Continuous Filling and Emptying of Storage Systems in Constraint-based Scheduling

Francis Sourd
CNRS-LIP6
4, place Jussieu
75252 Paris Cedex 05, France
Francis.Sourd@lip6.fr

Jérôme Rogerie
ILOG
9, rue de Verdun
94253 Gentilly Cedex, France
jrogerie@ilog.fr

Abstract

In constraint-based scheduling, storage systems are modeled by a class of resources called reservoirs. Activities can either produce a quantity of product to be stored in the reservoir or consume it, removing the quantity from the reservoir. Most of the models presented in the literature assume that the production and the consumption is instantaneous but it can be a very bad approximation for some problems, for example those dealing with tanks of fluid. This paper introduces the continuous reservoir model in which the activity fills or empties the reservoir at a constant rate from its start time to its end time. This model is generalized to deal with additional constraints, such that leaks or overflows, that cannot be modeled by activities. Timetabling, a classical technique for the propagation of resource constraints in constraint-based scheduling, is adapted and extended to this new model.

Keywords: Constraint-based scheduling, continuous reservoir.

1 Introduction

Inventory management is of key importance in production and project scheduling. Indeed, when necessary products are unavailable at the time they are required or when intermediate (or final) products cannot be stored, some activities must be delayed, which may be costly if they are critical. Basically, there are two main constraints that must be satisfied when one encounters a storage system (e.g. tank of oil, depository of raw materials or storehouse of products that must be delivered to customers) in a process scheduling problem. The first one is to be sure that an activity whose execution consumes some amount

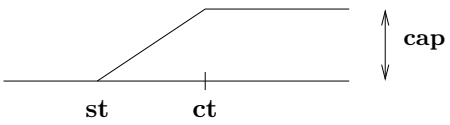

Figure 1: Contribution curve of a filling activity

of the stored product can actually use this amount of product at the time it is processed. The second one states that when an activity produces some amount of the stored product, there must be enough room in the storage system to receive it.

In this paper, we will use the generic term "*reservoir*" to refer to the storage system. We will call *level of the reservoir* the curve that represents at any time point the quantity that is stored in the reservoir when all the activities that produce or consume the product stored in the reservoir are scheduled. In the entire schedule time interval, the level of the reservoir must neither be negative nor exceed the *capacity of the reservoir* that is the maximum quantity that can be stored in the reservoir. So, the reservoir is very similar to resources as they are considered in "machine scheduling", the level of the reservoir being similar to the load of the resource. However, we remark that an "unlimited" number of producing and consuming activities can be processed at the same time provided that the sum of the consumption and production stays bounded.

In the literature [8, 10], the assumption that the action of filling or emptying the reservoir is instantaneous is generally made. Under this assumption, the level of the reservoir is a stepwise function, that is the level of the reservoir is modified at some time point, corresponding to one or several depletions or replenishments, and remains constant till the next discrete change. But these models with instantaneous depletions and replenishments may be too crude for some real-world process scheduling problems, typically in chemical industry [9]. For example, the filling or the emptying of a tank by a liquid is usually subject to a constant rate depending of the size and the number of the taps and siphons. A second typical example is the load and unload of batteries.

In this paper, we introduce the *continuous reservoir*. In this model, the activity fills or empties the reservoir at a constant rate, that is the replenishment or the depletion is continuous and linear between the start time and the end time of the activity. For example, if an activity, that starts at time **st** and completes at time **ct**, can fill the reservoir by a capacity[1] **cap**, the quantity put in the continuous reservoir by the activity at time t (also

[1]The terminology *capacity* comes from ILOG Scheduler model [6]. It refers to the capacity of the resource constraint.

stated as *contribution* of the activity at time t) is, as depicted in Figure 1:

$$\begin{cases} 0 & \text{if } t < \mathbf{st} \\ \frac{t-\mathbf{st}}{\mathbf{ct}-\mathbf{st}}\mathbf{cap} & \text{if } \mathbf{st} \leq t < \mathbf{ct} \\ \mathbf{cap} & \text{if } \mathbf{ct} \leq t \end{cases}$$

If the duration of the activity is null, we have the additional constraint that $\mathbf{st} = \mathbf{ct}$. So the filling (or emptying) operation is not continuous since the quantity **cap** is instantaneously put in (or removed from) the reservoir at time **st** — which is equal to **ct**. The contribution function is no more continuous but stepwise. Therefore, the continuous reservoir model subsumes the "discrete" reservoir model that assumes that any filling or emptying operation is instantaneous[2].

Note that this model makes no assumption on the duration of the filling or emptying process. In the context of an implementation in a constrained-based system, which is the aim of this work, the model is then seen as a global constraint between the capacity, start and completion variables of all the activities that fill or empty the reservoir. We refer to the book of Baptiste, Le Pape and Nuijten[2] or the paper of Laborie [8] or ILOG manuals [7, 6] for an introduction to constraint-based scheduling. Brucker [3] gives another recent presentation but timetable constraints, which are the main topic of this paper, are not presented in this paper. In short, the aim of a global constraint, such as the one presented in this paper, is very similar to a dynamic data structure. The constraint maintains the domain of all the variables attached to the constraint, that is the list of the possible values that can be taken by the variable. The only operation we can do is to remove some possible values in the domain of a variable. In response, the *constraint propagation* algorithm removes from the domain of other variables the values that can be proved to be inconsistent with the mathematical meaning of the constraint.

In our constraint-based approach, the processing time of each activity can be a variable. For each variable, there is a constraint linking this variable to the two variables representing the start and completion times. The processing time variable can of course also appear in other constraints.

The work presented in this paper is an extension of the timetable constraint that is a classical technique for the propagation of resource constraints in constraint-based scheduling [2, 8]. A second, very important, generalization of the timetable constraint is the introduction of the *function variables* that enable to model some effects on the level of

[2]However, as we will see in Section 5, in our implementation in ILOG Scheduler 5.2 [6], some features of the discrete reservoir cannot be extended to continuous reservoirs.

the reservoir that cannot be represented by filling and emptying activities. This facility is exemplified by the introduction of *leaks* and *overflows* in the continuous reservoir model. A reservoir is said to leak when it is emptied out at a fixed rate whenever it is not empty due to *e.g.* a hole or evaporation. It is said to overflow when anything that is put in the reservoir when it is full is lost due to *e.g.* the presence of an overflow pipe. To the best of our knowledge, we are the first ones to present an algorithmic solution to model and solve scheduling problems with leaks or overflows. In this paper, we try to give a theoretical and implementation-oriented view of the relationship between timetable constraints and function variables.

Several Mixed-Integer Programming (MIP) Models have been published for production planning and scheduling in plants with continuous reservoirs. The early model by Pinto and Grossmann [11] was improved by Alle and Pinto [1] who based their model on a formulation of the Travelling Salesperson Problem. Other recent models have been proposed by Ierapetritou and Floudas [5], Zhang and Sargent [16] and Mndez and Cerdá [9]. In these models, the profit is to be maximized.

Schwindt recently presented a branch-and-bound method for a scheduling problem involving discrete and continuous reservoirs in order to minimize a convex objective function in the start times of the activities [14]. In this algorithm, a convex program with linear constraints, which is a relaxation of the problem, is solved at each enumeration node. In a constraint-based approach similar to ours, Poder et al. [12] described a model in which the activities have a consumption profile that continuously vary with time. In particular, this model can be used to solve a producer/consumer problem which is a special case of our model. Let us finally observe that neither of these approaches deals with leak or overflow constraints.

In Section 2, we introduce the constrained data structure to maintain the lower and upper bounds for the level of the continuous reservoir in partially instantiated schedules. In Section 3, we present how the contents of this data structure are used to prune the search space. Leaks and overflows are presented in Section 4. Section 5 discusses some implementation issues and presents experimental results.

