

Temporal linear relaxation in IBM ILOG CP Optimizer

Philippe Laborie · Jérôme Rogerie

Received: 29 October 2013 / Accepted: 12 November 2014
© Springer Science+Business Media New York 2014

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.

Keywords Scheduling · Constraint programming · Linear programming · Irregular objective functions

1 Introduction

IBM ILOG CP Optimizer is a general constraint solver based on constraint programming (CP) techniques that implement a model-and-run paradigm. This paper focuses on scheduling problems. For these problems, CP Optimizer provides a relatively simple but very expressive modeling language based on the notion of interval variables (Laborie and Rogerie 2008; Laborie et al. 2009). 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.*). When this type of objective function is involved, providing a good time placement of activities can be a challenge even in the absence of resource constraints.

In this context, using a linear relaxation of the problem solved with Mathematical Programming (MP) 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 model solving, column generation (Desaulniers et al. 2005), and logical Bender's decomposition (Cîră et al. 2013)]. Although very efficient on specific scheduling problems, these approaches tend to be difficult to automate. In *hybrid* approaches like Refalo (2000), Beck and Refalo (2001), and Danna and Peron (2003), the solution to the linear relaxation is maintained 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. Unlike in hybrid approaches, the linear relaxation is only synchronized at every iteration of the optimization process that is to say, much less frequently than at each search node. Several linear formulations of resources have been developed in MP (Hooker 2006), but all these formulations are quite expensive. As we will see, our approach works around this complexity issue because we do not need a tight formulation of resource constraints in the relaxation. We will take advantage of the ability of CP to handle resource contention, whereas MP will essentially be used for optimizing resource selection and the positioning of activities in time given a set of temporal constraints. The Temporal Linear Relaxation (TLR) is a linear relaxation of

P. Laborie · J. Rogerie (✉)
IBM, Software Group, Paris, France
e-mail: rogerie@fr.ibm.com

P. Laborie
e-mail: laborie@fr.ibm.com

the scheduling model that handles temporal decisions (start and end of tasks, precedences) as well as most of the assignment decisions (the resource on which a task is processed).

After describing the CP Optimizer model for scheduling in Sect. 2, we give an overview of the architecture of the automatic search for scheduling problems in Sect. 3. The main purpose of this section is to show how the proposed 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 the classical Constraint Programming paradigm by introducing additional mathematical concepts (such as intervals Laborie and Rogerie 2008, sequences, or functions Laborie et al. 2009) 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 $\{\perp\} \cup \{[s, e) : s, e \in \mathbb{Z} \wedge s \leq e\}$. A Boolean $x(a)$ denotes the presence status of the interval variable. It is part of the decisions of the problem to decide whether interval a is present ($x(a)$ is true) or absent ($x(a)$ is false). More precisely, 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. In this case, $s(a)$, $e(a)$, and $l(a)$ have no meaning¹.

Absent interval variables have an operational meaning : an optional activity that has not been selected to be executed in the actual schedule will be represented by an absent interval variable in the solution. 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 the 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 variables is quite versatile. A typical use case 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), and the others being absent. By design, interval variables are optional, and they can be made non-optional by adding a constraint 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, b , and b_i denote interval variables and z an integer variable:

- A *presence constraint* $\text{presenceOf}(a)$ states that interval a is present, that is, $a \neq \perp$. This constraint can be used to compose an expression, for instance the constraint $\text{presenceOf}(a) \Rightarrow \text{presenceOf}(b)$ means that the presence of a implies the presence of b .
- A *precedence constraint* (e.g., $\text{endBeforeStart}(a, b, z)$) specifies a precedence between interval endpoints with an integer or variable minimal distance z provided that both intervals a and b are present.
- A *span constraint* $\text{span}(a, \{b_1, \dots, b_n\})$ states that if a is present, it starts together with the first present interval in $\{b_1, \dots, b_n\}$ and ends together with the last one. Interval a is absent if and only if all the b_i are absent.
- An *alternative constraint* $\text{alternative}(a, \{b_1, \dots, b_n\})$ models an exclusive alternative between $\{b_1, \dots, b_n\}$: if interval a is present, then exactly one of intervals $\{b_1, \dots, b_n\}$ 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 (Laborie and Rogerie 2008).

