MISTA 2013

Temporal Linear Relaxation in IBM ILOG CP Optimizer

Philippe Laborie · Jérôme Rogerie

Abstract IBM ILOG CP Optimizer is a constraint solver that implements a model-and-run paradigm. For scheduling problems CP Optimizer provides a relatively simple but very expressive modeling language based on the notion of interval variables. This paper presents the Temporal Linear Relaxation (TLR) used to guide the automatic search when solving scheduling problems that involve temporal and resource allocation costs. We give the rationale of the TLR, describe its integration in the automatic search of CP Optimizer and present the relaxation of most of the constraints and expressions of the model. An experimental study on a set of classical scheduling benchmarks shows that using the TLR is essential for problems with irregular temporal costs and generally helps for problems with resource allocation costs.

1 Introduction

IBM ILOG CP Optimizer is a general constraint solver that implements a model-and-run paradigm. This paper focuses on scheduling problems for which CP Optimizer provides a relatively simple but very expressive modeling language based on the notion of interval variables [12,13]. Real world scheduling applications often require optimizing irregular temporal costs (such as minimizing earliness/tardiness, minimizing or maximizing activities duration or delays between activities, *etc*) as well as non-temporal costs (resource allocation, non execution, *etc*). For these problems, even in the absence of resource constraints, providing a good time placement of activities can be a challenge. In this context, using a linear relaxation of the problem solved with Mathematical Programming techniques is a common approach. MP and CP are generally coupled in two ways. In *decomposition* approaches an MP model communicates with a distinct CP model (Sequential models, Column Generation, Logical Bender's Decomposition). Although very efficient on particular scheduling problems, these approaches

Philippe Laborie
IBM, Software Group
E-mail: laborie@fr.ibm.com

Jérôme Rogerie
IBM, Software Group
E-mail: rogerie@fr.ibm.com

tend to be difficult to generalize. In *hybrid* approaches like [3,8], the linearization is computed at each search node. For those approaches, scaling to large models may be an issue. This paper presents the approach implemented in CP Optimizer that lies somewhere in between and aims at providing both generality and scalability.

After recapping the CP Optimizer model for scheduling in section 2, we give an overview of the architecture of the automatic search for scheduling problems in section 3. The main purpose of this section is to show how the proposed Temporal Linear Relaxation (TLR) is integrated in the search. Section 4 describes the linearization of most of the scheduling concepts of the model in the TLR. The final section presents an experimental evaluation that shows the crucial positive impact of the TLR on the performance of the automatic search.

2 CP Optimizer Model for Scheduling

CP Optimizer extends classical constraint programming by introducing additional mathematical concepts (such as intervals [12], sequences or functions [13]) as new variables or expressions to capture the temporal aspects of scheduling.

An *interval variable* a is a decision variable whose domain is a subset of $\{\perp\} \cup \{[s,e) : s,e \in Z \wedge s \leq e\}$. A boolean $x(a)$ denotes the presence status of the interval variable. At a solution:

- if a is present, the value of a is $a = [s,e)$ and the presence status $x(a)$ is true. In this case, by a small abuse of notations $s(a)$, $e(a)$ and $l(a)$ respectively denote the start value s, the end value e and the length $(e-s)$ of interval variable a.
- else, a is absent, $a = \perp$ and the presence status $x(a)$ is false.

Absent interval variables have special meaning. Informally speaking, an absent interval variable is not considered by any constraint or expression on interval variables it is involved in. For example, if an absent interval variable a is used in a precedence constraint between interval variables a and b then, the constraint becomes inactive - in particular, interval variable b is not constrained by it. Each constraint and expression specifies how it handles absent interval variables.

An *optional* interval variable is an interval variable whose presence/absence status needs to be decided by the engine that is, such that singleton $\{\perp\}$ is strictly included in the domain. The use of optional interval variable is quite versatile. A typical use is for expressing an optional activity in a schedule. Another common use is for modeling a task requiring one machine among an alternative machine set. For each machine candidate an optional interval variable models the possible execution of the task on the machine. Then the solver will decide which one among those optional interval variables is present (that is, which machine is selected), the others being absent. By nature interval variables are optional and they can be made non-optional by adding constraints which remove $\perp$ from the domain.

The following constraints on interval variables are introduced to model the basic structure of scheduling problems. Let a, a_i and b denote interval variables and z an integer variable:

- A *presence constraint* presenceOf(a) states that interval a is present, that is $a \neq \perp$. This constraint can be composed, for instance presenceOf(a) $\Rightarrow$ presenceOf(b) means that the presence of a implies the presence of b.