2 Data structure for bounding the level of the continuous reservoir

In branch-and-bound algorithms such as constraint-based solvers, the efficiency of the search heavily relies on the existence of algorithmic methods to quickly compute lower and upper bounds in a partially instantiated schedule. In this section, we present how *timetables* — widely used in constraint-based scheduling because of their low algorithmic complexity and because they ensure the resource capacity constraints are satisfied when the problem is fully instantiated [2, 8] — can be adapted for the continuous reservoir model.

In the model described here, we will consider the three following decision variables attached to any activity a: the start time variable $\mathbf{st}(a)$, or shortly $\mathbf{st}$ when the latter notation is not ambiguous, whose domain is represented by the interval of $\mathbb{N}$ $\{\mathbf{st}_{\min}(a), \cdots, \mathbf{st}_{\max}(a)\}$, or again $\{\mathbf{st}_{\min}, \cdots, \mathbf{st}_{\max}\}$, the completion time variable $\mathbf{ct}$ whose domain is $\{\mathbf{ct}_{\min}, \cdots, \mathbf{ct}_{\max}\}$ and the capacity variable $\mathbf{cap}$ whose domain is $\{\mathbf{cap}_{\min}, \cdots, \mathbf{cap}_{\max}\}$. If $\mathbf{cap}_{\min}$ is non-negative, the activity is said to be a *filling activity* and if $\mathbf{cap}_{\max}$ is non-positive we speak about an *emptying activity*. In this description of the model, an activity is not required to be a priori emptying or filling when the problem is defined, that is we can have $\mathbf{cap}_{\min} < 0 < \mathbf{cap}_{\max}$. Such an activity will be called an *emptying/filling* activity. Note that our model does not require the duration of the activity to be fixed so, the start time and the end time are a priori two unrelated decision variables but we at least have the constraint that the start time is not later than the completion time.

The aim of the *timetable constraint* is to determine time intervals in which a task cannot be processed or, on the contrary, some time points where it must be in process or where it must have been entirely processed. As any constraint in a constraint-based system, our continuous timetable constraint can be viewed as an algorithm that finds (in polynomial time) some properties on the decision variables of a problem by considering only the variables related to the constraint — that are, in our case, the decision variables $\mathbf{st}$, $\mathbf{ct}$, $\mathbf{cap}$ associated to the activities that require the continuous reservoir. The properties found must be valid whatever the other constraints of the problem are. In constraint programming terms, we call this the *filtering algorithm* of the constraint.

It is also assumed that no other activities than the activities given in input of the timetable, such as activities that could be dynamically added to the problem, can fill or empty the reservoir. In ILOG Scheduler terms, the reservoir is said to be *closed*. As a consequence, for a given continuous reservoir r, we can denote by n_r or shortly n the

number of activities that can fill or empty r. We will see in the remainder of the section that, when the reservoir is not closed, no propagation can be made because we are unable to estimate the behavior of the reservoir either by a lower bound or by an upper bound.

For the simplicity of the presentation, we first assume that all the constraints stating that "the activity a fills (or empties) the reservoir r" are true when the problem is defined. In other words, we assume that these constraints do not appear in *meta-constraints* such as :

$$\textbf{if } (\text{condition}) \textbf{ then } (a \text{ fills } r)$$

or

$$(a \text{ fills } r_1) \textbf{ XOR } (a \text{ fills } r_2)$$

Indeed, in the first constraint, the constraint "a fills r" may be false without violating the whole meta-constraint in which it appears. Similarly, either "a fills r_1" or "a fills r_2" may a priori be false in the second meta-constraint. Section 2.4 is devoted to the treatment of such meta-constraints.

2.1 Minimum and maximum contributions, minimum and maximum levels

The timetable of the continuous reservoir stores an upper bound and a lower on the level of the reservoir all along the time interval in which the activities are to be scheduled. Let us call them respectively *maximum* and *minimum level*.

In order to compute a maximum level, we will consider the *maximum individual contribution* of each activity that is an upper bound for the contribution of the activity at any time t. When $\mathbf{cap}_{\max}$ is positive, the maximum contribution corresponds to the contribution function when the activity produces its maximum capacity ($\mathbf{cap}_{\max}$) starting at its earliest start time and ending at its earliest completion time (see Figure 2-a-b). In absence of other information than the start and completion time variables of the activity, this contribution is clearly the best upper bound for all the possible contributions of the filling activity. When $\mathbf{cap}_{\max}$ is negative (see Figure 2-c), that is for an emptying activity, the contribution is maximum when the consumption is minimum, that is when it starts at latest and when the consumed capacity is equal to $\mathbf{cap}_{\max}$ (we have then $|\mathbf{cap}_{\max}| < |\mathbf{cap}_{\min}|$). We then define the *maximum total contribution* of the reservoir as the sum of all the individual contributions. By construction, the maximum total contribution is a possible maximum level for the reservoir. However, we can observe that the total

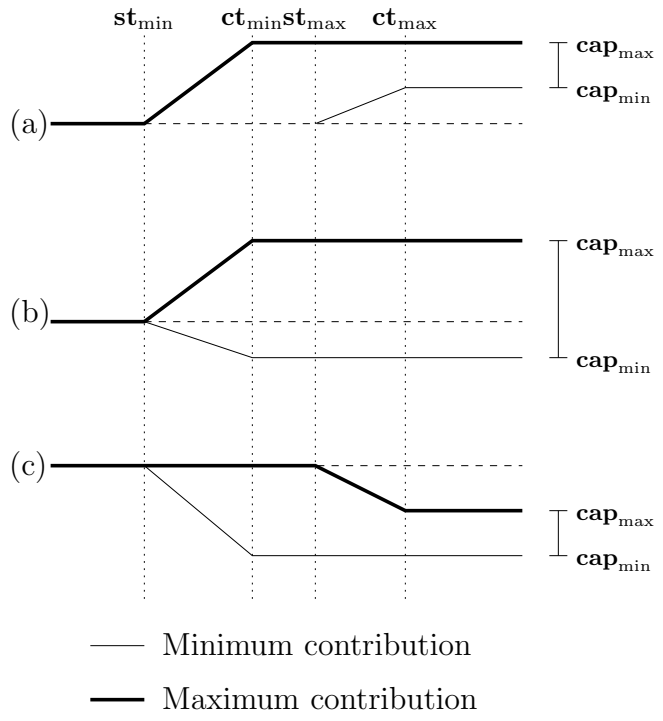

Figure 2: Minimum and maximum contribution for filling and emptying activities

contribution may be greater than the capacity of the reservoir at some point so that the best maximum level can be derived by taking the minimum (at each time point) between the maximum total contribution and the reservoir capacity.

Similarly, a *minimum level* of the continuous reservoir is a lower bound for the level of the reservoir. The *minimum contribution* of a filling activity is the contribution when the activity is scheduled at latest and the capacity is $\mathbf{cap}_{\min}$ (see Figure 2-a). The minimum contribution of an emptying or emptying/filling activity corresponds to the minimum capacity $\mathbf{cap}_{\min}$ consumed at earliest (see Figure 2-b-c). The *minimum total contribution* is the sum of the minimum (positive or negative) contributions of the n activities. Finally, we observe that the total contribution may be negative at some points so that the best minimum level is given by the maximum between the null function and the minimum total contribution.