CP Optimizer also provides numerical expressions related with the integer bounds (start, end, and length) of an interval variable. These expressions are shown in 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, $\text{endEval}(a, F, V^0)$ could be used to model an earliness /tardiness cost function F for an optional activity

¹ It is to be noted that $x(a)$, $s(a)$, $e(a)$, and $l(a)$ are not proper decision variables of the problem, and we use them here only for the purpose of defining interval variables. In the model, interval variables are basic variables, and they are not an assembly of lower level variables. The different features of an interval variable (presence status, start, end, and length) are constrained via some expressions that we will see later (Table 1).

Table 1 Expression semantics

$\text{startOf}(a, V^0)$	$s(a)$ if $x(a)$ is true V^0 otherwise
$\text{endOf}(a, V^0)$	$e(a)$ if $x(a)$ is true V^0 otherwise
$\text{lengthOf}(a, V^0)$	$l(a)$ if $x(a)$ is true V^0 otherwise
$\text{startEval}(a, F, V^0)$	$F(s(a))$ if $x(a)$ is true V^0 otherwise
$\text{endEval}(a, F, V^0)$	$F(e(a))$ if $x(a)$ is true V^0 otherwise
$\text{lengthEval}(a, F, V^0)$	$F(l(a))$ if $x(a)$ is true V^0 otherwise

modeled by an interval variable a evaluated on its end time, while value V^0 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.

For modeling resources, CP Optimizer offers concepts such as *cumul functions*, *state functions*, and *interval sequences*. These concepts are summarized below and detailed in Laborie et al. (2009).

- *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 $f = \sum_i \text{pulse}(a_i, h_i)$ with a pulse function $\text{pulse}(a_i, h_i)$. When interval variable a_i is present, a pulse function $\text{pulse}(a_i, h_i)$ is a step function equal to value h_i between the start and end of interval a_i and equal to value 0 outside this 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 where h_i is the quantity of resource used by activity a_i and maximal value K is the capacity of the resource.
- *State functions* A state function is a decision variable whose value is any chronological chain of non-overlapping segments $[s_i, e_i)$ associated with a positive integer state v_i . Different constraints between a state function variable f and an interval variable a are available to state that if interval a is present, (1) any segment $[s_i, e_i)$ of the state function overlapping a should have its state v_i within an integer range passed as argument to the constraint, (2) a segment of the state function starts

at the start point of interval a , (3) a segment of the state function ends at the endpoint of the interval a , (4) a segment of the state function covers interval a , or (5) a does not overlap any segment of the state function. State functions are typically used to model resources characterized by a state that may be changed over time (for instance, the temperature of an oven) and such that some activities require the resource to be in a particular state (or set of states) during their execution.

- *Interval sequence variables* An interval sequence variable is defined by a set of interval variables S . The value of an interval sequence variable is a permutation of the present intervals in set S . Different constraints on interval sequence variables are available to state (1) precedence relations between interval variables in the permutation, (2) temporal relations between endpoints of the intervals (for example, a *noOverlap* constraint states that, according to the permutation order, any interval in the sequence must end before its successor starts). Interval sequence variables and *noOverlap* constraints are typically used to model disjunctive unary resources and the possible transition constraints between consecutive activities on the resource (like sequence-dependent setup times or costs).

We group these three concepts under the generic term of *timeline*. Timelines are used to constrain the time positioning of a set of interval variables. The combination of constraints on the above timeline expressions together with structural constraints on interval variables like *span* and *alternative* facilitates the modeling of complex resources.

3 Integration of TLR in CP Optimizer automatic search

Large neighborhood search (LNS) (Shaw 1998) 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 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.

