On the complexity of global scheduling constraints under structural restrictions

Keywords: constraint programming, global constraints

Abstract

We investigate the computational complexity of two global constraints, CUMULATIVE and INTERDISTANCE. These are key constraints in modeling and solving scheduling problems. Enforcing domain consistency on both is NP-hard. However, in many practical applications, restricted versions of these constraints are sufficient. Examples are scheduling problems with a large number of similar tasks, or tasks sparsely distributed over time. Another example is runway sequencing problems in air-traffic control, where landing periods have a regular pattern. Such cases can be characterized in terms of structural restrictions on the constraints. We identify a number of such structural restrictions and investigate how they impact the computational complexity of propagating these global constraints. In particular, we prove that such restrictions often make propagation tractable.

1 Introduction

There are many varieties of scheduling problems: production, workforce, and resource scheduling, to name just three. Whilst such scheduling problems are intractable in general, the actual instances met in practice may contain restrictions that mean we can solve them efficiently. For example, many scheduling problems can be encoded into Allen's interval algebra. If the interval relations used in this encoding are limited to one of the 18 maximal tractable subalgebras [Krokhin *et al.*, 2001] then the problem can be efficiently solved. This is an example of tractability via a language restriction. An orthogonal way to obtain tractability is via structural restrictions. We impose restrictions on the structure of the problem like the number of processors, or the number of different start times which ensure tractability. We illustrate such a study of scheduling problems with four case studies. The first three are positive: we identify structural restrictions under which the scheduling problem is fixed-parameter tractable or polynomial to solve. The fourth case study is negative: we show that even under strong structural restrictions, a simple scheduling problem with uniform task lengths remains NP-hard.

Since the consistency check for CUMULATIVE and INTERDISTANCE are equivalent to checking whether there exists a schedule satisfying certain restrictions, our work can be seen as tackling scheduling problems. Marx (2011) recently noted noted that there is surprisingly little work on the fixed-parameter tractability of scheduling problems. The few existing examples [Bodlaender and Fellows, 1995; Fellows and McCartin, 2003] present mostly negative results. The only known fixed-parameter tractable results parameterize by the tree-width of the precedence graph and either the number of late tasks or the number of tasks that need to be on time. In contrast, we consider CUMULATIVE as a hard constraint, allowing no late tasks and do not restrict the precedence graph. Instead, we exploit parameters related to the number of distinct values defining characteristics of the tasks can take, and to how many tasks can potentially start at any given time point. Our results for the INTERDISTANCE constraint complement existing results on structural restrictions [Artiouchine *et al.*, 2008]. We note that the parameterized complexity of global constraints was initiated in [Bessière *et al.*, 2008]. Amongst other results, INTERDISTANCE was shown to be fixed-parameter tractable in the total number of holes in the domains of the variables. This line of research was continued in, e.g., [Fellows *et al.*, 2011a; Gaspers and Szeider, 2011; Samer and Szeider, 2011].

2 Background

CSP. A Constraint Satisfaction Problem (CSP) consists of a set of variables X, a set of values D, and a set of constraints C. The question is whether there exists an assignment $\alpha : V \to D$ satisfying all constraints. A *constraint* is a pair $\langle t, R\rangle$; its *scope* $t = (x_1, \ldots, x_r)$ is an r-tuple of variables and its *constraint relation* R is an r-ary relation on D. It is satisfied if $(\alpha(x_1), \ldots, \alpha(x_r)) \in R$. When there is no confusion we also write x_i instead of $\alpha(x_i)$. In the classical CSP, these relations are described explicitly, by listing all allowed tuples. A *global* constraint is one where the constraint relation is described implicitly, enabling a succinct description even when the scope is unbounded. Next, we describe the global constraints CUMULATIVE and INTERDISTANCE.

CUMULATIVE. This was introduced in CHIP for solving scheduling and placement problems [Aggoun and Beldiceanu, 1993]. To define the constraint we consider a set $T = \{1, ..., n\}$ of n tasks and P processors. Each task i has a tuple (r_i, l_i, d_i, p_i) of four natural numbers:

- r_i is the release time of the task, i.e., the time point at which it becomes available,
- l_i is the length of the task, i.e., the amount of time it needs to be processed,
- d_i is the deadline of the task, i.e., the time point at which it needs to have been processed, and
- p_i is the number of processors it uses.