Figure 2 illustrates the fact that the minimum and maximum individual contribution of each task is a piecewise linear function with at most 2 breakpoints so the minimum and maximum levels of the continuous reservoir are both piecewise linear functions with at most $2n$ breakpoints. We also recall that all these piecewise linear functions are defined

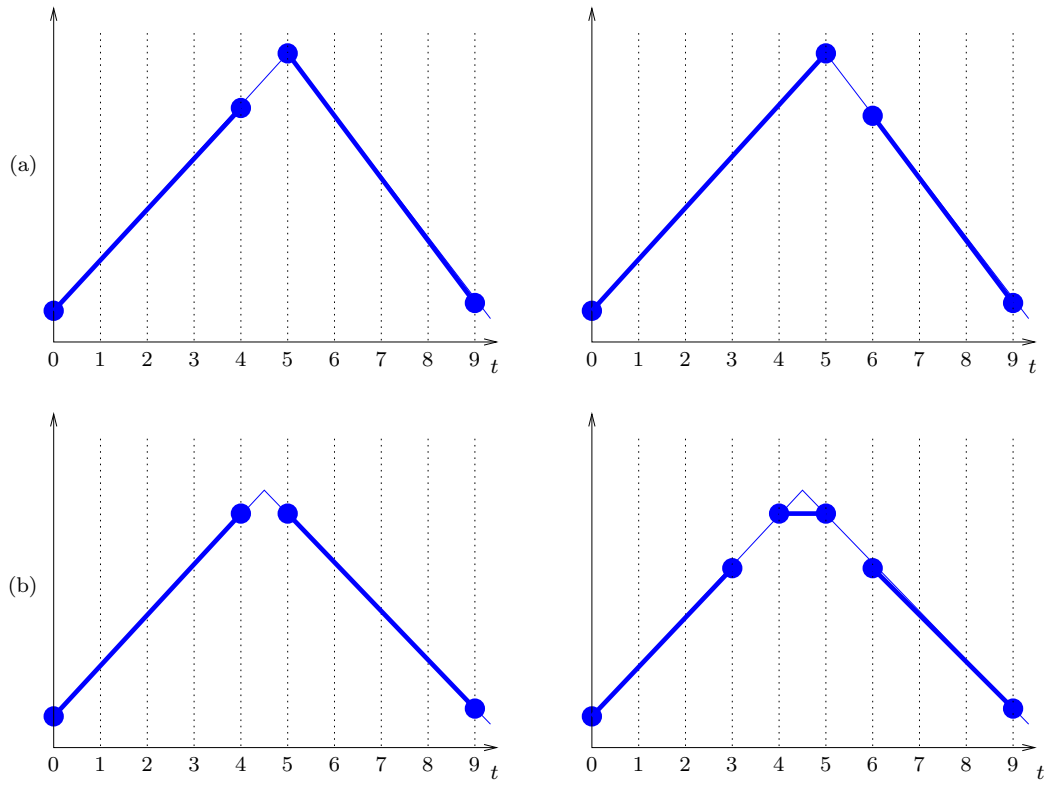

Figure 3: Segments representing a piecewise linear function. (a) The breakpoint abscissa of the function is integer. (b) The breakpoint abscissa is not integer.

on a bounded interval that corresponds to the scheduling time interval. So, each segment of the functions is bounded and has two endpoints.

The piecewise linear functions can be stored in a doubly-linked ordered list, each cell of the list corresponding to a segment of the function. Moreover, since in our model the time points are integer, all the breakpoints of the contribution curves are integer. This ensures that the length of each segment is at least 1. This is of key importance for the numerical stability of the algorithms using these functions because there is no segment with very small length. Indeed the slope of such small segments could not have been easily computed because it is very sensitive to numerical imprecision. However, we have just seen that the minimum (or maximum) level can result of the maximum (or minimum) function between two piecewise linear functions — namely, between the total contribution and the null function (or the constant function equal to the capacity of the reservoir). The resulting function is still piecewise linear but some breakpoints may have a non-integer

abscissa. In order to face this problem, the policy we choose to manage the piecewise linear functions was to represent a piecewise linear function f as an ordered list of segments — called the *segment representation* of f — such that :

- the abscissas x^i_{left} and x^i_{right} (with $x^i_{\text{left}} \leq x^i_{\text{right}}$) of both endpoints of each segment are integer;
- the left endpoint of the i^{th} segment is $(x^i_{\text{left}}, f(x^i_{\text{left}}))$ and its right endpoint is $(x^i_{\text{right}}, f(x^i_{\text{right}}))$ — $f(x^i_{\text{left}})$ and $f(x^i_{\text{right}})$ may be non-integer;
- the left abscissa of a segment is one greater than the right abscissa of its preceding segment, that is $x^{i+1}_{\text{left}} = x^i_{\text{right}} + 1$;
- two consecutive segments are non-colinear.

In other words, the segments representing f form a set packing of the points of f that have an integer abscissa. We can clearly observe with Figure 3 that this representation is not unique. In Figure 3-a, f is either represented by the segments $[(0, f(0)); (4, f(4)]$ and $[(5, f(5); (9, f(9))]$ on the left side of the figure or by the segments $[(0, f(0)); (5, f(5)]$ and $[(6, f(6); (9, f(9))]$ on the right side of the figure. We can even observe that the number of segments may change from a representation to another (Figure 3-b). However, we have the following result that shows that in any case the representation is efficient :

Lemma 1. *The piecewise linear function f is covered by a list of $O(\|f\|)$ segments where $\|f\|$ denotes the number of segments of f.*

Proof. Let us consider that f is linear on a time interval I and that there are 4 consecutive segments in its segment representation that have at least one endpoint in I. We clearly have that at least two consecutive segments have all their endpoints in I and must be linear because f is linear on I. This is a contradiction with the definition of the segment representation. So there are at most 3 segments in the segment representation that have at least one endpoint in I so that the length of the segment representation is at most $3\|f\|$. □

Figure 3 also shows that the segment representation does not represent f in any interval $(x^i_{\text{right}}, x^{i+1}_{\text{left}})$. It is not a problem since we have seen that the abscissas of the breakpoints of the level curve are integer in an instantiated solution. Therefore, in order to be sure that the capacity constraints are satisfied, it is sufficient to check them only at integer time points.

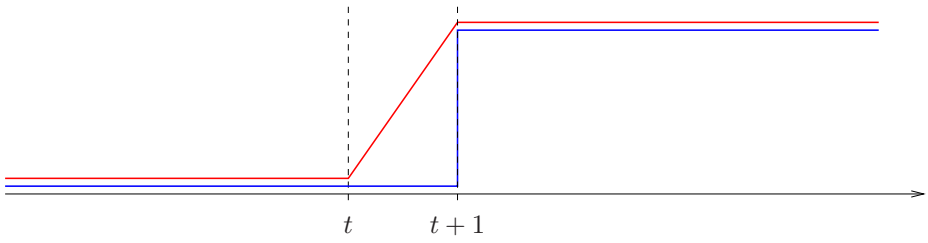

Figure 4: Instantaneous filling and unit duration filling

Remark. Since the capacity constraints are checked only at integer times, the contribution of an activity with a unit duration starting at time t is equivalent to the contribution of a null duration activity starting at time $t+1$ (see Figure 4). As far as the continuous reservoir model is concerned, both representations are equivalent.

2.2 Function variables

In order to give a compact formulation of the problem, it is interesting to introduce the so-called *function variables*. The function variables are decision variables that represents piecewise linear functions. For simplicity, we assume that the definition domain of a function variable is fixed when the variable is created. More formally, a function variable $\boldsymbol{f}$ is given as a 3-tuple $(D, \boldsymbol{f}_{\min}, \boldsymbol{f}_{\max})$ where D is the *definition domain*, that is a list of disjoint intervals sorted in increasing order, $\boldsymbol{f}_{\min}$ and $\boldsymbol{f}_{\max}$ are the *minimum and maximum* of $\boldsymbol{f}$. $\boldsymbol{f}_{\min}$ and $\boldsymbol{f}_{\max}$ are piecewise linear functions defined on D. The *(current) domain* of the possible values for the variable $\boldsymbol{f}$ are all the piecewise linear functions f such that $\boldsymbol{f}_{\min}(t) \leq f(t) \leq \boldsymbol{f}_{\max}(t)$ for any integer $t \in D$. $\boldsymbol{f}$ is said to be *bound* or *instantiated* when $\boldsymbol{f}_{\min}(t) = \boldsymbol{f}_{\max}(t)$ for any integer $t \in D$.

