Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

STEFAN HEINZ JENS SCHULZ**

Explanations for the Cumulative Constraint: an Experimental Study*

* Supported by the DFG Research Center MATHEON *Mathematics for key technologies* in Berlin.
** Technische Universität Berlin, Institut für Mathematik, Straße des 17. Juni 136, 10623 Berlin, Germany

ZIB-Report 11-13 (May 2011)

Explanations for the Cumulative Constraint: an Experimental Study*

Stefan Heinz[1] and Jens Schulz[2]

[1] Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany
heinz@zib.de
[2] Technische Universität Berlin, Institut für Mathematik, Straße des 17. Juni 136, 10623 Berlin, Germany
jschulz@math.tu-berlin.de

Abstract. In cumulative scheduling, conflict analysis seems to be one of the key ingredients to solve such problems efficiently. Thereby, the computational complexity of explanation algorithms plays an important role. Even more when we are faced with a backtracking system where explanations need to be constructed on the fly.
In this paper we present extensive computational results to analyze the impact of explanation algorithms for the cumulative constraint in a backward checking system. The considered explanation algorithms differ in their quality and computational complexity. We present results for the domain propagation algorithms time-tabling, edge-finding, and energetic reasoning.

1 Introduction

In cumulative scheduling we are given a set of jobs that require a certain amount of different resources. In our case, the resources are renewable with a constant capacity and each job is non-interruptible with a fixed processing time and demand request for several resources. A resource can be, for example, a group of worker with the same specialization, a set of machines, or entities like power supply.

Cumulative scheduling problems have been tackled with techniques from constraint programming (CP), integer programming (IP), or satisfiability testing (SAT). In recent years hybrid approaches are developed which combine methods from these areas. Currently, the best results are reported by a hybrid solver which uses CP and SAT techniques [13]. However, there are still instances with 60 jobs and four cumulative constraints published in the PSPLib [12] that resist to be solved to proven optimality.

Several exact approaches use a search tree to solve cumulative scheduling problems. The idea is to successively divide the given problem instance into smaller subproblems until the individual subproblems are easy to solve. The

* Supported by the DFG Research Center Matheon *Mathematics for key technologies* in Berlin.

best of all solutions found in the subproblems yields the global optimum. During the course of the algorithm, a *search tree* is created with each node representing one of the subproblems. These subproblems are usually generated by adding bound changes, called *branching decisions*, to the current problem. That means the feasibility region gets restricted. At each subproblem, mathematically sophisticated techniques exclude further values from the variables' domains. One of them is domain propagation which infers bound changes on the variables.

Recently it was discovered that *conflict analysis* plays an import role to solve cumulative scheduling problems efficiently [13, 7]. Conflict analysis is used to analyze infeasible subproblems which arise during the search in order to generate *conflict clauses* [10, 1] also known as *no-goods*. These conflict clauses are used to detect similar infeasible subproblems later in the search. In order to perform conflict analysis, a bound change which was performed during the search needs to be explained. Such an *explanation* is a set of bounds which infer the performed bound change. Note that bound changes which are branching decisions cannot be explained. There are two common ways to collect an explanations. One is to submit it directly with the bound change, called *forward checking*. The other way is to reconstruct an explanation on demand which means only if it is needed. This is known as *backward checking*. Both versions have their advantages and disadvantages. A forward checking system always constructs an explanation even if it is not needed, whereas the backward checking framework needs to be able to reconstruct an explanation for a bound change at any point during the search. In the latter case it can be computationally expensive to compute an explanation lazily, since the same bound changes might be explained multiple times.

In this paper we present extensive experimental results which give evidence that minimum-size explanations of bound changes are a crucial component for solving cumulative scheduling instances efficiently. We analyze the impact of constructing explanations in a backward checking system. For our study we consider the domain propagation algorithms time-tabling, edge-finding, and energetic reasoning, see [6]. In case of time-tabling, the complexity status of computing a minimum size explanation is still open. Thus, we evaluate three different heuristic approaches to deliver an appropriate explanation of small size. As benchmark set we use instances of the problem library PSPLib [12] and Pack instances from [4].

Related work. Scheduling problems have been widely studied in the literature. For an overview we refer to [4, 8]. The cumulative constraint was introduced by Aggoun and Beldicneau [3]. Current state-of-the-art propagation algorithms for cumulative are surveyed in [6, 4]. Best results on the instances we are focusing on are achieved by a solver combining CP and SAT techniques [13].

Learning from infeasible subproblems is one of the key ingredients in modern SAT solvers. This technique is called conflict analysis. The basic idea is to conclude a *conflict clause* which helps to prune the search tree and enables the solver to use *non-chronological backtracking*. For a detailed description see for example [10]. There are two main differences of IP and SAT solving in the context of conflict analysis. First, the variables of an IP do not need to be of binary type. Therefore, we have to extend the concept of the conflict graph: it has to

represent bound changes instead of variable fixings, see [1] for details. Second, infeasibility cannot only arise by propagation but also due to an infeasible linear program relaxation [1]. To be able to perform a conflict analysis, it is essential to explain bound changes. This means to state a set of bounds which lead to the proposed bound change. From that a conflict graph is created, then a cut in this graph is chosen, which produces a conflict clause that consists of the bound changes along the frontier of this cut. For cumulative constraints, explanation algorithms to some of the known propagation algorithms are given in [13, 7].

Outline. Section 2 introduces the notation of the considered scheduling problem. In Section 3 we present the propagation and different explanation algorithms that are used in our experimental study, presented in Section 4.

2 Cumulative Scheduling

In cumulative scheduling, an instance is given by a set $\mathcal{J}$ of n non-preemptable jobs with processing times $p_j \in \mathbb{N}$ for each job $j \in \mathcal{J}$. Each job j requests a certain demand r_j of a cumulative resource with capacity $C \in \mathbb{N}$. In a contraint program, a *cumulative* constraint is given by $\texttt{cumulative}(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$, i.e., vectors of start times, processing times and demands, and the capacity. The cumulative constraint enforces that at each point in time t, the cumulated demand of the jobs running at t, does not exceed the given capacity, i.e.,

$$\sum_{j \in \mathcal{J} : t \in [S_j, S_j + p_j)} r_j \leq C \qquad \text{for all } t.$$

Depending on the tightness of the *earliest start times* (est_j), *earliest completion times* (ect_j), *latest start times* (lst_j), and *latest completion times* (lct_j) for each job $j \in \mathcal{J}$, propagation algorithms are able to update variable bounds. Since we use start time variables S_j, the lower bound corresponds to est_j and the upper bound corresponds to lst_j.

3 Propagation and Explanation Algorithms

Explanations tend to create stronger conflict clauses if they include only few variables since we could expect that the constructed conflict graph has a smaller width and size. Hence, one would like to search for minimum sized explanations. On the other side, we are facing a backward checking system which implies that bound changes have to be explained several times during the search. Therefore, explanation algorithms should have a small complexity. In case of the cumulative constraint, computing a minimum sized explanation stands in contrast to a reasonable complexity. In this section we briefly introduce the three propagation algorithms used for our experiments. For each algorithm we state three variants to generate an explanation for a bound change. These constructions differ in their quality (the size of the explanation) and their computational complexity.