The large neighborhoods are all based on the initial generation of a partial order schedule (POS) (Policella et al. 2004) 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 nodes in \mathcal{A} are the interval

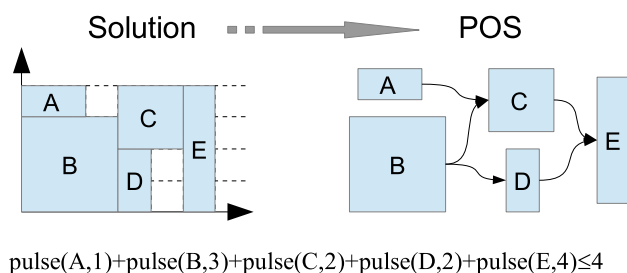


Fig. 1 Example of a partial order schedule (POS)

variables of the problem and 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 Godard et al. (2005), Laborie and Rogerie (2008). An example of generated POS for a single cumul function of maximal value 4 involving 5 interval variables is shown in Fig. 1. Large neighborhoods in the portfolio differ in the way they select a subset of interval variables to be relaxed (this subset is called the LNS *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. Furthermore, the presence status of interval variables belonging to the LNS fragment is also relaxed. Typically, this will allow the relaxed interval variables to be re-allocated on different resources. The relaxed POS P' is then used to enforce precedence constraints between interval variables before applying a completion strategy to re-optimize the relaxed fragment.

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. Furthermore, in case of objective function related with resource allocation, it is important to reallocate the activities to the cheapest or the most efficient resources, and this boils down to re-allocating the presence status of the interval variables in the LNS fragment. In this context, some of the strategies solve the TLR described in this paper. The optimal solution of this TLR gives continuous indicative values ($x_i \in [0, 1]$, s_i , e_i) for the presence, start, and end of each interval variable a_i .

The completion strategy based on the TLR implements a generalization of the *SetTimes* strategy described in Godard et al. (2005). It considers interval variables in a chronological way by increasing indicative endpoint values (s_i , e_i) in the TLR solution. If the selected interval variable a_i has an

unfixed presence status in the CP engine, it will create a decision point to fix the presence status. The decision is randomized so that the probability to first branch on setting the interval a_i absent is equal to x_i , that way, the closer x_i is to 1, the larger the probability to set it present on the left branch. If the selected interval variable a_i is already present in the CP engine, then a decision point is created to fix the interval position (start, end). In the left branch, the strategy will try to set it as close as possible to its indicative values s_i and e_i . When a failure occurs in the left branch, the interval variable is marked as *unselectable*, meaning that it cannot be selected anymore by the strategy, 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 and all interval variables are present, this strategy boils down to *SetTimes*. It is to be noted that the above search strategy naturally interleaves the fixing of presence status and start/end values for the different variables, while for a given interval variable, presence status is always fixed before its start/end values.

The strategies used in the automatic search follow from the type of objective function which is automatically inspected before solving the problem. The completion strategy portfolio is configured as follows:

- 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 the 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, and
- for other objective functions (no presence-related terms and no irregular temporal terms), only strategies based on *SetTimes* are used.

Beside its use as a heuristic to guide the search in the completion strategies, the TLR is also used for two other purposes:

- A TLR is solved at the root node of the global search in order to compute a lower bound on the cost. The lower bound allows to stop the search when a feasible solution is found with a cost equal to this value. It is useful in some particular cases for which the linear relaxation is tight enough, for instance a Simple Temporal Network with convex piecewise linear start/end costs.
- Each time a TLR is solved at the root node of an LNS iteration, a propagation based on reduced costs is performed to reduce the variable domains.

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. In this section, lower case symbols represent variables, whereas upper case symbols represent constants. We use underline and overline to represent the current lower and upper bounds of a variable, so for instance if y is a variable, its current lower and upper bound will be, respectively, denoted \underline{y} and \overline{y} . We also use a superscript notation Y^0 to denote the particular value taken by a variable or expression y when the related interval variable(s) is (are) absent.

4.1 Interval variables and logical constraints

An interval variable a_i is represented by four numerical variables in the TLR: three continuous 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 case of precedence constraint with delay between the start and the end of the interval, and it will be described in Sect. 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 (Laborie and Rogerie 2008). The TLR formulation of these constraints consists of binary linear inequalities like $x_i \leq x_j$.

4.2 Precedence constraints

The precedence conditions are constraints like $\text{endBeforeStart}(a_i, a_j, z_{ij})$ where z_{ij} is a delay. Delay z_{ij} can be a constant or variable. These constraints and the constraints on interval lengths are aggregated by CP Optimizer in a temporal constraint network (Laborie and Rogerie 2008). 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 $[Y_i, \overline{Y}_i]$ 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 $Y_i^0 \in [Y_i, \overline{Y}_i]$ 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 $[Z_{i,j}, \overline{Z}_{i,j}]$ 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.