We can easily write simple usual constraints between function variables. For example, the propagation algorithm for the constraint $\boldsymbol{f} \leq \boldsymbol{g}$, meaning that for every t, $\boldsymbol{f}(t) \leq \boldsymbol{g}(t)$ for two functions variables $\boldsymbol{f}$ and $\boldsymbol{g}$ that have the same definition domain, can be written in a few lines. Indeed, when $\boldsymbol{f}_{\min}$ is modified, we just have to make the update $\boldsymbol{g}_{\min} \leftarrow \max(\boldsymbol{g}_{\min}, \boldsymbol{f}_{\min})$ and symmetrically, when $\boldsymbol{g}_{\max}$ is modified, we make the update $\boldsymbol{f}_{\max} \leftarrow \min(\boldsymbol{f}_{\max}, \boldsymbol{g}_{\max})$. We will also meet the constraint "$\boldsymbol{f}$ is non-increasing" in Section 4. This constraint is propagated by implementing the two following rules: each time that $\boldsymbol{f}_{\max}$ is modified at time t, we make the update $\boldsymbol{f}_{\max} \leftarrow \max(\boldsymbol{f}_{\max}, \boldsymbol{f}_{\max}(t))$ on the interval $D \cap [t, +\infty)$ and symmetrically, when $\boldsymbol{f}_{\min}$ is modified at t, $\boldsymbol{f}_{\min}$ must be similarly updated on $D \cap (-\infty, t]$. Obviously, these rules maintain both $\boldsymbol{f}_{\min}$ and $\boldsymbol{f}_{\max}$

non-increasing so that $\boldsymbol{f}$ is non-decreasing when it becomes instantiated.

According to what we have just presented in the first part of this section, the contribution of each activity a that fills or empties a reservoir corresponds to a function variable, denoted by $\boldsymbol{cb}(a)$. In the same way, the level of a continuous reservoir r is also a function variable denoted by $\boldsymbol{level}(r)$. With these new concepts, the resource constraint for a continuous reservoir r is simply formulated as the following functional equation :

$$\sum_{a \in r} \boldsymbol{cb}(a) = \boldsymbol{level}(r) \tag{1}$$

where "$a \in r$" means that the activity a fills or empties the reservoir r. The equality must be satisfied on the definition domain of the function variable $\boldsymbol{level}$. This definition domain is equal to the time interval on which we want the timetable constraint to be satisfied — in most cases, it is the whole scheduling horizon. When the function variable $\boldsymbol{level}(r)$ is created, its minimum is the null function and its maximum is the constant function equal to the capacity of the reservoir. Of course, it is also possible to initialize these minimum and maximum levels with any piecewise linear functions in order to model, for example, time-varying reservoir capacity and time-varying minimum level requirements.

In order to efficiently implement the propagation algorithms in Section 3, we will rely on the minimum and maximum total contributions defined in Section 2.1. For a continuous reservoir r, they can be represented by an auxiliary function variable $\boldsymbol{cb}(r)$. Following what we previously said, this new variable is defined by the constraint

$$\boldsymbol{cb}(r) = \sum_{a \in r} \boldsymbol{cb}(a) \tag{2}$$

which is in fact very similar to the resource constraint (1). However, by opposition to the variable $\boldsymbol{level}(r)$, we do not fix neither a minimum nor a maximum at the creation of $\boldsymbol{cb}(r)$. Furthermore, as $\boldsymbol{cb}(r)$ is an expression of the activity requirements and since it only appears in its definition constraint (2), we know that its domain will never be modified except by the modification of an individual contribution. Therefore, we only have to write a propagation algorithm to update $\boldsymbol{cb}(r)$ when some $\boldsymbol{cb}(a)$ is modified. The update procedure is presented in Section 2.3.

In order to have an efficient implementation, we will see in the rest of the paper that the individual contribution variables $\boldsymbol{cb}(a)$ have not to be explicitly stored in memory. However, they are very useful to describe the relationships between the activities and the reservoir. More importantly, we will see in Section 4 that they are the fair mean to model extra contributions that have an influence on the level of the reservoir and that cannot be

modeled by usual activities. Section 4 will focus on leaks and overflows. They can also be used to model a so-called *initial level* function that represent the effects of operations that are deliberately ignored in the model. For a simple example, if the level of the reservoir is non-null at time 0 due to scheduled operations, the initial level function will be the constant function equal to the value of the level of the reservoir at time 0 (which may be a constant or a variable).

Therefore, we will study the propagation algorithms for the continuous reservoir constraints (1) as the sum of function variables but, for each algorithm, we will observe that the algorithm can also be adapted to deal with activity contributions without using an explicit representation of the corresponding function variable.

2.3 Maintaining the total contribution

When an individual contribution function changes in equation (2), the minimum and/or the maximum total contribution of the reservoir have in general to be updated. When working with function variables, the procedure is as simple as if we would work with integer variables in the propagation engine. To update the minimum total contribution when the minimum of $\boldsymbol{cb}(a)$ $(a \in r)$ is modified, we do:

- Compute the delta function δ that is the difference between the value of $\boldsymbol{cb}_{\min}(a)$ before and after its modification. δ is clearly a piecewise linear function.
- Add δ to $\boldsymbol{cb}_{\min}(r)$.

Note that the function δ can be computed when the modification of the function variable is done, which avoids to compute it at the propagation time. The procedure to update $\boldsymbol{cb}_{\max}$ is of course symmetric.

We now observe that the computation of δ in the first step can be derived from the variables of the activity. We illustrate it by studying the case where the earliest start time of a filling activity a increases, that is $\boldsymbol{cb}_{\max}(a)$ is modified — and of course decreased. In what follows, all the variables are related to the activity a so, for shorter notations, we omit the reference. The amount by which the contribution $\boldsymbol{cb}_{\max}$ is decreased depends on the relative order of the old earliest start time (denoted by $\mathbf{st}^{\text{old}}_{\min}$), the new earliest start time ($\mathbf{st}_{\min}$) and the earliest completion time ($\mathbf{ct}_{\min}$). Figure 5(1) shows how the maximum contribution is modified when $\mathbf{st}^{\text{old}}_{\min} < \mathbf{st}_{\min} < \mathbf{ct}_{\min}$. Clearly, the maximum contribution of the activity is not modified before $\mathbf{st}^{\text{old}}_{\min}$ and after $\mathbf{ct}_{\min}$. At time point

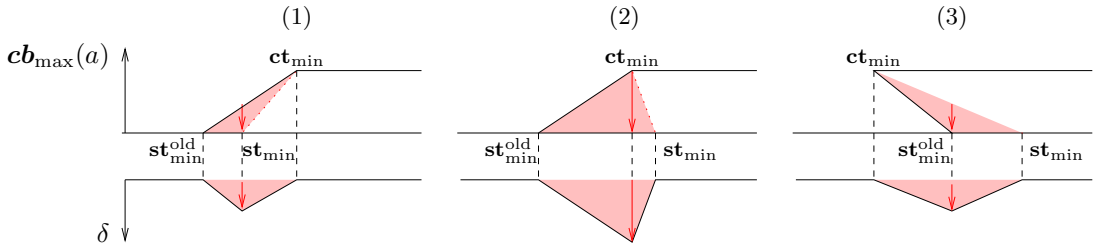

Figure 5: Updating the timetable when an earliest start time is modified

$\mathbf{st}_{\min}$, it is decreased by

$$\delta(\mathbf{st}_{\min}) = \frac{\mathbf{st}_{\min} - \mathbf{st}^{\text{old}}_{\min}}{\mathbf{ct}_{\min} - \mathbf{st}^{\text{old}}_{\min}} \times \mathbf{cap}_{\max}.$$

The modification of the level is linear between $\mathbf{st}^{\text{old}}_{\min}$ and $\mathbf{st}_{\min}$ and between $\mathbf{st}_{\min}$ and $\mathbf{ct}_{\min}$. Therefore, δ is this piecewise linear function with four segments.