– A *precedence constraint* (*e.g.* endBeforeStart(a, b, z)) specifies a precedence between interval end-points with an integer or variable minimal distance z provided both intervals a and b are present.
– A *span constraint* span $(a, \{b_1, ..., b_n\})$ states that if a is present, it starts together with the first present interval in $\{b_1, ..., b_n\}$ and ends together with the last one. Interval a is absent if and only if all the b_i are absent.
– An *alternative constraint* alternative $(a, \{b_1, ..., b_n\})$ models an exclusive alternative between $\{b_1, ..., b_n\}$: if interval a is present then exactly one of intervals $\{b_1, ..., b_n\}$ is present and a starts and ends together with this chosen one. Interval a is absent if and only if all the b_i are absent.

These constraints make it easy to capture the hierarchical, logical and temporal structure of scheduling problems (the breakdown structure of a project, optional activities, alternative modes/recipes/processes, etc.) in a well-defined CP paradigm [12].

CP Optimizer also provides numerical expressions related with the integer bounds (start, end, length) of an interval variable. These expressions are shown on Table 1. The length of an interval variable is the distance between its start and end points. In these expressions, F is a piecewise linear function. For instance $endEval(a, F, V)$ could be used to model an earliness/tardiness cost function F for an optional activity modeled by an interval variable a evaluated on its end time while value V represents a non-execution cost for the activity. A wide range of arithmetic expressions and constraints are available in CP Optimizer to combine these numerical expressions.

startOf(a, V)	$s(a)$ if $x(a)$ is true V otherwise
endOf(a, V)	$e(a)$ if $x(a)$ is true V otherwise
lengthOf(a, V)	$l(a)$ if $x(a)$ is true V otherwise
startEval(a, F, V)	$F(s(a))$ if $x(a)$ is true V otherwise
endEval(a, F, V)	$F(e(a))$ if $x(a)$ is true V otherwise
lengthEval(a, F, V)	$F(l(a))$ if $x(a)$ is true V otherwise

Table 1: Expression semantics

For modeling resources, and more generally any aspect of the problem involving a timeline, CP Optimizer offers concepts such as *cumul functions*, *state functions* and *interval sequences* [13].

We illustrate how timelines are relaxed in the TLR in section 4 for the particular case of *cumul functions*. A cumul function is an expression whose value in a solution is an integer stepwise function. A cumul function is defined as the sum of individual contributions of interval variables. For instance, an interval variable a_i could contribute to a cumul function $f = \sum_i pulse(a_i, h_i)$ with a pulse function $pulse(a_i, h_i)$. When interval variable a_i is present, a pulse function $pulse(a_i, h_i)$ is a step function equal to value h_i between the start and end of interval a_i and equal to value 0 outside the interval. Otherwise, if the interval variable is absent, it is the 0 function. Constraints can be posted on cumul functions to limit their allowed minimal or maximal value

over the complete horizon ($f \leq K$) or over some constant or variable intervals. Cumul functions are typically used to model cumulative resources.

The combination of constraints on the above timeline expressions together with structural constraints on interval variables like *span* and *alternative* facilitate the modeling of complex resources.

3 Integration of TLR in CP Optimizer Automatic Search.

Large Neighborhood Search (LNS) [20] is a component of CP Optimizer automatic search for scheduling. It consists of a process of continual relaxation and re-optimization: a first solution is computed and iteratively improved. Each iteration consists of a relaxation step followed by a re-optimization of the relaxed solution. This process continues until some condition is satisfied, typically, when the solution can be proved to be optimal or when a time limit is reached. In CP Optimizer this approach is robustified by using portfolios of large neighborhoods and completion strategies in combination with Machine Learning techniques to converge on the most efficient neighborhoods and completion strategies for the problem being solved.

In this context, the large neighborhoods are all based on the initial generation of a Partial Order Schedule (POS) [17] constructed from a completely instantiated solution where interval variables have fixed start and end values. A POS is a directed graph G($\mathcal{A}$, $\mathcal{E}$) where the edges in $\mathcal{E}$ are precedence constraints between interval variables with the property that any solution to the graph is also a resource-feasible solution that is, it satisfies the constraints posted on *interval sequence variables*, *cumul expressions* and *state function variables*. Algorithms for transforming a fully instantiated solution into a POS P are described in [10,12]. Large neighborhoods in the portfolio differ in the way they select a subset of interval variables to be relaxed (fragment). The relaxed POS P' is obtained by removing from P all the edges involving at least one selected interval variable and adding new edges to repair broken paths. The relaxed POS P' is then used to enforce precedence constraints between interval variables before applying a completion strategy.