The scope of the constraint contains a variable s_i for each $i \in T$, and it is satisfied by an assignment α if

- $s_i \geq r_i$ for each task $i \in T$,
- $s_i + l_i \leq d_i$ for each task $i \in T$, and
- $\sum_{i \in T : s_i \leq j < s_i + l_i} p_i \leq P$ at each time point $j \in \mathbb{N}$.

We also consider precedence constraints between tasks, expressed by a directed acyclic *precedence graph* $G = (T, A)$. For each arc $(u, v) \in A$, we constrain that $s_u + l_u \leq s_v$.

Due to its expressivity, CUMULATIVE is widely used in constraint programming and is implemented in most constraint solvers. Therefore, a lot of research has been carried out on improving filtering algorithms for this constraint. Enforcing domain consistency on the constraint is NP-hard. However, a number of efficient filtering algorithms have been proposed, e.g. [Caseau and Laburthe, 1996; Mercier and Hentenryck, 2008; Schutt and Wolf, 2010; Vilím, 2011].

INTERDISTANCE. Régin (1997) introduced the INTERDISTANCE constraint, which is useful in modeling scheduling problems on a single processor where all tasks have the same length. We are given a set $T = \{1, ..., n\}$ of n tasks, a task length l, and each task i has a set $S_i \subseteq \mathbb{N}$ of possible start times. The question is whether each task can be assigned a start time $s_i \in S_i$ such that no two tasks overlap, i.e., for every two distinct tasks $i, j \in T$, we have that $|s_i - s_j| \geq l$. Enforcing bounds consistency on the INTERDISTANCE constraint is polynomial [Artiouchine and Baptiste, 2007; Quimper *et al.*, 2008]. However, enforcing domain consistency is NP-hard [Artiouchine *et al.*, 2008].

Parameterized Complexity. Parameterized complexity is a 2-dimensional analogue of the theory of NP-completeness [Downey and Fellows, 1999; Fellows *et al.*, 2011b; Flum and Grohe, 2006; Niedermeier, 2006], where instances are equipped with a parameter k. A parameterized problem Π is *fixed-parameter tractable* (FPT) if there is a computable function f such that every instance (I, k) of Π can be solved in time $f(k) \cdot n^{O(1)}$, where n is the instance size. The theory has a fundamental hierarchy of intractability classes $W[1] \subseteq W[2] \subseteq \ldots$ and a problem that is hard for any of these classes (under parameterized reductions) is not believed to be FPT.

3 Task types

We first consider the CUMULATIVE constraint when the number of distinct release times and deadlines is bounded and there is a single processor. W.l.o.g. we assume that the smallest release time is zero and the latest deadline is d^*. This problem is already NP-hard.

Theorem 1 ([Garey and Johnson, 1979])**.** *Checking consistency of the* CUMULATIVE *constraint is weakly NP-complete even if the number of distinct releases is two, the number of distinct deadlines is also two, and there is a single processor.*

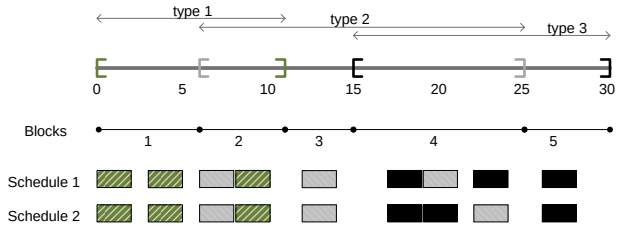

Figure 1: The CUMULATIVE constraint.

Theorem 1 suggests that we need to restrict further the problem to identify tractable cases. In the spirit of parameterizing by the number of numbers [Fellows *et al.*, 2012], we consider the CUMULATIVE constraint parameterized by the number of distinct characteristics. Observe that with this parameterization, the number of distinct quadruples consisting of a release time, a task length, a deadline, and a number of processors, is also bounded by the parameter. Therefore, we introduce a notion of a task *type*. A task i with characteristics (r_i, l_i, d_i, p_i) belongs to the (r, l, d, p)-*type* iff $(r_i, l_i, d_i, p_i) = (r, l, d, p)$.

Given a sequence of tasks, a *run* of the (r, l, d, p)-type is a maximal consecutive subsequence of (r, l, d, p)-type tasks. An *empty* run contains no tasks. A *block* is a maximal time interval that does not contain release times nor deadlines. Given the CUMULATIVE constraint, we can always partition the time-line, $[0, d^*]$, into a set of blocks B.

