18	113	842	842	0.79s	842*	0.73s	842*	0.71s
la19	113	796	796	1.53s	796*	1.08s	796*	1.33s
la20	113	857	857	0.88s	857*	0.41s	857*	0.41s
la21	173	895-1009	1024	2.83s	1013	598.97s	1014	599.43s
la22	173	832-880	883	4.29s	**880***	6.21s	**880***	6.02s
la23	171	950	950	2.97s	950*	17.26s	950*	10.44s
la24	174	881-908	912	3.88s	**908***	80.21s	**908***	66.11s
la25	174	894-936	945	1.76s	**936***	21.89s	**936***	21.57s
la26	227	1089-1107	1127	5.48s	1127	598.72s	1125	599.11s
la27	227	1181	1189	9.25s	1189	598.86s	1189	598.85s
la28	226	1116-1142	1149	3.44s	1144	599.30s	1147	599.13s
la29	227	1058-1111	1121	5.47s	1121	598.84s	1116	599.53s
la30	227	1147-1195	1214	9.22s	1208	599.52s	n/a	598.80s
la31	341	1523-1533	1541	9.58s	1538	600.11s	n/a	598.97s
la32	341	1698	1698	1.85s	1698*	101.44s	1698*	52.09s
la33	339	1547	1547	1.40s	1547*	27.53s	1547*	45.41s
la34	339	1592-1599	1600	9.35s	**1599***	52.03s	**1599***	148.32s
la35	339	1736	1736	0.41s	1736*	3.37s	1736*	11.90s
la36	258	1006-1160	1164	8.08s	**1160***	26.01s	**1160***	67.89s
la37	258	1397	1397	3.48s	1397*	1.59s	1397*	1.45s
la38	257	1019-1143	1147	6.90s	**1141***	436.15s	**1141***	365.34s
la39	257	1151-1184	1186	8.68s	**1184***	13.12s	**1184***	14.44s
la40	258	1034-1144	1152	7.78s	**1144***	473.22s	**1144***	510.35s
mt06	42	55	55	0.00s	55*	0.09s	55*	0.07s
mt10	113	871	873	1.61s	871*	2.13s	871*	2.21s
mt20	113	1088	1089	3.52s	1088*	0.91s	1088*	0.56s

Table 8: Results on Hu **rdata** with initial UB and time taken from [15].

Inst	#o-task	LB-UB	Initial Sol		disjEF		disjDPEF	
			UB	time	UB	time	UB	time
la01	96	570	572	1.97s	570	598.55s	571	598.60s
la02	94	529-530	531	1.31s	**529***	431.20s	**529***	472.77s
la03	99	477	479	1.36s	477	598.64s	477	598.57s
la04	101	502	503	0.62s	502	598.61s	502	598.70s
la05	103	457	458	1.78s	457	598.63s	457	598.64s
la06	141	799	800	2.99s	799	598.65s	799	598.70s
la07	147	749	750	1.13s	749	598.68s	749	598.77s
la08	145	765	766	0.35s	765	598.64s	765	598.68s
la09	144	853	854	2.29s	853	598.55s	853	598.68s
la10	147	804	805	1.32s	804	598.54s	804	598.61s
la11	203	1071	1071	2.56s	1071	598.65s	1071	598.78s
la12	199	936	936	0.08s	936	598.42s	936	598.64s
la13	198	1038	1038	0.90s	1038	598.13s	1038	598.68s
la14	197	1070	1070	0.28s	1070	598.73s	1070	598.59s
la15	198	1089	1090	1.76s	1089	598.70s	1089	598.82s
la16	201	717	717	0.07s	717*	0.71s	717*	0.67s
la17	193	646	646	0.03s	646*	0.86s	646*	0.87s
la18	199	666	666	1.79s	666*	0.58s	666*	0.61s
la19	196	647-700	702	1.90s	**700***	1.35s	**700***	1.10s
la20	199	756	756	0.03s	756*	0.65s	756*	0.66s
la21	301	808-833	841	7.81s	n/a	598.79s	n/a	599.01s
la22	306	737-758	764	5.14s	764	598.97s	n/a	598.93s
la23	306	816-832	846	6.50s	n/a	598.88s	n/a	598.99s
la24	297	775-801	814	4.06s	n/a	598.82s	814	599.05s
la25	302	752-785	795	3.38s	793	599.34s	n/a	598.96s
la26	391	1056-1061	1064	7.69s	n/a	599.00s	n/a	599.05s
la27	392	1085-1090	1093	7.47s	n/a	598.96s	n/a	598.95s
la28	402	1075-1080	1082	7.54s	n/a	599.00s	n/a	599.02s
la29	399	993-997	999	4.03s	n/a	599.15s	n/a	599.12s
la30	392	1068-1078	1081	7.78s	n/a	599.04s	n/a	599.02s
la31	576	1520-1521	1522	8.61s	n/a	599.37s	n/a	599.40s
la32	585	1657-1659	1660	12.67s	n/a	599.36s	n/a	599.53s
la33	581	1497-1498	1500	11.48s	n/a	599.28s	n/a	599.50s
la34	584	1535-1536	1537	7.28s	n/a	599.41s	n/a	599.48s
la35	592	1549-1550	1551	15.28s	n/a	599.33s	n/a	599.53s
la36	439	1016-1028	1032	4.90s	**1023***	38.98s	**1023***	65.48s
la37	437	989-1066	1081	9.52s	1077	600.50s	1073	601.50s
la38	444	943-960	968	9.32s	**954***	44.19s	**954***	37.67s
la39	436	966-1018	1034	2.78s	**1011***	539.80s	**1011***	575.64s
la40	441	955-956	974	6.11s	968	601.03s	964	602.38s
mt06	74	47	47	0.00s	47*	0.13s	47*	0.13s
mt10	196	686	686	2.71s	686*	0.91s	686*	0.86s
mt20	194	1022	1023	3.53s	1022	598.74s	1022	598.75s

