Implementing Cumulative Functions for Conditional Task Intervals using a Generalized Cumulative Constraint

Pierre Schaus
UCLouvain, INGI [Place Sainte Barbe 2/L5.02.01, 1348 Louvain-la-Neuve], Belgium

Roger Kameugne
Université de Maroua, Faculté des sciences [P.O. Box 814, Maroua, Cameroon], Cameroon
UCLouvain, INGI [Place Sainte Barbe 2/L5.02.01, 1348 Louvain-la-Neuve], Belgium

Charles Thomas
UCLouvain, INGI [Place Sainte Barbe 2/L5.02.01, 1348 Louvain-la-Neuve], Belgium

Abstract

Cumulative functions were introduced in [3] as a mean to express and construct consumption profiles over conditional task intervals [2]. The only two solvers that fully support their use are IBM CP-Optimizer [4] and Optal-CP [6]. Although their usage, construction, and semantics have been thoroughly described in [3], we have not found any explanations on how to practically implement these in a CP Solver. We explain how a capacity range constraint on a cumulative function expression can be translated into a generalized cumulative constraint that accommodates optional activities and negative consumption. We propose a simple traversal of the Abstract Syntax Trees (AST) representing the expression to create a flattened set of activities. This approach can be implemented in any constraint programming solver without the need for developing a specific global constraint that would be aware of the cumulative function expression. We demonstrate how to use this implementation on a simple scheduling example.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming; Computing methodologies → Planning and scheduling

Keywords and phrases Constraint Programming, Scheduling, Cumulative Function

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

A conditional interval variable [2] is a decision variable used to model optional tasks in constraint-based scheduling problems. An interval variable x is characterized by a start s, an end e, a duration d and a status indicating if the interval is absent (denoted $x = \perp$) or present. Its domain can be expressed as a subset of $\{\perp\} \cup \{[s, e) \mid s \leq e, s, e \in \mathbb{Z}\}$. The duration d links to the other attributes s and e by the relation $d = e - s$. An interval variable x is fixed if $x = \perp$ or $x = [s, e)$ (the interval starts at s and ends at e).

A cumulative function [3] is a step-wise integer function of time used to model the cumulative contribution of a set of conditional interval variables. The contribution of a single interval variable is represented by an elementary cumulative function receiving an interval x and a non-negative height value h or a non-negative height range $[h_{min}, h_{max}]$. The existing elementary functions introduced in [3] are represented on Figure 1.

Cumulative functions (elementary or not) can then be combined with the `plus/minus` operators to form a new cumulative function representing the addition or subtraction of two functions. A cumulative function is thus an Abstract Syntax Tree (AST) with elementary function at the leaf-nodes. The height of a cumulative function can then be constrained to be within a range interval every time where at least one task executes. This is done with the $alwaysIn(f, minCapa, maxCapa)$ construct.

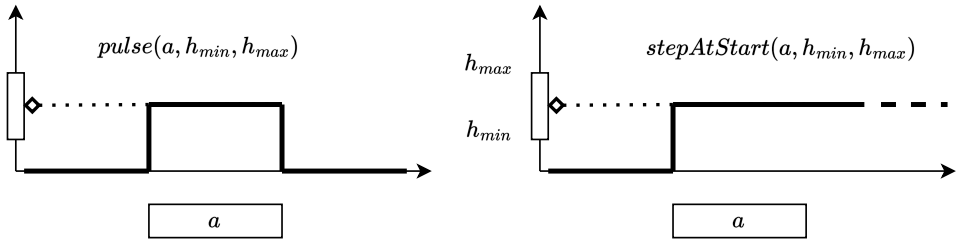

Figure 1 Elementary cumulative functions. A similar function exist for `stepAtEnd`.

2 Implementation

In our implementation based on MiniCP [5], each node of the AST contains a non-negative height variable denoted by h. These heights are created when elementary functions are created but do not exists in internal nodes of the AST.

Whenever an $alwaysIn(f, minCapa, maxCapa)$ constraint is posted, the tree is traversed to collect all the task intervals from the elementary functions at the leaf nodes. Each of these intervals is transformed into a cumulative activity. The cumulative activity's height is the height of the task interval at the leaf-node, possibly multiplied by $+1$ or -1, depending on whether it is involved in an even or an odd number of times as the right member of a minus expression down its path from the root of the AST.

There is a subtlety concerning step elementary functions. These activities are not the ones of the original task interval but rather defined on new a task interval that finishes at $+\infty$ with its start synchronized with the original task interval.

The resulting cumulative function can then be subjected to constraints such as resource capacity constraints. To the best of our knowledge only one constraint of the literature supports optional activities, with possibly negative consumption's and min/max capacities. This is the Generalized Cumulative constraint of [1] that we use.

▶ **Example 1.** Let us consider three task intervals: A, B and C that contribute to a cumulative function $f = minus(stepAtStart(A, 2), plus(pulse(B, 1), stepAtEnd(C, 1)))$ The resulting AST and cumulative function are represented in Figure 2. On the right, the solid parts represent the time windows of the tasks while the hatched parts represent their contribution to the profile. Note that while B is added with a pulse function and thus contributes only during its time window, both A and C are added with step functions. Therefore, the actual activities created are A' and C' that execute until the time horizon (vertical dashed line). The final cumulative function f is represented by a dotted line. Assuming a $alwaysIn(f, minCapa, maxCapa)$ constraint is posted, the cumulative activities passed to the Generalized Cumulative constraint of [1] are A', B, C' having respectively a height of $2, -1, -1$.

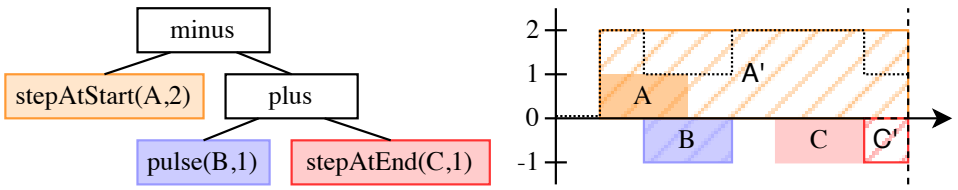

Figure 2 Example 1: AST (left) and cumulative function (right)

References

1 Nicolas Beldiceanu and Mats Carlsson. A new multi-resource cumulatives constraint with negative heights. In *CP*, volume 2470 of *Lecture Notes in Computer Science*, pages 63–79. Springer, 2002.

2 Philippe Laborie and Jerome Rogerie. Reasoning with conditional time-intervals. In *FLAIRS*, pages 555–560. AAAI Press, 2008.

3 Philippe Laborie, Jerome Rogerie, Paul Shaw, and Petr Vilím. Reasoning with conditional time-intervals. part II: an algebraical model for resources. In *FLAIRS*. AAAI Press, 2009.

4 Philippe Laborie, Jerome Rogerie, Paul Shaw, and Petr Vilím. IBM ILOG CP optimizer for scheduling - 20+ years of scheduling with constraints at IBM/ILOG. *Constraints An Int. J.*, 23(2):210–250, 2018.

5 L. Michel, P. Schaus, and P. Van Hentenryck. Minicp: a lightweight solver for constraint programming. *Mathematical Programming Computation*, 13(1):133–184, 2021. doi:10.1007/s12532-020-00190-7.

6 Diego Olivier Fernandez Pons and Petr Vilím. Optalcp : un nouveau moteur d'ordonnacement. In *Journées Francophones de Programmation par Contraintes 2024 (JFPC 2024)*, 2024.