Example 1. *Consider the* CUMULATIVE *constraint in Fig. 1. The task types are* $(0, 2, 10, 1)$, $(6, 2, 25, 1)$, *and* $(15, 2, 30, 1)$. *Each type consists of three tasks of length 2. Release times and deadlines partition the time line into five blocks.*

The results in this section assume precedences are defined over task types rather than individual tasks.

Lemma 1. *Consider the* CUMULATIVE *constraint on a single processor. Let B be the partitioning of the time-line into blocks and s be a solution of the constraint. Consider $b_j \in B$ and the set of tasks T' such that $[s_i, s_i + l_i] \subseteq b_j$ for all tasks i in T'. Then any permutation of tasks in T' satisfying the precedence constraints gives a solution.*

Proof. The proof follows from the definition of b_j. □

We write that the ith task is in a block b_j iff $[s_i, s_i + l_i] \subseteq b_j$. Schedule 2 in Fig. 1 is a permutation of Schedule 1.

The parameter considered in this section is q, the number of types. Denote by b the number of blocks in B, $|B| = b$. First, note that $b \leq 2q - 1$.

Lemma 2. *Consider the* CUMULATIVE *constraint on a single processor. If there exists a solution of the constraint then there exists a solution where for each block b_i, there is at most one run of each type among the tasks starting in block b_i.*

Proof. Consider any block b_i. By Lemma 1, any permutation of tasks in a block that satisfies the precedence constraints gives a solution. Let π be a topological order of task types in the precedence graph G that start in b_i. Let j be the type of the last task that starts in b_i. Note that this task could end after d_i and, w.l.o.g., assume that this task is the last task in

π. Reorder the tasks starting in b_i according to π. This makes sure that there exists at most one run of each type among the tasks completely contained in b_i and the last run has the same type as the task that crosses the border between this block and the following block. Since b_i was chosen arbitrarily and only tasks that lie completely within b_i were reordered, we can perform the same operation in each other block $b_{i'}$. □

Given Lemma 2, we can write an integer linear program (ILP) to check consistency of the CUMULATIVE constraint, whose number of variables is bounded by a function of q. This shows that consistency checking for CUMULATIVE is FPT. In the INTEGER LINEAR PROGRAMMING FEASIBILITY problem (ILPF), the input is an $m \times n$ matrix $\mathbf{A}$ and m-vector $\mathbf{b}$ of integers, the parameter is n, and we ask if there is an n-vector $\mathbf{x}$ of integers satisfying the m inequalities $\mathbf{Ax} \leq \mathbf{b}$. ILPF is FPT in the number of variables [Lenstra, 1983]. The current fastest algorithm solves ILPF in time $O(n^{2.5n+o(n)} \cdot L)$, where L is the input size [Lokshtanov, 2009].

Theorem 2. *Checking the consistency of the* CUMULATIVE *constraint on a single processor is fixed-parameter tractable parameterized by the number of task types.*

Proof. We encode the problem of checking the consistency of the CUMULATIVE constraint into a set of ILPs, each with a number of variables bounded by a function of q. The size of this set is bounded by a function of q as well, and the CUMULATIVE constraint is satisfiable iff at least one ILP is feasible. By Lemma 2 we know that if a solution exists then there exists a solution with at most one run of each type starting in each block. The last run in each block might cross the border between this block and the next block. To simplify notations, we assume that there exists exactly one run of each type starting in each block, so that some runs are empty. We use a pair (i, j) to denote the ith run of type j. Then the total number of runs is $bq \leq (2q-1)q$. Any solution of the constraint with at most one run of each type among the tasks starting in the same block constitutes a permutation of such runs (i, j), $i = 1, \ldots, b, j = 1, \ldots, q$ that satisfies precedence constraints. Moreover, the number of permutations $O((bq)!)$ is also bounded by a function of q. For each valid permutation of runs, we will solve an ILP to check if this permutation can be extended to a feasible solution.

Given a permutation of runs π, we write $(i, j) \prec (k, l) \in \pi$ iff the ith run of type j precedes the kth run of type l in π. We emphasize that we only consider permutations that satisfy precedence constraints. To build the ILP, we need to identify the start time of each run and the number of tasks in each run. We introduce two sets of integer variables. The first set of variables is $s_{i,j}$, $i = 1, \ldots, b, j = 1, \ldots, q$. Variable $s_{i,j}$ equals the starting time for the ith run of the jth type, $s_{i,j} \in [0, d^*]$. The second set of variables is $x_{i,j}$, $i = 1, \ldots, b, j = 1, \ldots, q$. Variable $x_{i,j}$ equals the number of tasks in the ith run of type j, $x_{i,j} \in [0, n_j]$, where n_j is the number of tasks of type j. Note that $x_{i,j} = 0$ means that the ith run of type j is empty. We obtain the following ILP for $i, k \in \{1, \ldots, b\}$ and $j, l \in \{1, \ldots, q\}$:

$$r_j \leq s_{i,j}, \qquad \forall i, j \quad (1)$$

$$s_{i,j} + x_{i,j} \cdot l_j \leq d_j, \qquad \forall i, j \quad (2)$$

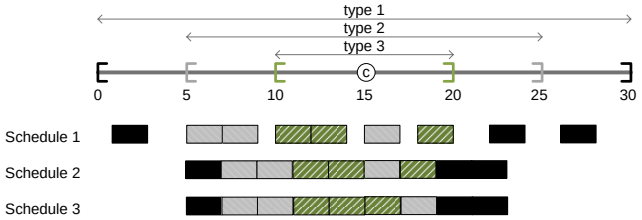

Figure 2: The CUMULATIVE constraint with nested types.

$$s_{i,j} + x_{i,j} \cdot l_j \leq s_{k,l}, \quad \forall i, j, k, l \text{ s.t. } (i, j) \prec (k, l) \in \pi \quad (3)$$

$$\sum_{i=1}^{q} x_{i,j} = n_j. \qquad \forall j \quad (4)$$

Equations 1–2 make sure that each run of type j is performed within the interval $[r_j, d_j]$. Equation 3 ensures that runs are performed in the order specified by the permutation π and that they do not overlap. Finally, Equation 4 guarantees that all tasks are performed. Since the number of variables is bounded by a function of q, checking the consistency of CUMULATIVE on a single processor is fixed-parameter tractable parameterized by the number of task types. □

Corollary 1. *Checking the consistency of the* CUMULATIVE *constraint on multiple processors is* FPT *parameterized by the number of task types and the number of processors if each task requires one processor,* $p_i = 1$, $i = 1, \ldots, n$.

Proof. As in the proof of Theorem 2, we need to consider all possible permutations of runs that satisfy precedence constraints. However, each run is characterized by an additional parameter which is a machine where this run is scheduled. The tuple (i, j, k) describes the ith run of jth type that runs on the kth machine. The total number of runs is $O(qbP)$ which is bounded by the parameter. We solve an ILP for each permutation in a similar way. However, we need to add extra constraints to the ILP to make sure that for each pair of tasks of types j and j', such that $j \prec j'$ in G, that run on different processors, respect precedence. Hence, if we have $(i, j, k) \prec (i', j', k')$, $k \neq k'$, in a permutation π and $j \prec j'$ in G, then we add the constraint: $s_{i,j,k} + x_{i,j} \cdot l_j \leq s_{i',j',k'}$ The obtained ILP has $O(qbP)$ variables. Hence, the problem is FPT in the number of task types and processors. □

3.1 Nested task types

Here, we consider a special case of the CUMULATIVE constraint where the task types have a nested structure. The considered parameter is still the number of task types, q.

Definition 1. *Task types are* nested *if* $r_i \leq r_{i+1}$, $d_{i+1} \leq d_i$ *and there is a common point* c *with* $c \in [r_i, d_i]$, $i = 1, \ldots, n$.

Example 2. *The* CUMULATIVE *constraint in Fig. 2 has three nested types:* $(0, 2, 30, 1)$, $(5, 2, 25, 1)$ *and* $(10, 2, 20, 1)$.

Next we show some useful properties of this restriction of the CUMULATIVE constraint. Recall that n_j denotes the number of tasks of type j. We say that a solution *has no gaps* if there exist time points t_1 and t_2 such that $\sum_{j=1}^{q} n_j l_j = t_2 - t_1 + 1$ and the processor is idle within $[0, t_1) \cup [t_2, d^*]$.