The completion strategies partially explore a search tree with a limited number of backtracks. At the root node of an LNS iteration, in case of irregular temporal costs, an important issue is finding a good time placement for the interval variables in the LNS fragment considering the frozen part of the POS. In this context some of the strategies solve the TLR described in this paper. The optimal solution of this TLR gives indicative start and end values for each interval variable. The search is a generalization of the *SetTimes* strategy recapped in [10]. It considers interval variables by increasing indicative end point values and tries in the left branch to set them as close as possible to their indicative values. When a failure occurs in the left branch, the interval variable is marked *unselectable* and will remain so until constraint propagation removes the value tried in the left branch from the domain of the variable. When the objective is regular, this strategy boils down to *SetTimes*.

The objective function is automatically analyzed before solving the problem and the completion strategy portfolio is configured as follow:

- for objective functions involving some irregular temporal terms (like earliness cost or interval length maximization), only completion strategies using the TLR are used,

– for objective functions involving terms related with presence of interval variables (like resource allocation costs) but only regular temporal objective, both strategies using the TLR and the *SetTimes* are used, in this case the learning techniques will favor the most efficient strategy on the instance being solved,
– for other objective functions (no presence-related terms and no irregular temporal terms), only strategies based on *SetTimes* are used.

Note that the TLR is also used at the root node of the global search tree to compute a lower bound on the cost.

4 Temporal Linear Relaxation Formulations

The TLR is a continuous Linear Program that represents the relaxation of the current problem solved by the engine at the root node of an LNS iteration. In CP Optimizer, this TLR is solved using CPLEX. This section describes the relaxation of most of the elements of the model.

4.1 Interval Variables and Logical Constraints

An interval variable a_i is represented by four numerical variables in the TLR: three temporal variables s_i, e_i and l_i for the start point, end point and length together with one continuous $[0, 1]$ variable x_i for the presence status. The range of temporal variables s_i, e_i and l_i is the domain of the interval start, end and length in the CP engine. The constraint on the length of interval a_i can be seen as a special cases of precedence constraint with delay between the start and the end of the interval, so it will be described in section 4.2.

In the CP engine, binary logical constraints like $\text{presenceOf}(a_i) \Rightarrow \text{presenceOf}(a_j)$ are aggregated into an implication network that is updated from the model statements and from engine deductions [12]. The TLR formulation of these constraints consists of binary linear inequalities like $x_i \leq x_j$.

4.2 Precedence Constraints

Precedence constraints are constraints like $\text{endBeforeStart}(a_i, a_j, z_{ij})$ where z_{ij} is a delay. Delay z_{ij} can be constant or variable. These constraints and the constraints on interval lengths are aggregated by CP Optimizer in a temporal constraint network [12]. A precedence relation is a constraint of the form $(x_i \wedge x_j) \Rightarrow (y_i + z_{i,j} \leq y_j)$ where x_i and x_j are the presence status of intervals a_i and a_j and y_i, y_j the interval endpoints related with the precedence constraint (s_i or e_i, s_j or e_j). Let $M_i^{i,j}$ and $M_j^{i,j}$ denote large enough coefficients, the LP formulation reads:

$$y_i + z_{i,j} \leq y_j + M_i^{i,j}(1 - x_i) + M_j^{i,j}(1 - x_j)$$

Note that in the presence of logical relations between intervals, this formulation can be simplified. For instance if $x_i \Rightarrow \neg x_j$, the precedence constraint does not need to be linearized at all because it is inactive. If the presence status of interval variables is equivalent, that is $x_i \Leftrightarrow x_j$, a single big-M coefficient can be used. This is in particular

the case for precedence constraints representing the length of interval variables as, here, y_i and y_j are endpoints of the same variable.

Let $[\underline{Y_i}, \overline{Y_i}]$ denote the domain of y_i. In order to get a tighter linear relaxation, the values of big-M coefficients are minimized by selecting a specific absence value $Y_i^0 \in [\underline{Y_i}, \overline{Y_i}]$ when the interval is absent ($x_i = 0$). The following constraint is added to fix y_i to value Y_i^0 when $x_i = 0$:

$$Y_i^0 + (\underline{Y_i} - Y_i^0)x_i \leq y_i \leq Y_i^0 + (\overline{Y_i} - Y_i^0)x_i$$

Under these conditions, if $[\underline{Z_{i,j}}, \overline{Z_{i,j}}]$ denotes the domain of $z_{i,j}$ for the general case we can select:

$$M_i^{i,j} = \max\left(0, Y_i^0 + \overline{Z_{i,j}} - \underline{Y_j}\right), M_j^{i,j} = \max\left(0, \overline{Y_i} + \overline{Z_{i,j}} - Y_j^0\right)$$

Absence values Y_i^0 are selected so as to minimize the big-M coefficients resulting in a tighter relaxation. For instance if $x_i = x_j$, $Z_{i,j}$ constant, and $Y_i^0 + Z_{i,j} \leq Y_j^0$, the big-M value is zero.