Figure 5 also illustrates the case where $\mathbf{st}^{\text{old}}_{\min} \leq \mathbf{ct}_{\min} \leq \mathbf{st}_{\min}$ (second sub-figure) and the case where $\mathbf{ct}_{\min} < \mathbf{st}^{\text{old}}_{\min} < \mathbf{st}_{\min}$ (third sub-figure). In the second case, we observe that the start time becomes greater than or equal to the completion time : this will trigger the propagation algorithm of the constraint that ensures that $\mathbf{st} \leq \mathbf{ct}$. However, if another modification of $\mathbf{st}_{\min}$ happens before that this constraint is propagated, the ranking of $\mathbf{ct}_{\min}$, $\mathbf{st}^{\text{old}}_{\min}$ and $\mathbf{st}_{\min}$ will be as in the third case with $\mathbf{st}^{\text{old}}_{\min} > \mathbf{ct}_{\min}$. Figure 5 illustrates all the possible cases because we have by definition $\mathbf{st}^{\text{old}}_{\min} < \mathbf{st}_{\min}$. In any case, considering a vertical gap as a segment, the function δ has four segments.

It would not be interesting to detail what happens in all the possible cases when the minimum or maximum of **st**, **ct** and **cap** changes but we observe that the delta function to update the function variable $\boldsymbol{cb}(r)$ has always a constant number of segments that can be computed from the minimum and maximum values of the three variables attached to the activity. Moreover, it follows from what we have said in Section 2.1 that, independently of the number of updates it has undergone, the minimum and maximum of the function variable $\boldsymbol{cb}(r)$ have always $O(n)$ breakpoints. So, each update operation is done in $O(n)$ time but is faster in practice (typically in constant time) thanks to some cache policy.

2.4 Conditional filling/emptying constraints

When the filling or emptying constraint between an activity a and a continuous reservoir r appears in a meta-constraint, the individual contribution of the activity a is *conditional* (that means it can be null) because the filling/emptying constraint may be false.

Therefore, the definition of the individual contribution must be modified in order to take into account that the contribution can be null at any point. Therefore the maximum conditional contribution is $\max(0, \boldsymbol{cb}_{\max}(a))$ and the minimum conditional contribution is $\min(0, \boldsymbol{cb}_{\min}(a))$.

There is no difficulty into adapting the algorithm to update the total contribution when there are conditional contributions. An additional update procedure must be added to propagate the property that a conditional filling/emptying constraint becomes true. In this propagation procedure, the function δ corresponds to

- $\boldsymbol{cb}_{\min}(a)$ when updating $\boldsymbol{cb}_{\min}(r)$ when $\mathbf{cap}_{\min}(a)$ becomes greater than 0 or
- $\boldsymbol{cb}_{\max}(a)$ when updating $\boldsymbol{cb}_{\max}(r)$ when $\mathbf{cap}_{\max}(a)$ becomes negative.

Symmetrically, another procedure must propagate the possibility for a conditional filling/emptying constraint to become false. In this case, the maximum total contribution is decreased by $\boldsymbol{cb}_{\max}(a)$ and the minimum total contribution is increased by $\boldsymbol{cb}_{\min}(a)$.

3 Propagation rules

The minimum and maximum levels of the continuous reservoir are used to limit the search space. First of all, if the minimum level is greater than the maximum level at some time point, there is of course no possible schedule satisfying the current partial assignment and the search fails.

The minimum and maximum levels can also be used to reduce the domain of the possible values for the function variables representing the individual contributions of the activities. From these deductions, we can directly derive some domain reductions for the variables **st**, **ct** and **cap** of the corresponding activity.

In order to obtain the first propagation rule, we replace, in the expression of the maximum total contribution, the individual contribution $\boldsymbol{cb}_{\max}(a)$ by the function variable $\boldsymbol{cb}(a)$. Clearly, once the function variable is instantiated, the expression $\boldsymbol{cb}_{\max}(r) - \boldsymbol{cb}_{\max}(a) + \boldsymbol{cb}(a)$ is an upper bound for the final level of the reservoir so that we must have :

$$\boldsymbol{level}_{\min}(r) \leq \boldsymbol{cb}_{\max}(r) - \boldsymbol{cb}_{\max}(a) + \boldsymbol{cb}(a)$$

that is

$$\boldsymbol{cb}(a) \geq \boldsymbol{level}_{\min}(r) - \boldsymbol{cb}_{\max}(r) + \boldsymbol{cb}_{\max}(a) \tag{3}$$

This inequality gives a rule to update the minimum contribution of a. Namely, by denoting the piecewise linear function on the right side of the above expression by $\kappa_{\min}(r,a)$, the rule is:

$$\boldsymbol{cb}_{\min}(a) \leftarrow \max\left(\boldsymbol{cb}_{\min}(a), \kappa_{\min}(r,a)\right)$$

Symmetrically, we have the following inequality and rule to update $\boldsymbol{cb}_{\max}(a)$:

$$\begin{array}{rcl} \boldsymbol{cb}(a) & \leq & \kappa_{\max}(r,a) = \boldsymbol{level}_{\max}(r) - \boldsymbol{cb}_{\min}(r) + \boldsymbol{cb}_{\min}(a) \\ \boldsymbol{cb}_{\max}(a) & \leftarrow & \min\left(\boldsymbol{cb}_{\max}(a), \kappa_{\max}(r,a)\right) \end{array} \quad (4)$$

We observe that if, after an update, we have $\boldsymbol{cb}_{\min}(a)(t) > 0$ or $\boldsymbol{cb}_{\max}(a)(t) < 0$ at some time point t, then the contribution becomes sure, either filling or emptying. This information must be propagated, if the filling/emptying constraint appears in a meta-constraint.

When we know that $\boldsymbol{cb}(a)$ is the contribution of a filling or emptying activity, we can directly interpret the functions $\kappa_{\min}(r,a)$ and $\kappa_{\max}(r,a)$ in order to directly update the variables $\mathbf{st}(a)$, $\mathbf{ct}(a)$ and $\mathbf{cap}(a)$ without explicitly computing $\boldsymbol{cb}(a)$. When considering the variable $\mathbf{cap}(a)$, we clearly have by (3) and (4):

$$\begin{array}{rcl} \mathbf{cap}_{\min}(a) & \leftarrow & \max\left(\mathbf{cap}_{\min}(a), \max\limits_{t}\left(\kappa_{\min}(r,a)(t)\right)\right) \\ \mathbf{cap}_{\max}(a) & \leftarrow & \min\left(\mathbf{cap}_{\max}(a), \min\limits_{t}\left(\kappa_{\max}(r,a)(t)\right)\right) \end{array}$$

Here again, $\mathbf{cap}_{\min}(a) > 0$ or $\mathbf{cap}_{\max}(a) < 0$ means that the filling or emptying constraint associated to the activity a is true. The start time and completion time of a must satisfy

$$\begin{array}{rcccl} \kappa_{\min}(r,a)(t) > 0 & \Rightarrow & \mathbf{st}(a) \leq t & \Rightarrow & \mathbf{st}_{\max}(a) \leftarrow \min(\mathbf{st}_{\max}(a), t) \\ \kappa_{\max}(r,a)(t) < 0 & \Rightarrow & \mathbf{st}(a) \leq t & \Rightarrow & \mathbf{st}_{\max}(a) \leftarrow \min(\mathbf{st}_{\max}(a), t) \\ 0 > \kappa_{\min}(r,a)(t) > \mathbf{cap}_{\max}(a) & \Rightarrow & \mathbf{ct}(a) \geq t & \Rightarrow & \mathbf{ct}_{\min}(a) \leftarrow \max(\mathbf{ct}_{\min}(a), t) \\ 0 < \kappa_{\max}(r,a)(t) < \mathbf{cap}_{\min}(a) & \Rightarrow & \mathbf{ct}(a) \geq t & \Rightarrow & \mathbf{ct}_{\min}(a) \leftarrow \max(\mathbf{ct}_{\min}(a), t) \end{array}$$

These four propagation rules can be improved when we can get information on the filling or emptying rate. For example, if the processing time of the activity a is given by the variable $\mathbf{pt}(a)$, the minimum filling rate (assuming $\mathbf{cap}_{\min}(a) > 0$) is given by $\mathbf{rate}_{\min}(a) = \mathbf{cap}_{\min}(a)/\mathbf{pt}_{\max}(a)$ while the maximum rate is $\mathbf{rate}_{\max}(a) = \mathbf{cap}_{\max}(a)/\mathbf{pt}_{\min}(a)$. Also the minimum and maximum rate can eventually be set by external constraints, such as the *slope constraint* implemented in ILOG Scheduler that constrains the ratio between the two integers **cap** and **pt** to be "not too far" from a given real value (more details can be found in [6]).