Lemma 3. *Consider the* CUMULATIVE *constraint for a single processor with nested types. If there exists a solution of the constraint then there exists a solution with no gaps.*

Proof. Let s be a solution of the constraint. Consider the first two consecutive tasks i and k in the schedule such that $s_i + l_i < s_k$, where s_i and s_k are starting times of tasks i and k, respectively. Let c be any time point between the release time and the deadline of the innermost type. If $s_k \leq c$ then we can shift all tasks scheduled before the ith task and the ith task to the right by $s_k - (s_i + l_i)$ time units. This preserves the solution due to the nested structure of the types. Similarly, if $s_i + l_i \geq c$ then we can shift all tasks scheduled after the kth task and the kth task to the left by $s_k - (s_i + l_i)$ time units. Finally, if $c \in [s_i + l_i, s_k]$ then we shift tasks scheduled before the ith task and the ith task to the right by $c - s_i + l_i$ and tasks scheduled after the kth task and the kth task to the left by $s_k - c$ time units. Again, this transformation preserves the solution due to the nested structure of the types. □

Example 3. *Consider the* CUMULATIVE *constraint from Example 2. Schedule 1 is a valid solution of the constraint. Schedule 2 is a solution with no gaps.*

Lemma 4. *Consider the* CUMULATIVE *constraint for a single processor with nested types. If there exists a solution to the constraint then there exists a solution where there are at most two runs of each type and runs obey the following order:* $(1,1) \prec \ldots \prec (1,q-1) \prec (1,q) \prec (2,q-1) \prec \ldots \prec (2,1)$.

Proof. (Sketch) Consider the first and the second scheduled tasks of type q (q_1 and q_2) and the task j of type j that obey the order $q_1 \prec j \prec \ldots \prec q_2$ in a solution. Then, we can swap tasks j and q_2 in the schedule and obtain another valid solution. Similarly, we reorder the remaining tasks. □

Corollary 2. *Consider the* CUMULATIVE *constraint with a single processor with nested types. If there exists a solution of the constraint then there exists a solution that satisfies the conditions of Lemmas 3–4.*

Schedule 3 in Fig. 2 is a solution that satisfies the conditions of Lemmas 3–4.

Theorem 3. *Checking the consistency of the* CUMULATIVE *constraint on a single processor with nested types is fixed-parameter tractable parameterized by the number of task types.*

Proof. First, we encode the problem of checking the consistency without the precedence graph G into an ILP with a number of variables bounded by a function of q. Then we show how to take into account G. By Lemmas 3–4, if a solution exists then we know that there exists a solution with no gaps and runs obey the order $(1,1) \prec (1,2) \prec \ldots \prec (1,q-1) \prec (1,q) \prec (2,q-1) \prec \ldots \prec (2,1)$. We exploit these properties to build the ILP. We know that tasks of type j are split into at most two runs. The first run completes before one of the common points $c \in [r_q, d_q]$ and the second run starts after this point. However, we do not know c and how many tasks of each type are scheduled before c. First, we need to select c. Second we need to determine the number of tasks in these two runs. For convenience, we assume that the single run of type q is partitioned into two runs at c. The time point $c \in [r_q, d_q]$ is encoded as a variable in the ILP. Variable x_j, $j = 1, \ldots, q$ equals the number of tasks of type j that are scheduled before c. We obtain the following ILP.

$$r_q \leq c \leq d_q$$

$$c - \sum_{k=j}^{q} x_k l_k \geq r_j \qquad \forall j \in \{1, \ldots, q\} \qquad (5)$$

$$c + \sum_{k=j}^{q} (n_k - x_k) l_k \leq d_j \qquad \forall j \in \{1, \ldots, q\} \qquad (6)$$

Equations 5–6 makes sure that each run of the jth type is performed within the interval $[r_j, d_j]$ by exploiting the fact that there are no gaps between tasks (Lemma 3) and runs occur in a specific order (Lemma 4).

Next, we show how to extend this ILP to take into account the precedence constraints. Consider two types i and j such that $i \prec j$. We distinguish two cases. Suppose that the ith type is nested inside the jth type. In this case, x_j must be equal zero, otherwise one of the tasks of type j is scheduled before a task of type i (we assume that there exists at least one task of each type.) Suppose that the jth type is nested inside the ith type. In this case, $x_i = n_i$. Hence, depending on the nesting we add one of these equality constraints to ILP. We perform this extension for each precedence constraint. □

Note that Theorem 3 follows from Theorem 2. However, the ILP in the proof of Theorem 3 has only $O(q)$ variables, while each of the $O((bq)!)$ ILPs in the proof of Theorem 2 contains $O(bq)$ variables. These considerations lead to much faster running times for a nested structure of task types.