It is important to keep in mind that even when the interval variable is absent ($x_i = 0$), in the TLR, the interval endpoints y_i (s_i or e_i) are still defined and associated with a particular absence value Y_i^0 (S_i^0 or E_i^0). We have seen in the model that expressions on interval variables (like $\text{startOf}(a, V)$) must take the specified value V when the interval variable is absent. It will be part of the role of the linearization of expressions to reconcile the absence value of the interval endpoints in the TLR (like S_i^0 in the case of startOf) with the absence value V of the expression (see Sects. 4.5 and 4.7).

4.3 Alternative constraints

An alternative constraint $\text{alternative}(a, \{b_1, \dots, b_n\})$ is both an exclusive disjunction of the presence status of the b_i and start and end points synchronization between a and the b_i (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 Sect. 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 $\text{span}(a, \{b_1, \dots, b_n\})$ is a conjunction of

- a set of precedences $\text{startBeforeStart}(a, b_i)$ and $\text{endBeforeEnd}(b_i, a)$. These precedence constraints are linearized as described in Sect. 4.2,

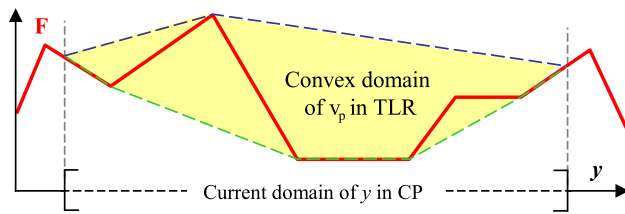


Fig. 2 Temporal relaxation of piecewise linear expressions

- a set of implications $x(b_i) \Rightarrow x(a)$ that are added in the TLR ($x(b_i) \leq x(a)$), and
- as the spanning interval cannot start (resp. end) before (resp. after) the minimal start value (resp. maximal end value) of the spanned intervals, the remaining endpoints 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

CP Optimizer provides piecewise linear expressions evaluated on interval endpoints and lengths (see Table 1). We have for instance expression $\text{endEval}(a, F, V^0)$ where F is a piecewise linear function. The value of this expression is $F(\text{endOf}(a))$ when interval a is present and V^0 when the interval is absent. Let y denote the endpoint of the interval variable involved in the expression. For instance in the case of $\text{endEval}(a, F, V^0)$, y denotes the end of interval variable a .

Let us suppose for now that interval variable a is present. In the TLR, the value v_p of the expression² is convexified over the CP domain $[Y, \bar{Y}]$ of the interval endpoint y as illustrated in Fig. 2.

When the interval variable is optional, if x denotes the presence status of the interval variable, the expression is linearized as $v = v_p + (1 - x)(V^0 - F(Y^0))$.

Indeed, when the interval variable is present ($x = 1$), we will be back to the above case with $v = v_p$. When the interval variable is absent ($x = 0$), we will have $y = Y^0$ by definition of Y^0 (see Sect. 4.2). In this case by definition of v_p as being the convex domain of F , $F(Y^0)$ will still be a possible value for v_p , and thus $F(Y^0) + (1 - x)(V^0 - F(Y^0)) = V^0$ a possible value for the expression.

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 (Laborie and Rogerie 2008). Unary expressions on the ranges of an interval variable follow the same scheme as piecewise linear expressions: for example, the expression $\text{startOf}(a, V^0)$ 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 Sect. 4.2).

² Subscript p stands for *presence* value as this is the value of the expression in case the interval variable is present.

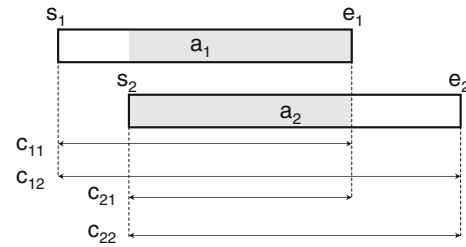


Fig. 3 Overlap length expression

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 , $\log(x)$, etc.