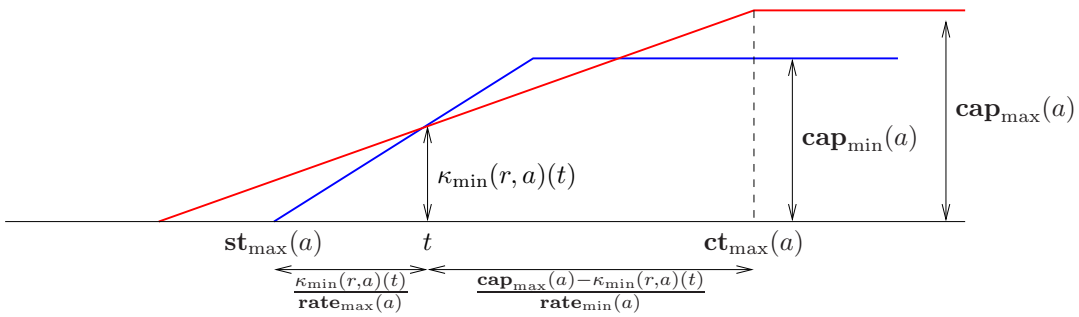

Figure 6: Using filling rate to update $\mathbf{st}_{\max}(a)$ and $\mathbf{ct}_{\max}(a)$

We now present how the rate is used on the first propagation rule for $\mathbf{st}_{\max}(a)$, namely $\kappa_{\min}(r,a)(t) > 0 \Rightarrow \mathbf{st}_{\max}(a) = \min(\mathbf{st}_{\max}(a), t)$. The condition means that the contribution of a must be at least $\kappa_{\min}(r,a)(t)$ at time point t. Given the filling rates, the shortest time for the activity a to produce the contribution $\kappa_{\min}(r,a)(t)$ is $\kappa_{\min}(r,a)(t)/\mathbf{rate}_{\max}(a)$ (see Figure 6). So, a cannot start after $t - \kappa_{\min}(r,a)(t)/\mathbf{rate}_{\max}(a)$, which improves the propagation rule for $\mathbf{st}_{\max}(a)$. Moreover, we can also derive a propagation rule to update $\mathbf{ct}_{\max}(a)$ by observing that a can at most produce the contribution $\mathbf{cap}_{\max}(a) - \kappa_{\min}(r,a)(t)$, which take at most the time $(\mathbf{cap}_{\max}(a) - \kappa_{\min}(r,a)(t))/\mathbf{rate}_{\min}(a)$. So $\mathbf{ct}_{\max}(a)$ is at most $t + (\mathbf{cap}_{\max}(a) - \kappa_{\min}(r,a)(t))/\mathbf{rate}_{\min}(a)$ (see Figure 6). Obviously, similar improvements can be achieved for the other propagation rules for $\mathbf{st}$ and $\mathbf{ct}$ as well.

In an algorithmic view, all the above rules are to be checked for each one of the $O(n)$ segments of $\kappa_{\min}(r,a)$ and $\kappa_{\max}(r,a)$. More precisely, for each segment of $\kappa_{\min}(r,a)$ (and symmetrically for each segment of $\kappa_{\max}(r,a)$), we check whether there is some t such that either $\kappa_{\min}(r,a)(t) = 0$ or $\kappa_{\min}(r,a)(t) = \mathbf{cap}_{\max}(a)$ and, according the rule, $\mathbf{st}_{\max}(a)$ or $\mathbf{ct}_{\min}(a)$ is updated. If we know $\mathbf{rate}_{\min}$ or $\mathbf{rate}_{\max}$, we also have to check the "improved rule" for the endpoints of each segment. Therefore, the update for an activity a is done in $O(n)$ time so that the complete update for all the activities takes $O(n^2)$ time in the worst case. However, it is faster in practice thanks to several code optimization tricks. We can for example keep in memory the intervals on which $\kappa_{\min}$ and $\kappa_{\max}$ are modified to avoid to scan the whole time horizon.

Finally, we remark that the time complexity for the continuous reservoir timetable is the same as for the discrete reservoir and capacity resource timetables.

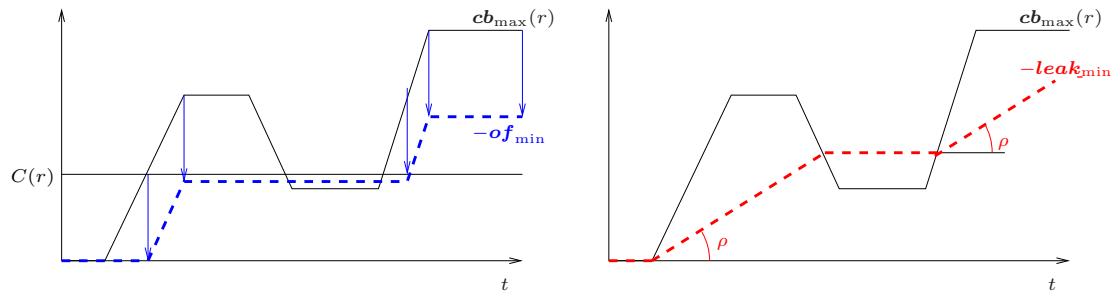

Figure 7: Building the leak and the overflow contribution from the total contribution

4 Leaks and overflows

With the function variables presented in Section 2.2, it becomes easy to model phenomena that have an influence on the level of the reservoir but that cannot be modeled only by classical activities. As an illustration, we present how to model a continuous reservoir that may leak or overflow.

A reservoir *leaks* when it is emptied out at a fixed rate whenever it is not empty. This of course models a container with a hole or an evaporating liquid. A reservoir *overflows* when it can be filled by more than its remaining capacity but definitely loses what is put in the reservoir when it is full. This models a reservoir with an overflow pipe but also the filling of a battery : a full battery can receive energy even if this energy cannot be stored and is lost. Leaks and overflows cannot be easily modeled by activities because they depend on the level of the reservoir.

The contributions of both leak and overflow are *negative* and *non-increasing* since what is vanished is never recovered. We are going to see that the contributions are piecewise linear so that they can be represented by function variables, respectively denoted by $\boldsymbol{leak}(r)$ and $\boldsymbol{of}(r)$ (or simply $\boldsymbol{leak}$ and $\boldsymbol{of}$). So, for a continuous reservoir r, we have the functional equation $\boldsymbol{cb}(r) = \sum_{a \in r} \boldsymbol{cb}(a) + \boldsymbol{leak}(r) + \boldsymbol{of}(r)$. Eventually, we are going to derive upper and lower bounds for both $\boldsymbol{leak}$ and $\boldsymbol{of}$ from the other activity contributions.