4.3 Alternative Constraints

An alternative constraint $\mathit{alternative}\,(a, \{b_1, ..., b_n\})$ is both an exclusive disjunction of the presence status of the b_i's and start and end points synchronization between a and the b_i's (precedence constraints $\text{startAtStart}(a, b_i)$, $\text{endAtEnd}(a, b_i)$). The precedences are added to the temporal network. Beside the relaxation of those precedences (see §4.2), the constraint $x(a) = \sum_{i \in \{1..n\}} x(b_i)$ is added to TLR.

4.4 Span Constraints

A span constraint $\mathit{span}\,(a, \{b_1, ..., b_n\})$ is a conjunction of:

- a set of precedences b_i starts after and ends before a that are linearized as described in §4.2.
- a set of implications $x(b_i) \Rightarrow x(a)$ that are added in the TLR ($x(b_i) \leq x(a)$)
- conditions $s(a) \geq \min_{i \in \{1..n\}} s(b_i)$ and $e(a) \leq \max_{i \in \{1..n\}} e(b_i)$ are convexified in the TLR.

4.5 Piecewise Linear Expressions on Interval Endpoints

CP Optimizer provides piecewise linear expressions evaluated on interval endpoints and lengths, for instance $\text{endEval}(a, F, V^0)$ where F is a piecewise linear function (see Table 1). This expressions evaluates to $F(\text{endOf}(a))$ when interval variable a is present and to V^0 when the interval is absent. In the TLR, this expression is convexified over the CP domain of the interval endpoint $y = e(a)$ as illustrated on Fig. 1.

When the interval variable is optional, if v_p denotes the convexified domain in the TLR, x and y respectively the presence and temporal variables of the interval of absence value of Y^0, the expression is linearized as:

$$v_p + (1 - x)(V^0 - F(Y^0))$$

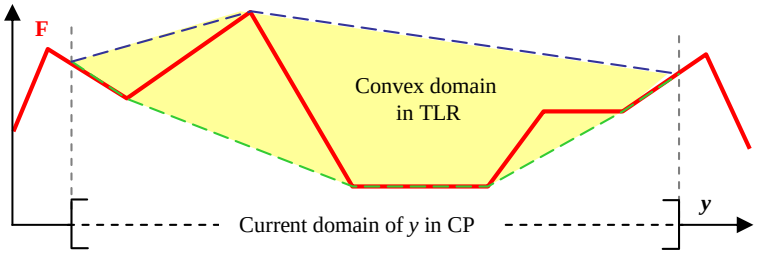

Fig. 1: Temporal relaxation of piecewise-linear expressions

More generally, all the expressions over interval variables in CP Optimizer involve some absence value V^0 that represents the value of the expression when some interval is absent [12]. Unary expressions on the ranges of an interval variable follow the same scheme as piecewise linear expressions: for example, the expression $\text{startOf}(a, V^o)$ is relaxed by $s(a) + (1 - x(a))(V^0 - S^0)$ where S^0 is the absence value of the start of a (see §4.2).

4.6 Arithmetical Expressions

CP Optimizer provides a wide range of arithmetical non-linear expressions such as $\min_i x_i$, $\max_i x_i$, xy, $1/x$, $|x|$, x^k, e^x, $\log(x)$, ...

The TLR implements some classical linear relaxations of those expressions. For instance:

- Univariate expressions are convexified in a similar way as piecewise linear expressions. For instance expressions x^{2k+1} are convexified using envelopes studied in [14],
- Bilinear terms xy assuming $0 \leq x$ and $0 \leq y$ are relaxed as $\underline{X}y \leq xy \leq \overline{X}y$ and $x\underline{Y} \leq xy \leq x\overline{Y}$

4.7 Overlap Length expressions

An *overlap length* expression $\text{overlapLength}(a_1, a_2, V^0)$ measures the length of the intersection of two intervals a_1 and a_2 as illustrated on Fig. 2 and returns value V^0 if some of the interval variables a_1 or a_2 are absent.