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 Liberti and Pantelides (2003), and
- Bilinear terms xy assuming $x \geq 0$ and $y \geq 0$ are relaxed and represented by a new variable z such that $\underline{X}y \leq z \leq \bar{X}y$ and $x\underline{Y} \leq z \leq x\bar{Y}$

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 in Fig. 3. The expression returns value V^0 if some of the interval variables a_1 or a_2 is absent.

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

Candidate terms: $\forall i, j \in \{1, 2\}, c_{ij} = \max(0, e_j - s_i)$

Overlap length: $\min(c_{11}, c_{22}, c_{12}, c_{21})$.

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 $\sum_i q_i \text{overlapLength}(a_i, b, 0)$.

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

4.7.1 Handling optionality 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 us consider the expression $\text{overlapLength}(a_1, a_2, V^0)$. Let x_1 and x_2 denote the $[0,1]$ continuous variables in the TLR that represents the presence status of intervals a_1 and a_2 . Variable ov_p denotes the value of the overlap length expression computed by considering the values of the end-points of the interval variables in the TLR ($s_1, e_1, l_1, s_2, e_2, l_2$). When both interval variables x_1 and x_2 are present, the value of this expression coincides with the value of expression $\text{overlapLength}(a_1, a_2, V^0)$, that is why we call such an expression ov_p the *present interval formula*. Its formulation will be described in Sect. 4.7.2. When both interval variables x_1 and x_2 are absent, the value of ov_p denoted OV^0 will be equal to the overlap length of intervals $[S_1^0, E_1^0]$ and $[S_2^0, E_2^0]$. \underline{OV} and \overline{OV} denote the lower and upper bound of ov_p as computed using the current bounds in the propagation engine. With $x_{12} = x_1 x_2$, the overlap length expression ov to linearize is

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

We consider two cases depending on whether or not there exists some relation between the presence statuses x_1 and x_2 .

General case (no relation between x_1 and x_2). Relation $x_{12} = x_1 x_2$ is linearized as $x_{12} \leq x_1$, $x_{12} \leq x_2$, and $x_1 + x_2 \leq x_{12} + 1$. The formulation in the TLR consists in linearizing the bilinear term $x_{12}ov_p$ in Eq. 1:

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

Case with dependencies between x_1 and x_2 . The formulation of the general case above can be tighten in case some dependency exists between the presence statuses:

- if $x_1 \Rightarrow \neg x_2$, we know that $x_{12} = 0$ and $ov = V^0$
- if $x_1 \Rightarrow x_2$, we know that $x_{12} = x_1$ so we can directly use x_1 instead of x_{12} in the linearization of $x_{12}ov_p$
- if $x_1 \Leftrightarrow x_2$, we can linearize the overlap length as $ov = ov_p + (1 - x_1)(V^0 - OV^0)$

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

For the overlap length expression, 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 :

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

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_p . Figure 4 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}' is 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.

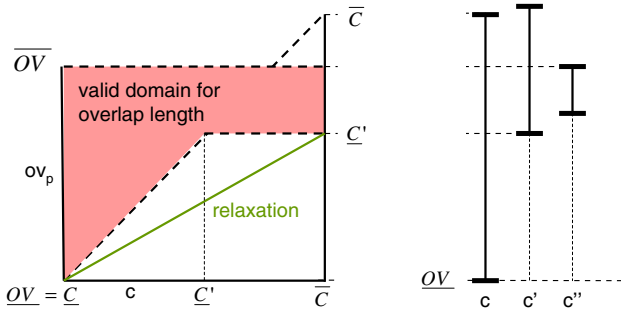


Fig. 4 Overlap length relaxation of lower bound

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:

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

- *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 are

$$\forall i, j \in \{1, 2\} | i \neq j \left\{ \begin{array}{l} s_i \leq s_j \Rightarrow \begin{cases} c_{1j} \in \mathcal{S} \Rightarrow c_{1i} \notin \mathcal{S} \\ c_{2j} \in \mathcal{S} \Rightarrow c_{2i} \notin \mathcal{S} \end{cases} \\ e_i \leq e_j \Rightarrow \begin{cases} c_{i1} \in \mathcal{S} \Rightarrow c_{j1} \notin \mathcal{S} \\ c_{i2} \in \mathcal{S} \Rightarrow c_{j2} \notin \mathcal{S} \end{cases} \end{array} \right.$$

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

Timelines (resources) 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. Indeed, although several linear formulations of resources have been developed in MP (Hooker 2006), these formulations are quite expensive. Typically, their size depends on the number of time values for time-indexed formulations of cumulative resources or is quadratic in the number of activities for disjunctive formulations.