4.1 Overflow

We first present how to estimate the overflow of a reservoir r of capacity $C(r)$. For the simplicity of the presentation, we imagine that the reservoir r cannot leak so that the function $\boldsymbol{cb}_{\max}(r) = \sum_{a \in r} \boldsymbol{cb}(a)$ represents the sum of all the individual contributions but the overflow. If r leaked, $\boldsymbol{leak}$ would have to be added to this sum. We imagine that the

reservoir r is enclosed in a reservoir R with an infinite capacity so that what is lost by r stays in the reservoir R. r can be filled by an activity but R cannot. Therefore, the level of R at time t is at most $\boldsymbol{cb}_{\max}(R)(t) = \max_{t' \leq t}(\boldsymbol{cb}_{\max}(r)(t'))$. Since the overflow is possible only when r is full, that is only when $\boldsymbol{cb}_{\max}(r)(t') > C(r)$, we have

$$-\boldsymbol{of}(t) \leq \max\left(0, \max_{t' \leq t}(\boldsymbol{cb}_{\max}(r)(t') - C(r)\right).$$

We recall that $-\boldsymbol{of}$ is the absolute value of the overflow since the function is negative. So, a rule to update $\boldsymbol{of}_{\min}$ is directly derived from the above inequality. Symmetrically, $\boldsymbol{of}_{\max}$ can be updated by noting that $-\boldsymbol{of}(t) \geq \max\left(0, \max_{t' \leq t}(\boldsymbol{cb}_{\min}(r)(t')) - C(r)\right)$. Clearly, both $\boldsymbol{of}_{\min}$ and $\boldsymbol{of}_{\max}$ are piecewise linear functions, which have at most the number of segments of $\boldsymbol{cb}_{\max}(r)$ and $\boldsymbol{cb}_{\min}(r)$ respectively [15]. The construction of $\boldsymbol{of}_{\max}$ is illustrated in Figure 7. The function variable $\boldsymbol{of}$ is instantiated as soon as $\boldsymbol{cb}(r)$ is instantiated, which is clearly achieved when all the activities are scheduled.

4.2 Leak

We now study the leak contribution of a reservoir r. For simplicity again, we now assume that r cannot overflow and we assume that the leak rate is a constant denoted by ρ. The leak rate is equal to ρ whenever the reservoir is not empty that is $\boldsymbol{cb}(r)(t) + \boldsymbol{leak}(t) > 0$. When the reservoir stays empty for a time interval, it means that either the contribution is null for this interval or that the slope of the contribution is less than ρ, which means that anything added to the reservoir is lost because of the leak. As a consequence, we get a lower bound on the strongest leak $\boldsymbol{leak}_{\min}$, by building the piecewise linear function such that $\boldsymbol{leak}_{\min}(0) = 0$, and

$$\frac{\mathrm{d}\boldsymbol{leak}_{\min}}{\mathrm{d}t}(t) = \begin{cases} -\rho & \text{if } -\boldsymbol{leak}_{\min}(t) < \boldsymbol{cb}_{\max}(r)(t) \\ -\min\left(\rho, \frac{\mathrm{d}\boldsymbol{cb}_{\max}(r)}{\mathrm{d}t}(t)\right) & \text{if } -\boldsymbol{leak}_{\min}(t) = \boldsymbol{cb}_{\max}(r)(t) \\ 0 & \text{otherwise} \end{cases}$$

Note that since $\boldsymbol{leak}_{\min}$ and $\boldsymbol{leak}_{\max}$ are not differentiable at their breakpoints, the derivative should be understood as derivative from the right in the above equations as well as in the following. The construction of $\boldsymbol{leak}_{\min}$ is illustrated in Figure 7. The computation of an upper bound on the leak is once again symmetric since the leak is mandatory when $\boldsymbol{cb}_{\min}(r)(t) + \boldsymbol{leak}(t) > 0$. Accordingly, it follows from the construction

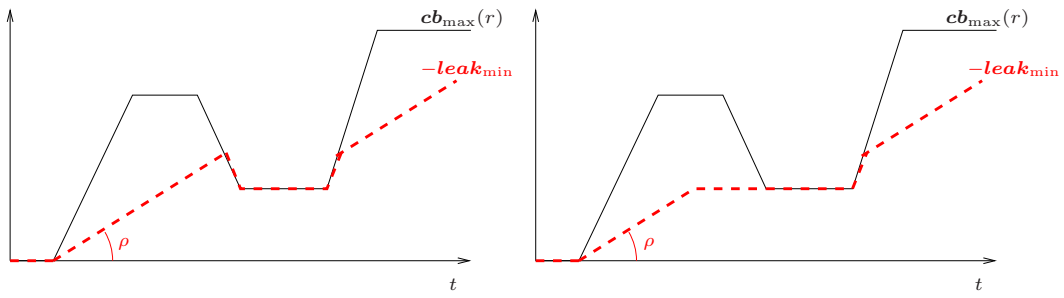

Figure 8: Improving $\boldsymbol{leak}_{\min}$

of $\boldsymbol{leak}_{\max}$ that $\boldsymbol{leak}_{\max}(0) = 0$ and

$$\frac{\mathrm{d}\boldsymbol{leak}_{\max}}{\mathrm{d}t}(t) = \begin{cases} -\rho & \text{if } -\boldsymbol{leak}_{\max}(t) < \boldsymbol{cb}_{\min}(r)(t) \\ -\min\left(\rho, \frac{\mathrm{d}\boldsymbol{cb}_{\min}(r)}{\mathrm{d}t}(t)\right) & \text{if } -\boldsymbol{leak}_{\max}(t) = \boldsymbol{cb}_{\min}(r)(t) \\ 0 & \text{otherwise} \end{cases}$$

As for the overflow, the function variable $\boldsymbol{leak}$ is instantiated as soon as $\boldsymbol{cb}(r)$ is instantiated. We also observe that $\boldsymbol{leak}_{\min}$ can be improved: for example, we should have that, at any time t, $-\boldsymbol{leak}_{\min}(t) \leq \boldsymbol{cb}_{\max}(r)(t)$, which is not satisfied in Figure 7. However, we observe that this inequality is the rewriting of (3) in presence of a leak (with $\boldsymbol{level}_{\min}(r) = 0$). When this constraint is propagated (see the left side of Figure 8 for our example), $\boldsymbol{leak}_{\min}$ may not be non-increasing anymore. By propagating the constraint that $\boldsymbol{leak}$ is non-increasing (see Section 2.2), we still improve the lower bound on the leak contribution (right side of Figure 8).

5 Implementation and tests

The continuous reservoir model is implemented in ILOG Scheduler 5.2 [6]. We introduced a new class named `IloContinuousReservoir` that inherits the capacity resource class (`IloCapResource`) just like the discrete reservoir class (`IloReservoir`). We remark that the member functions of these classes are very similar. The main difference is that there is no "*time extent*" in the resource constraints between activities and continuous reservoir. The time extent is a parameter that specifies that the instantaneous filling (or emptying) process is done at the start or at the completion of the activity. It can also specify that the activity fills at its start and empties the same quantity at its completion. Since both the start and completion time play a role in the semantic of a continuous filling process, time

extent cannot be adapted to the continuous reservoir model. In order to model a filling followed by an emptying, we only have to define two activities linked by a precedence relation. In ILOG Scheduler 5.2, function variables, leaks and overflows are implemented but are not officially documented.

We also mention a difference between the model presented here and the implementation in ILOG Scheduler. The activities are required to be either filling or emptying when they are defined, that is the initial domain of **cap** cannot have $\mathbf{cap}_{\min} < 0$ and $\mathbf{cap}_{\max} > 0$. In real world problems, such emptying/filling activities are very rare. Eventually, they could be modeled by a pair of activities, the one filling the other emptying, sharing the same start and completion time variables.