The overlap length is the minimum length of the four possible candidate intervals between an end point and a start point of intervals $\{a_1, a_2\}$:

$$\begin{array}{ll} \text{Candidate terms:} & \forall i, j \in \{1, 2\},\ c_{ij} = \max(0, e_j - s_i) \\ \text{Present intervals formula:} & \min(c_{11}, c_{22}, c_{12}, c_{21}) \end{array}$$

Overlap length expressions are a useful complement of precedence constraints. They can be used for measuring the partial or total temporal disjunction between two tasks or for computing the total energy required by a set of activities a_i requiring q_i units of resource over some time window b as $\sum_i q_i \, \text{overlapLength}(a_i, b)$.

The relaxation of the overlap length expression will allow to present a general scheme for the formulation in the TLR of any binary expression and constraint on

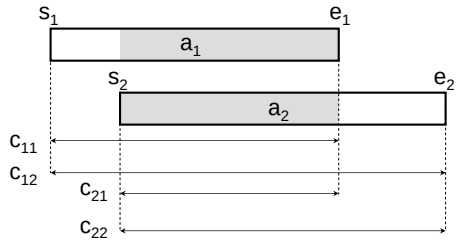

Fig. 2: Overlap length expression

interval variables. It will also illustrate how to exploit the data structure of propagation algorithms to calculate a tight convexification of the present intervals formulation.

4.7.1 Handling Presence Statuses in Binary Expressions

This section shows how similar ideas as the ones used to linearize unary expressions can be applied to binary expressions like *overlap length*.

Let's consider an expression overlapLength(a_1, a_2, V^0). The [0,1] variables in the TLR that represents the presence status of intervals a_1 and a_2 are denoted x_1, x_2 while ov_p denotes the expression calculated from interval endpoints and lengths (s_1, e_1, l_1, s_2, e_2, l_2) assuming both intervals are present. $\underline{OV}$, $\overline{OV}$ are the lower and upper bound of ov_p as computed using the current bounds in the propagation engine. OV^0 is the value of the expression as computed from the absence value of the intervals endpoints in the TLR (see §4.2). With $x_{12} = x_1 x_2$, the expression ov to relax is:

$$ov = x_{12}\, ov_p + (1 - x_{12})\, V^0$$

The formulation of these constraint in the TLR consist in relaxing $x_{12}\, ov_p$. There are two cases depending on the relation between presence statuses:

– In the general case, with x_{12} a Boolean variable and ov_p being positive:

$$\begin{aligned} \underline{OV}\, x_{12} + (1 - x_{12})\, V^0 \leq ov \leq ov_p + (1 - x_{12})\, V^0 \\ \leq \overline{OV}\, x_{12} + (1 - x_{12})\, V^0 \end{aligned} \tag{1}$$

– When both interval variables a_1 and a_2 are absent, variable ov_p will take the absence value OV^0. This can be exploited to tighten the linear relaxation depending on the logical relations in the logical network. In such cases, an equation similar with the unary expression case is obtained.

$$ov = ov_p + (1 - x_1)(V^0 - OV^0) \tag{2}$$

There are four possible formulations for the expression in the TLR.

Logical relation	Presence variable	TLR formulation
if $x_1 \Rightarrow \neg x_2$	$x_{12} = 0$	$ov = V^0$
if $x_1 \Leftrightarrow x_2$	$x_{12} = x_1$	(1)
if $x_1 \Rightarrow x_2$	$x_{12} = x_1$	(2)
otherwise	$x_{12} \leq x_1,\ x_{12} \leq x_2$ $x_1 + x_2 \leq x_{12} + 1$	(2)

The formulation (1) is also used when the involved ranges of one of the interval are fixed. The case where both intervals are present ($x_{12} = 1$) is also covered by formulation (1).

This formulation is a general scheme that can be applied to any binary expression or constraint. We will now see how the present interval formula ov_p is linearized in the case of the *overlap length* expression.

4.7.2 Convexification of Present Interval Formula ov_p

We have seen that ov_p can be formulated as the minimal value of four candidate terms $ov_p = \min(c_{11}, c_{22}, c_{12}, c_{21})$. Terms c_{ii} are the length of interval a_i, they already have a corresponding term l_i in the TLR. Terms c_{12} and c_{21} measure how much a_1 precedes or follows a_2. Their linearization in the TLR is the one of the $\max(0, e_j - s_i)$ expression.