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 Sect. 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 that 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 is added in the TLR for each cumul function.

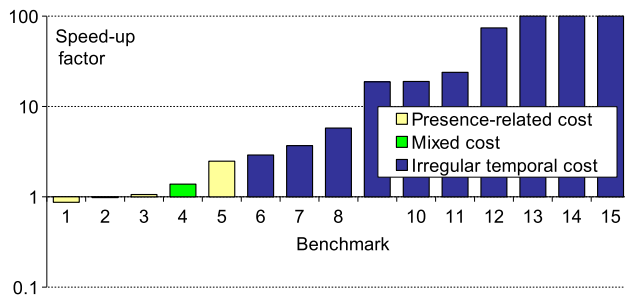
5 Experimental evaluation

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) Audit scheduling problems (Brucker and Schumacher 1999) (P)
- (2) Job-shop with earliness/tardiness costs (Baptiste et al. 2008) (T)
- (3) Oversubscribed satellite communication scheduling (Kramer et al. 2007) (P)
- (4) Personal tasks scheduling (Refanidis 2007) (T&P)
- (5) Multi-machine assignment scheduling (Bockmayr and Pizaruk 2006) (P)
- (6) RCPSP with discounted cash flows (Vanhoucke 2010) (T)
- (7) Single machine instances of the Masc Lib (Nuijten et al. 2004) (T)
- (8) Flow-shop with earliness/tardiness costs (Morton and Pentico 1993) (T)
- (9) RCPSP with earliness/tardiness costs (Vanhoucke et al. 2001) (T)
- (10) Aircraft landings scheduling (Beasley et al. 2000) (T)
- (11) Single machine scheduling with earliness/tardiness costs (Bulbul et al. 2007) (T)
- (12) Maximal quality RCPSP (Policella et al. 2005) (T)
- (13) A non-public scheduling problem with earliness/tardiness costs (T)
- (14) Common due-date scheduling problems (Biskup and Feldmann 2001) (T)

Table 2 Benchmarks overview

Benchmark	Instances	Size	Time limit
1	40	[94,561]	[60,177]
2	48	[63,411]	[105,588]
3	150	[4300,7270]	[176,264]
4	70	[95,915]	[300,300]
5	34	[65,663]	[200,200]
6	50	[403,403]	[300,300]
7	116	[17,1000]	[60,1058]
8	12	[34,309]	[105,917]
9	60	[43,103]	[176,176]
10	25	[22,254]	[60,176]
11	40	[102,402]	[176,705]
12	400	[51,66]	[60,60]
13	1	4280	706
14	20	[202,402]	[350,700]
15	18	[151,301]	[177,353]

**Fig. 5** Speed-up factor when using TLR

(15) Dynamic resource feasibility problems (Sakkout 1998) (T)

Experiments were performed with CP Optimizer V12.6.0.1. For each instance of the above benchmarks, we generate a reference by running for a certain time CP Optimizer with the TLR switched off. The time limit depends on the benchmark and the problem size. The speed-up factor measures on average how much less time it takes to get to that reference with the TLR activated (default settings). The number of instances of each benchmark as well as the range of problem sizes (in number of variables) and time limits (in s) are described on Table 2. Note that in CP Optimizer, the TLR can be switched off by setting the search parameter `TemporalRelaxation` to value `Off`.

The results are summarized in Fig. 5. In 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 the 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 the 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 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, and
- Considering concurrent relaxations that exploit other dominant structures of the problem like capacity planning, general assignment, or sequencing problems.