4 Number of pairwise overlapping start intervals

In this section, we consider a parameter that is related to the well-known disjunctive ratio [Baptiste and Pape, 2000] and the number of overlapping release-deadline windows. The *start interval* of a task $i \in T$ is the interval $S[i] = [r_i; d_i - l_i]$, which we assume to be non-empty for all tasks. Clearly, in any valid instantiation each task starts at a time point that is within its start interval. For a time point $t \in \mathbb{N}$, we denote by $T[t]$ the set of tasks whose start interval contains t. We consider the parameterization by $\mathfrak{S} := \max_{t \in \mathbb{N}} |T[t]|$. We note that this parameter is more general than the number of overlapping release-deadline windows since for each job, its start interval is strictly contained in its release-deadline window. As usual, we also assume that there is a precedence graph G that defines precedence relations between individual tasks.

Theorem 4. *Checking the consistency of the* CUMULATIVE *constraint on a single processor is fixed-parameter tractable parameterized by* $\mathfrak{S}$.

Proof. We can check the consistency using dynamic programming. Let $min(\emptyset) = \infty$. Let $\prec$ include explicit precedences from G and precedences implied by the start intervals. Given a subset of the tasks S, let $ms(S)$ be the the minimum value of $max(\{s_i : i \in S\})$ over all valid instantiations where all tasks in S are scheduled before those not in S. The

following dynamic program computes $ms(S)$.

$$ms(S) = \begin{cases} -\infty & \text{if } S = \emptyset \\ \infty & \text{if } \exists i \in S, j \notin S \text{ s.t. } j \prec i \\ \min\{\max(r_i, ms(S \setminus \{i\})) + l_i : i \in S \\ \quad \text{and } ms(S \setminus \{i\}) + l_i \leq d_i\} & \text{otherwise} \end{cases}$$

The cumulative constraint with precedences is consistent iff $ms(T) \neq \infty$. Naively, it looks like the dynamic program has complexity $O(n2^n)$. However, there are many S for which $ms(S) = \infty$ due to the constraints. Such S do not have to be stored or expanded. Given a subset S, let $lr(S) = \text{argmax}_{i \in S} r_i$. We claim that for each task k, there are at most $O(2^{|T[r_k]|})$ subsets with $lr(S) = k$ which need to be evaluated in the dynamic program. Suppose S is one such subset. Every task i for which $r_i > r_k$ is not in S by the definition of $lr(S)$. Every task i for which $d_i < r_k + l_k + l_i$ must be in S by the implied precedence constraint. Hence the only tasks which may be in S are those for which $r_i \leq r_k$ and $d_i - l_i \geq r_k + l_k$, which means their start interval contains time r_k. Thus there are at most $O(2^{|T[r_k]|})$ relevant subsets with $lr(S) = k$ as claimed. The task k can be any of the n tasks, so there are $O(n2^{\mathfrak{S}})$ relevant subsets. Each subset is processed in the dynamic program by extending it with any of the $O(n)$ remaining tasks. Hence the overall complexity is $O(n^2 2^{\mathfrak{S}})$, so the consistency check is FPT in $\mathfrak{S}$. □

5 Decomposable Planning Horizon

Consider the CUMULATIVE constraint where each task $i \in T$ has a set $S_i \in \mathbb{N}$ of possible start times. Let the *overlap graph* L be defined as follows. For each task i and each $v \in S_i$, we add the pair (i, v) to L as a vertex. We add an edge between two vertices (i, v) and (j, w) if $[v, v + l_i] \cap [w, w + l_j] \neq \emptyset$. This overlap graph might contain valuable structural information that can be exploited algorithmically. A *complete graph* is a graph with an edge between every pair of distinct vertices. The following theorem could, for instance, correspond to a scenario where tasks are only executed during the daytime, but no processor can execute two tasks during one day.

Theorem 5. *Checking the consistency of the* CUMULATIVE *constraint on multiple processors is polynomial if $p_i = 1$ for each task $i \in T$ and L is a disjoint union of complete graphs.*

Proof. Suppose L is a disjoint union of the k complete graphs $L_1, \ldots, L_k$. Consider the vertices in L_i. The set $\{[v, v + l_j] : (j, v) \in L_i\}$ is pairwise intersecting since L_i is a complete graph. This implies there is some value v_i which is contained in all those intervals. Thus at most P of the assignments in L_i can be taken or the resource constraint will be violated at time v_i. We reduce this to a global cardinality constraint [Régin, 1996] as follows. For each task i, we have a variable x_i with domain $\{j \in \{1, \ldots, k\} : \exists v \in S_i, (i, v) \in L_j\}$. The cumulative constraint is satisfiable iff $gcc([x_1, \ldots, x_n], [1, \ldots, k], [0, \ldots, 0], [P, \ldots, P])$ is satisfiable, i.e., every task is associated to a clique, and each clique has at most P associated tasks. The reduction is polynomial time, as is solving the global cardinality constraint. □

Restrictions on domains	Parameters		
		$types$	$\mathfrak{S}$
No restrictions	NPC Theorem 1	FPT Theorem 2	FPT Theorem 4
No restrictions + precedences	NPC from Theorem 1	FPT Theorem 2	FPT Theorem 4
Nesting	NPC from Theorem 1	FPT Theorem 3	–
Nesting + precedences	NPC from Theorem 1	FPT Theorem 3	–
Decomposable into conflict intervals	P Theorem 5	–	–

Table 1: Summary of results for checking consistency for the CUMULATIVE constraint.