Upper bound formulation of ov_p. The upper bound of ov_p is formulated by stating constraints $ov_p \leq c_{ij}$ for all $i, j \in \{1, 2\}$. Good practices in propagation consist in memorizing and incrementally maintaining supports for bounds on range involved in a constraint or an expression. By support, we mean a piece of information that can be updated in reasonable time and that avoids further calculations. For the upper bound of the overlap expression, with $\mathcal{C} = \{c_{ij}, i, j \in \{1, 2\}\}$, the support $\mathcal{S}$ is a minimal set of candidates required to calculate the upper bound of ov_p. Initially, we set $\mathcal{S} = \mathcal{C}$. If $\underline{C}$ and $\overline{C}$ respectively denote the lower and upper bounds on a candidate c:

$$\forall c \in \mathcal{S},\ \left\{\exists\ c' \in \mathcal{S} \setminus \{c\} \mid \overline{C}' \leq \underline{C}\right\} \Rightarrow c \notin \mathcal{S}$$

Maintaining $\mathcal{S}$ is of interest if the constrained structures involving the variables allow to reduce its size. Here are some typical usages of $\mathcal{S}$ in the propagation engine and in the TLR:

- $\mathcal{S}$ *is empty*: the propagation engine has found an inconsistency.
- $\mathcal{S}$ *is a singleton* $\{c\}$: the propagation engine filtering rule is $\overline{OV} = \overline{C}$ and the constraint $c = ov_p$ is added to the TLR.

Convexification of the lower bound of ov_p. By considering a minimal set $\mathcal{S}$, the propagation may achieve stronger domain reductions on a range of the intervals. Eventually, $\mathcal{S}$ allows to improve the convexification of ov_r. Fig. 3 illustrates the idea of this convexification. We suppose $|\mathcal{S}| > 1$. Be c and c' the two candidates in $\mathcal{S}$ such that $\underline{C}$ is the minimal value of the overlap length ($\underline{OV} = \underline{C}$) and $\underline{C}'$ the next minimal value of the overlap length.

Suppose we have $\underline{C} < \underline{C}'$. Necessarily, $\underline{C}' < \overline{OV} \leq \overline{C}$: otherwise c would be the only candidate in $\mathcal{S}$. Consequently, an upper bound inequality can be stated as:

$$(\overline{C} - \underline{OV})ov_p \geq (\underline{OV}' - \underline{OV})c + \underline{OV}(\overline{C} - \underline{OV}')$$

Removing candidates of same or near lower bound strengthen the relaxation. Next paragraph show how precedences may remove candidates.

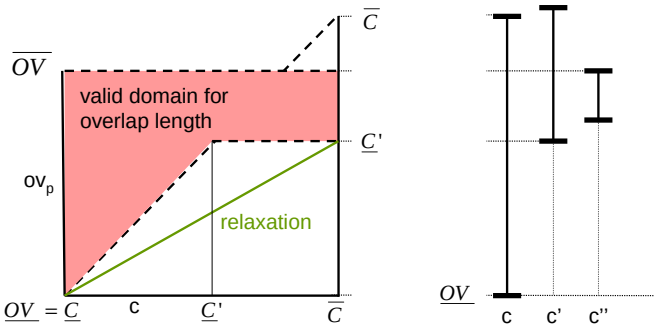

Fig. 3: Overlap length relaxation of lower bound

Candidate elimination . The propagation engine uses two kinds of argument to compute and incrementally maintain a set $\mathcal{S}$.

- *Propagation of the constraint network on the expression*: when a new upper bound ub of the overlap length is propagated, the following rule applies:

$$\forall c \in \mathcal{S},\ \left(\exists c' \in \mathcal{S} \setminus \{c\}, \min(\overline{C}', ub)) \leq \underline{C}\right) \Rightarrow c \notin \mathcal{S}$$

- *Precedence network*: if, from the precedence network or the actual bound of the range of intervals, $e_1 \leq e_2$ then $c_{11} = e_1 - s_1 \leq e_2 - s_1 \leq c_{21}$. That is c_{21} can eventually be removed from $\mathcal{S}$. The elimination rules is:

$$\forall i,j \in \{1,2\} \begin{cases} s_i \leq s_{j\neq i} \Rightarrow \begin{cases} c_{1j} \in \mathcal{S} \Rightarrow c_{1i} \notin \mathcal{S} \\ c_{2j} \in \mathcal{S} \Rightarrow c_{2i} \notin \mathcal{S} \end{cases} \\ e_i \leq e_{j\neq i} \Rightarrow \begin{cases} c_{i1} \in \mathcal{S} \Rightarrow c_{j1} \notin \mathcal{S} \\ c_{i2} \in \mathcal{S} \Rightarrow c_{j2} \notin \mathcal{S} \end{cases} \end{cases}$$

In CP Optimizer, the TLR is computed after restoration of the relaxed POS. The relaxed POS contains many precedence relations that strongly benefits the candidate elimination.