Our implementation was mainly tested on some problems provided by ILOG customers. These tests cannot be presented here because some confidential information cannot be disclosed. However, we briefly describe some technical tests. First, in order to test the correctness of the implementation, we generated dozens of random instances with one or several continuous reservoirs. In these instances there were between 5 and 10 activities. Processing times were chosen between 5 and 20 and the consumption/production were chosen between -10 and 10. We also created instances in which there is one An accurate test for the numerical stability was

1. to "manually" build a feasible solution by instantiating the start time, completion time and capacity for each activity,

2. to create an instance in which the level of the reservoir is required to be "nearly" equal to the level of the solution manually built and the start/completion time variables of the activities (and eventually the capacity variable) are not constrained. This instance has clearly a feasible solution.

3. to solve the instance created in step 2 and check that a solution is found. In order to find a solution, we used the default search goal implemented in ILOG Solver which tries each possible value for each variable.

Since the size of the instances is small, the computational times was short (less than 1 second on a PC 1GHz) for instances in which there is one reservoir and in which capacity variable of each activity is fixed. However, computation times clearly grow when the domains of the capacity variables become larger as well as when there are more machines.

In order to test the efficiency of the implementation of the timetable constraint for the continuous reservoirs, we generated instances that can be either modeled by a discrete

ℓ	Makespan
0	58
1	57
2...14	55
15...20	Infeasible

Table 1: Makespan of a 6×6 job-shop in function of the initial level

or a continuous reservoir. In such instances, all filling and emptying activities have null duration, that is $\mathbf{ct} = \mathbf{st}$. We observed that the results obtained with the continuous reservoir coincided well with the results obtained with the discrete model (in fact, the search trees coincided too) and that the overcomputation time when using the continuous model versus the discrete model is about 50%. This performance ratio is quite satisfactory given that the continuous timetable the update procedures are indeed more complex. In order to improve this ratio, we could write special propagation procedures for activities with null duration : it would be interesting to solve mixed models with both instantaneous and non-instantaneous filling/emptying activities.

We also tested the timetable constraint on "academic" optimization problems. Namely, we created variants of jobshop instants derived from the famous Fisher and Thompson's instances [4] in which each activity can fill or empty a single common reservoir. The production or consumption quantity of each activity was randomly between -2 and 2. The other important parameters of the instances are the capacity of the continuous reservoir and its initial level, which is a constant $\ell \in [0, C(r)]$ that represents the level of the reservoir at time 0, before any operation of the instant starts. Let Q be the algebraic sum of all the production/consumption quantities. Two necessary conditions for an instance to be feasible are that $\ell + Q \geq 0$ and $\ell + Q \leq C(r)$. Clearly, when the capacity of the reservoir is large enough and the initial level is neither too low nor too high, the presence of the reservoir does not practically constrain the problem and the problem is not more difficult than the job-shop problem without reservoir. In contrast, experimental tests show that, when $\ell \approx 0$ or $\ell \approx C(r)$ or $\ell + Q \approx 0$ or $\ell + Q \approx C(r)$ (especially when $|Q|$ is smaller), instances may be more difficult to solve, which means that the reservoir constraint is active.

Table 1 presents computational results for 21 variants of the instance MT06 in presence of a continuous reservoir with capacity 20 and the initial level ℓ varying in $\{0, 1, \cdots, 20\}$.

As the algebric sum Q is equal to 5, the instances with $\ell \geq 16$ are clearly infeasible. However, we also observed that the instance $\ell = 15$ is also infeasible. For low initial level ($\ell = 0, 1$), the problem is feasible but the makespan is increased. For intermediate initial levels ($2 \leq \ell \leq 14$), the makespan is equal to the minimal makespan for the problem without reservoir constraint. Computation times are less than 1 second for each instance. We also try to solve a variant of MT10 with an initial level $\ell = 0$. We found a makespan equal to 962 (the makespan without reservoir is 930) after about 10 hours of CPU time.

We can finally observe that a pure Constraint Programming (CP) approach is unlikely to compete with the MIP models presented in the literature [1, 9] because the problems studied in these papers have to deal with capacity planning, inventory management issues and the maximization of a sum criterion, for which MIP is a method of choice. It would be interesting to test the behavior of a hybrid combination of CP and MIP on these problems, as it seems that such a combination may be efficient (*e.g.* [13]).

6 Conclusion

This paper has introduced a new class of resource in constrained-based scheduling : the continuous reservoir. This model has a real practical relevance since several users of ILOG Scheduler asked for it. The timetable constraint associated to the continuous reservoir is an extension of the timetable constraint for discrete resources. By introducing function variables in this paper, we have presented the timetable constraint in a very general framework. In practice, this framework is useful because it enables the modeling of contributions that cannot simply be represented by activities but only by function variables subject to complex constraints, as for example the leaks and overflows.

Future research and implementation will take an interest in adapting the *balance* constraint for continuous reservoirs [8]. The balance constraint focuses on the precedence relations between activities in order to derive better update rules. In ILOG Scheduler 5.2, the balance constraint is only implemented for discrete reservoirs (and for discrete resources).

Acknowledgments

The work described in this paper was done when the first author worked for ILOG Inc. The authors would like to thank Filippo Foccaci and Wim Nuijten for their help for the implementation and test of the continuous reservoir model in ILOG Scheduler 5.2. They

are also indebted to the anonymous referees for the references about MIP models and for their constructive remarks.

References

[1] A. Alle and J.M. Pinto. Mixed-integer programming models for the scheduling and operational optimization of multiproduct continuous plants. *Industrial & Engineering Chemistry Research*, 41:2689–2704, 2002.

[2] Ph. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*. Kluwer Academic Publishers, Norwell, Massachusetts, USA, 2001.

[3] Peter Brucker. Scheduling and constraint propagation. *Discrete Applied Mathematics*, 123:227–256, 2002.

[4] H. Fisher and G.L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In J.F. Muth and G.L. Thompson, editors, *Industrial Scheduling*. Prentice Hall, 1963.

[5] M.G. Ierapetritou and C. Floudas. Effective continuous-time formulation for short-term scheduling. 2. continuous and semicontinuous processes. *Industrial and Engineering Chemistry Research*, 37:4360–4374, 1998.

[6] ILOG Inc. *ILOG Scheduler 5.2 User's Manual and Reference Manual*, December 2001.

[7] ILOG Inc. *ILOG Solver 5.2 User's Manual and Reference Manual*, December 2001.

[8] Ph. Laborie. Algorithms for propagating resource constraints in AI Planning and Scheduling: Existing approaches and new results. *Artificial Intelligence*, 143:151–188, 2003.

[9] C.A. Méndez and J. Cerdá. An efficient MILP continuous-time formulation for short-term scheduling of multiproduct continuous facilities. *Computers and Chemical Engineering*, 26:687–695, 2002.

[10] K. Neumann and C. Schwindt. Project scheduling with inventory constraints. *Mathematical Methods of Operations Research*, 56:513–533, 2002.

[11] J.M. Pinto and I.E. Grossmann. Optimal cyclic scheduling of multistage multiproduct plants. *Computers and Chemical Engineering*, 18:797–816, 1994.

[12] E. Poder, N. Beldiceanu, and E. Sanlaville. Computing a lower approximation of the compulsory part of a task with varying duration and varying resource consumption. *European Journal of Operational Research*, 2003. In press.

[13] R. Rodosek, M. Wallace, and M. Hajian. A new approach to integrating mixed integer programming and constraint logic programming. *Annals of Operations Research*, 86:63–87, 1999.

[14] C. Schwindt. Scheduling of continuous production in process industries. In *Proceedings of the 8th International Workshop on Project Management and Scheduling (PMS 2002)*, pages 314–317, 2002.

[15] F. Sourd. Scheduling a sequence of tasks with general completion costs. Research report 2002/013, LIP6, 2002.

[16] X. Zhang and R.W.H. Sargent. The optimal operation of mixed production facilities-extensions and improvements. *Computers and Chemical Engineering*, 22:1287–1295, 1998.