References

- Baptiste, P., Flamini, M., & Sourd, F. (2008). Lagrangian bounds for just-in-time job-shop scheduling. *Computers and Operations Research*, 35, 906–915.
- Beasley, J., Krishnamoorthy, M., Sharaiha, Y., & Abramson, D. (2000). Scheduling aircraft landings—The static case. *Transportation Science*, 34, 180–197.
- Beck, C., & Refalo, P. (2001). A hybrid approach to scheduling with earliness and tardiness costs. In *Proceedings of the 3th international CP-AI-OR conference (CP-AI-OR 2001)*.
- Biskup, D., & Feldmann, M. (2001). Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates. *Computers and Operation Research*, 28(8), 787–801.
- Bockmayr, A., & Pisaruk, N. (2006). Detecting infeasibility and generating cuts for mixed integer programming using constraint programming. *Computers and Operations Research*, 33, 2777–2786.

- Brucker, P., & Schumacher, D. (1999). A new tabu search procedure for an audit-scheduling problem. *Journal of Scheduling*, 2, 157–173.
- Bulbul, K., Kaminsky, P., & Yano, C. (2007). Preemption in single machine earliness/tardiness scheduling. *Journal of Scheduling*, 10, 271–292.
- Ciré, A., Coban, E., & Hooker, J. (2013). Mixed integer programming vs. logic-based benders decomposition for planning and scheduling. In *Proceedings of the CP-AI-OR 2013*.
- Danna, E., & Perron, L. (2003). Structured versus unstructured large neighborhood search: A case study on job-shop scheduling problems with earliness and tardiness costs. In *Proceedings of the 9th International CP Conference (CP 2003)* (pp. 817–821).
- Desaulniers, G., Desrosiers, J., & Solomon, M. (Eds.). (2005). *Column generation*. New York: Springer.
- Godard, D., Laborie, P., & Nuijten, W. (2005). Randomized Large Neighborhood Search for Cumulative Scheduling. In *Proceedings of the ICAPS-05* (pp. 81–89).
- Hooker, J. (2006). *Integrated methods for optimization*. Heidelberg: Springer.
- Kramer, L.A., Barbulescu, L.V., & Smith, S.F. (2007). Understanding performance tradeoffs in algorithms for solving oversubscribed scheduling. In *Proceedings of the 22nd AAAI conference on artificial intelligence (AAAI-07)* (pp. 1019–1024).
- Laborie, P., & Rogerie, J. (2008). Reasoning with conditional time-intervals. In *Proceedings of the 21th international FLAIRS conference (FLAIRS 2008)*.
- Laborie, P., Rogerie, J., Shaw, P., & Vilím, P. (2009). Reasoning with conditional time-intervals, Part II: An algebraical model for resources. In *Proceedings of the 22th international FLAIRS conference (FLAIRS 2009)*.
- Liberti, L., & Pantelides, C. C. (2003). Convex envelopes of monomials of odd degree. *Journal of Global Optimization*, 25, 157–168.
- Morton, T., & Pentico, D. (1993). *Heuristic scheduling systems*. New York: Wiley.
- Nuijten, W., Bousonville, T., Focacci, F., Godard, D., & Pape, C.L. (2004). Towards an industrial manufacturing scheduling problem and test bed. In *Proceedings of the 9th international conference on project management and scheduling*.
- Policella, N., Cesta, A., Oddi, A., & Smith, S. (2004). Generating robust schedules through temporal flexibility. In *Proceedings ICAPS-04*, Whistler.
- Policella, N., Wang, X., Smith, S., & Oddi, A. (2005). Exploiting temporal flexibility to obtain high quality schedules. In *Proceedings of the AAAI-2005*.
- Refalo, P. (2000). Linear formulation of constraint programming models and hybrid solvers. In *Proceedings of the CP-2000*.
- Refanidis, I. (2007). Managing personal tasks with time constraints and preferences. In *Proceedings of the 17th international conference on automated planning and scheduling systems (ICAPS-07)* (pp. 272–279).
- el Sakkout, H., Richards, T., & Wallace, M. (1998). Minimal perturbation in dynamic scheduling. In *Proceedings of the ECAI-98*.
- Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *Proceedings of the CP-98* (pp. 417–431).
- Vanhoucke, M. (2010). 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.
- Vanhoucke, M., Demeulemeester, E., & Herroelen, W. (2001). An exact procedure for the resource-constrained weighted earliness-tardiness project scheduling problem. *Annals of Operations Research*, 102(1–4), 179–196.