4.8 Timelines and their Constraints

Resources and more generally in CP Optimizer, timelines and their related constraints, are often discrete, sometimes disjunctive and generally non-linear concepts. That is one of the reasons why scheduling problems are often difficult to solve with MP techniques alone.

By construction of the POS in the LNS, timelines values are represented as a set of precedence constraints. Those precedence constraints give flexibility for re-inserting the interval variables of the relaxed LNS fragment. They also fit well in the TLR because they are linear constraints (see §4.2). In this context, the TLR solved at the root node of an LNS iteration will provide guidelines for re-positioning the relaxed interval variables in the frozen part of the temporal network.

A very tight relaxation of the timelines constraints is not crucial here because most of those constraints are already linearized as precedence constraints in the POS. This

being said, the TLR includes some global linear constraints related with timelines. For instance for a maximum value of a cumul function like: $f = \sum_i pulse(a_i, h_i) \leq K$, if T_{min} and T_{max} respectively denote the minimal start value and maximal end value of all possibly present intervals a_i then the following constraint must hold:

$$\sum_i l_i \, h_i \leq K(T_{max} - T_{min})$$

This type of energetic constraints are added in the TLR for each cumul function.

5 Experimental Evaluation

We studied the impact of the TLR on 15 scheduling benchmarks involving irregular temporal cost functions (T) and/or interval presence related costs (P):

1. Oversubscribed satellite communication scheduling problems [11] (P)
2. Job-shop scheduling problems with earliness/tardiness costs [1] (T)
3. Audit scheduling problems [6] (P)
4. Multi-machine assignment scheduling problem [5] (P)
5. Single machine instances of the Masc library [16] (T)
6. Personal tasks scheduling [19] (T&P)
7. RCPSP with discounted cash flows [21] (T)
8. Single machine scheduling problems with earliness/tardiness costs [7] (T)
9. Flow-shop scheduling problems with earliness/tardiness costs [15] (T)
10. A non-public scheduling problem involving earliness/tardiness costs (T)
11. Aircraft landings scheduling [2] (T)
12. Common due-date scheduling problems [4] (T)
13. Dynamic resource feasibility problems [9] (T)
14. Maximal quality RCPSP [18] (T)
15. RCPSP with earliness/tardiness costs [22] (T)

Experiments were performed with CP Optimizer V12.5. For each benchmark we estimate the speed-up factor of TLR as the ratio of the time it takes to find a solution of similar quality for a version of CP Optimizer not using the TLR and for the default version of CP Optimizer (using the TLR). Note that in CP Optimizer, the TLR can be switched off by setting parameter `TemporalRelaxation` to value `Off`. The results are summarized on Fig. 4. On the figure, the speed-up factor is capped to 100. On benchmarks with speed-up greater than 100, we noticed that even the initial solution using the TLR could never be reached within the time-limit when not using the TLR.

Three categories of benchmarks are considered: problems with irregular temporal costs, problems with presence-related cost and problems with both types of costs.

The experimental evaluation clearly shows that the TLR is essential for problems with irregular temporal costs and generally helps for problem with presence related costs.

6 Conclusion

This paper presents the temporal linear relaxation of the CP Optimizer model and its integration in the automatic search to control the completion strategies. We think

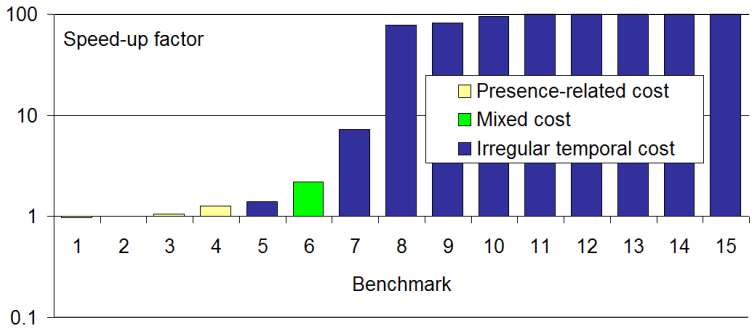

Fig. 4: Speed-up factor when using TLR

that this loose coupling between a relaxation and the CP engine at the root node of each LNS iteration is an efficient and robust method if the relaxation can exploit the dominant structures of the problem to provide well informed guidelines for the completion strategy. In the context of CP Optimizer where only a part of the POS is unfrozen, the TLR tends to be very informative as most of the resource constraints are still captured by frozen precedence arcs of the POS. For instance in the extreme case of an empty fragment where no interval variables are relaxed in the POS, the TLR boils down to the computation of the optimal placement of interval variables given the temporal network of the POS.