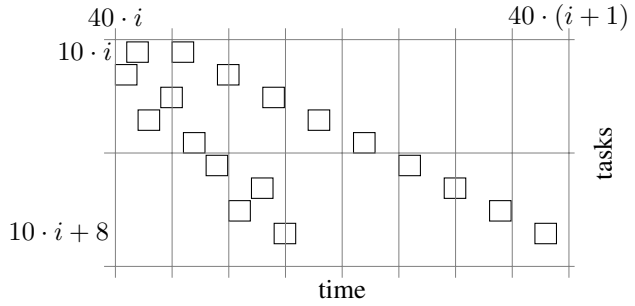

Figure 3: Gadget for variable x_i.

6 Uniform task lengths

An important variant of the CUMULATIVE constraint is the INTERDISTANCE constraint, which is related to Runway sequencing problems for air-traffic control [Artiouchine and Baptiste, 2007; Artiouchine *et al.*, 2008].

6.1 Start times consist of few intervals

The consistency check for INTERDISTANCE is polynomial if all S_i are intervals, i.e., each S_i contains only consecutive integer values [Artiouchine and Baptiste, 2007]. When $l = 1$, INTERDISTANCE is equal to the ALLDIFFERENT constraint, whose consistency check is polynomial [Régin, 1994]. Based on a result by Spieksma and Crama (1992), Artiouchine et al. (2008), we observe that consistency checking is NP-hard when $l = 2$ and each S_i consists of at most three intervals. We close this gap by proving that the problem remains NP-hard when $l = 2$ and each S_i consists of at most two intervals.

Theorem 6. *Checking the consistency of the* INTERDISTANCE *constraint is NP-complete when $l = 2$ and each S_i consists of at most two intervals.*

Proof (sketch). We give a polynomial time reduction from $(\leq 3, \leq 3)$-SAT, an NP-complete variant of the Satisfiability problem where each clause has at most three literals and each variable occurs at most three times [Tovey, 1984]. Given a $(\leq 3, \leq 3)$-SAT formula F, denote by $X = \{x_0, x_1, \ldots, x_{n-1}\}$ its set of variables and $\mathcal{C} = \{C_0, C_1, \ldots, C_{m-1}\}$ its set of clauses. A clause $C_i \in \mathcal{C}$ is a set of at most three literals.

We set the task length $l = 2$. For each variable $x_i \in X$, we introduce the following variable tasks (see Fig. 3):

- task number $10 \cdot i$ with $S_{10i} = \{40i + 1, 40i + 5\}$ is the *main* task for variable x_i,

- tasks number $10 \cdot i + 1, \ldots, 10 \cdot i + 8$ with

$$S_{10i+j} = \begin{cases} \{40i, 40i + 9\} & \text{if } j = 1 \\ \{40i + 4, 40i + 13\} & \text{if } j = 2 \\ \{40i + 2, 40i + 17\} & \text{if } j = 3 \\ \{40i + 6, 40i + 21\} & \text{if } j = 4 \\ \{40i + 8, 40i + 25\} & \text{if } j = 5 \\ \{40i + 12, 40i + 29\} & \text{if } j = 6 \\ \{40i + 10, 40i + 33\} & \text{if } j = 7 \\ \{40i + 14, 40i + 37\} & \text{if } j = 8. \end{cases}$$

For each clause $C_i \in \mathcal{C}$, we introduce four new tasks. Suppose $C_i = \{l_{i_1}, l_{i_2}, l_{i_3}\}$, each $l_{i_j} = x_{i_j}$ or $l_{i_j} = \neg x_{i_j}$, $j = 0, 1, 2$, and l_{i_j} is the k_{i_j}th occurrence of variable x_{i_j} in F, $1 \leq k_{i_j} \leq 3$. Then the new tasks are

- literal tasks numbered $10n + 4i + j$, with $j = 0, 1, 2$, with possible starting times $S_{10n+4i+j} =$

$$\begin{cases} \{40i_j + 13 + 8k_{i_j}, 40n + 8i + 2j\} & \text{if } l_{i_j} = x_{i_j} \\ \{40i_j + 9 + 8k_{i_j}, 40n + 8i + 2j\} & \text{if } l_{i_j} = \neg x_{i_j}. \end{cases}$$

- a clause task numbered $10n + 4i + 3$ with possible starting times in the interval $[40n + 8i; 40n + 8i + 4]$.