Table 9: Results on Hu **vdata** with initial UB and time taken from [15].

Inst	#o-task	LB-UB	Initial Sol		disjEF		disjDPEF	
		UB	time	UB	time	UB	time	
la01	142	570	571	1.08s	570	598.59s	570	598.53s
la02	134	529	530	2.29s	529	598.74s	529	598.58s
la03	128	477	478	0.58s	477	598.63s	477	598.64s
la04	119	502	502	1.15s	502	598.54s	502	598.57s
la05	119	457	458	1.40s	457	598.55s	457	598.59s
la06	182	799	799	0.97s	799	598.71s	799	598.60s
la07	182	749	750	3.55s	750	598.67s	749	598.77s
la08	194	765	766	1.81s	765	598.65s	765	598.85s
la09	190	853	853	2.59s	853	598.28s	853	598.28s
la10	200	804	804	1.66s	804	598.53s	804	598.74s
la11	253	1071	1071	0.68s	1071	598.62s	1071	598.79s
la12	248	936	936	0.59s	936	598.72s	n/a	598.79s
la13	245	1038	1038	0.65s	1038	598.61s	1038	598.67s
la14	244	1070	1070	0.53s	1070	598.32s	1070	598.30s
la15	263	1089	1090	1.55s	1090	598.78s	1089	598.74s
la16	470	717	717	0.01s	717*	1.61s	717*	1.59s
la17	479	646	646	0.01s	646*	1.65s	646*	1.74s
la18	479	663	663	0.01s	663*	1.69s	663*	1.74s
la19	480	617	617	0.12s	617*	1.63s	617*	1.60s
la20	487	756	756	0.01s	756*	1.90s	756*	1.97s
la21	715	800-804	808	4.66s	n/a	599.57s	n/a	599.45s
la22	677	733-735	740	6.42s	n/a	599.42s	n/a	599.36s
la23	680	809-813	816	5.66s	n/a	599.41s	n/a	599.45s
la24	727	773-775	779	6.79s	n/a	599.44s	n/a	599.60s
la25	725	751-756	757	6.27s	n/a	599.46s	n/a	599.47s
la26	917	1052-1053	1055	8.55s	n/a	599.75s	n/a	599.71s
la27	915	1084	1086	6.44s	n/a	599.75s	n/a	599.76s
la28	897	1069	1071	12.73s	n/a	599.83s	n/a	599.90s
la29	881	993-994	995	13.06s	n/a	599.77s	n/a	599.87s
la30	930	1068-1069	1070	9.60s	n/a	599.80s	n/a	599.86s
la31	1380	1520	1520	16.92s	n/a	600.89s	n/a	600.67s
la32	1362	1657-1658	1658	13.00s	n/a	600.70s	n/a	600.68s
la33	1354	1497	1498	17.14s	n/a	600.23s	n/a	600.75s
la34	1387	1535	1536	13.50s	n/a	600.72s	n/a	600.96s
la35	1394	1549	1549	20.69s	n/a	600.81s	n/a	601.05s
la36	1507	948	948	0.28s	948*	55.62s	948*	25.41s
la37	1492	986	986	0.44s	986*	55.52s	986*	198.16s
la38	1479	943	943	0.09s	943*	49.89s	943*	27.80s
la39	1470	922	922	0.22s	922*	12.57s	922*	29.49s
la40	1458	955	955	0.26s	955*	26.07s	955*	24.82s
mt06	103	47	47	0.00s	47*	0.17s	47*	0.18s
mt10	448	655	655	0.02s	655*	1.36s	655*	1.40s
mt20	262	1022	1022	3.79s	1022	598.71s	n/a	598.81s