Future work will concern both the control and the diversification of the temporal relaxation by:

- Controlling the size and the content of the TLR depending on the characteristics of the problem and the current impact on the search,
- Exploring tighter relaxations in the TLR, for instance by solving a MILP enforcing the boolean presence status of interval variables,
- Considering concurrent relaxations that exploit other dominant structures of the problem like capacity planning or sequencing sub-problems.

References

1. P. Baptiste, M. Flamini, and F. Sourd. Lagrangian bounds for just-in-time job-shop scheduling. *Computers and Operations Research*, 35:906–915, 2008.
2. J.E. Beasley, M. Krishnamoorthy, Y.M. Sharaiha, and D. Abramson. Scheduling aircraft landings - the static case. *Transportation Science*, 34:180–197, 2000.
3. C. Beck and P. Refalo. A hybrid approach to scheduling with earliness and tardiness costs. In *Proc. 3th International CP-AI-OR Conference (CP-AI-OR 2001)*, 2001.
4. D. Biskup and M. Feldmann. Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates. *Computers and Operation Research*, 28(8):787–801, 2001.
5. A. Bockmayr and N. Pisaruk. Detecting infeasibility and generating cuts for mixed integer programming using constraint programming. *Computers and Operations Research*, 33:2777–2786, 2006.
6. P. Brucker and D. Schumacher. A new tabu search procedure for an audit-scheduling problem. *Journal of Scheduling*, 2:157–173, 1999.
7. K. Bulbul, P. Kaminsky, and C. Yano. Preemption in single machine earliness/tardiness scheduling. *Journal of Scheduling*, 10:271–292, 2007.

8. E. Danna and L. Perron. Structured vs. unstructured large neighborhood search: A case study on job-shop scheduling problems with earliness and tardiness costs. In *Proc. 9th International CP Conference (CP 2003)*, pages 817–821, 2003.
9. H. el Sakkout, T. Richards, and M. Wallace. Minimal perturbation in dynamic scheduling. In *Proc. ECAI-98*, 1998.
10. D. Godard, P. Laborie, and W. Nuijten. Randomized Large Neighborhood Search for Cumulative Scheduling. In *Proc. ICAPS-05*, pages 81–89, 2005.
11. L. A. Kramer, L. V. Barbulescu, and S. F. Smith. Understanding Performance Tradeoffs in Algorithms for Solving Oversubscribed Scheduling. In *Proc. 22nd AAAI Conference on Artificial Intelligence (AAAI-07)*, pages 1019–1024, 2007.
12. P. Laborie and J. Rogerie. Reasoning with conditional time-intervals. In *Proc. 21th International FLAIRS Conference (FLAIRS 2008)*, 2008.
13. P. Laborie, J. Rogerie, P. Shaw, and P. Vilím. Reasoning with Conditional Time-intervals, Part II: an Algebraical Model for Resources. In *Proc. 22th International FLAIRS Conference (FLAIRS 2009)*, 2009.
14. L. Liberti and C. C. Pantelides. Convex envelopes of monomials of odd degree. *Journal of Global Optimization*, 25:157–168, 2003.
15. T. Morton and D. Pentico. *Heuristic Scheduling Systems*. Wiley, 1993.
16. W. Nuijten, T. Bousonville, F. Focacci, D. Godard, and C. Le Pape. Towards an industrial manufacturing scheduling problem and test bed. In *Proc. 9th Int. Conf. on Project Management and Scheduling*, 2004.
17. N. Policella, A. Cesta, A. Oddi, and S.F. Smith. Generating robust schedules through temporal flexibility. In *Proceedings ICAPS-04*, Whistler, Canada, June 2004.
18. N. Policella, X. Wang, S.F. Smith, and A. Oddi. Exploiting temporal flexibility to obtain high quality schedules. In *Proc. AAAI-2005*, 2005.
19. I. Refanidis. Managing personal tasks with time constraints and preferences. In *Proc. 17th International Conference on Automated Planning and Scheduling Systems (ICAPS-07)*, pages 272–279, 2007.
20. P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Proc. CP-98*, pages 417–431, 1998.
21. M. Vanhoucke. A scatter search heuristic for maximising the net present value of a resource-constrained project with fixed activity cash flows. *International Journal of Production Research*, 48:1983 – 2001, 2010.
22. M. Vanhoucke, E. Demeulemeester, and W. Herroelen. An exact procedure for the resource-constrained weighted earliness-tardiness project scheduling problem. *Annals of Operations Research*, 102(1-4):179–196, 2001.