This finishes the description of the reduction.

One can show that the $(\leq 3, \leq 3)$-SAT instance is satisfiable if and only if the constructed INTERDISTANCE constraint has a valid instantiation. To this end, we set $\alpha(x_i) =$ true if the main task for variable x_i is scheduled at time point $40i + 1$ and $\alpha(x_i) =$ false otherwise. In the first case, each task $10 \cdot i + j$, with $j \in \{3, 5, 7\}$, is scheduled at time point $40i + 4j + 5$. Therefore, for a clause containing $\neg x_i$, the corresponding literal task is scheduled after the time point $40n$. The argument for positive literals is similar. Since the clause task occupies two units of time from the six units where literal clauses could be scheduled after $40n$, one of the literal tasks must be scheduled before $40n$ and the corresponding literal therefore satisfies the clause. □

6.2 Start times at equal distances

Artiouchine et al. (2008) also studied restrictions of the problem where $S_i = \{r_i, r_i + h, \ldots, r_i + k_i \cdot h\}$. In this case, which we call REGULAR INTERDISTANCE, airplane i approaching a terminal at time r_i can either land immediately or fly some number of at most k_i loops in a holding pattern, each taking some fixed amount of time h, before it lands.

A *proper interval graph* is a graph such that its vertices can be bijectively mapped to intervals of the real line, all with integer coordinates and equal lengths, such that two vertices are adjacent iff their intervals intersect. The set of these intervals is the *interval model* of the graph.

We give a reduction from a graph coloring problem for proper interval graphs, called (γ, μ)-coloring [Bonomo *et al.*, 2009]. The input is a proper interval graph $G = (V, E)$ and functions $\gamma, \mu : V \to \mathbb{N}$ and the question is whether there exists a coloring $f : V \to \mathbb{N}$ assigning distinct colors to every two adjacent vertices such that $\gamma(v) \leq f(v) \leq \mu(v)$.

We construct a REGULAR INTERDISTANCE constraint as follows. The task length l is the length of an interval in the interval model. The integer h is the position of the rightmost right endpoint of an interval in the interval model. For each vertex u_i with interval $[a_i; a_i + l)$, create a task i with $S_i = \{a_i + \gamma(u_i) \cdot h, a_i + (\gamma(u_i) + 1) \cdot h, \ldots, a_i + \mu(u_i) \cdot h\}$.

It is easy to see that G has a (γ, μ)-coloring iff there is a legal instantiation for the constructed REGULAR INTERDISTANCE constraint. Since Precoloring Extension, a special case of (γ, μ)-coloring, is NP-complete on proper interval graphs [Marx, 2006], we have the following theorem.

Theorem 7. *Checking consistency of* REGULAR INTERDISTANCE *is NP-complete, even if each* $k_i \in \{1, k\}$ *for an integer* k.

Consider now the special case of the REGULAR INTERDISTANCE constraint where $k_1 = \cdots = k_n := k$ and $\max_{i \in T} r_i + l \leq \max_{i \in T} r_i + T$, i.e., each task has k time slots where it can be scheduled and the time line can be partitioned into disjoint intervals $I_0, \ldots, I_k$ where I_i contains all time points where a task may be scheduled if it occupies its ith time slot. This special case was shown to be linear-time solvable by means of a mixed-integer programming formulation [Artiouchine *et al.*, 2008]. But note that for this special case the above reduction can be reversed, giving a k-coloring problem for proper interval graphs, which is linear-time solvable [Golumbic, 1980]. Thus, we obtain the same theorem as [Artiouchine *et al.*, 2008] by simpler means.

Theorem 8. *The consistency of the* REGULAR INTERDISTANCE *constraint with* $k_1 = \cdots = k_n$ *and* $\max_{i \in T} r_i + l \leq \max_{i \in T} r_i + T$ *can be checked in linear time.*

The requirement that $k_1 = \cdots = k_n$ is essential in this theorem. If we drop it we obtain an instance of the NP-complete μ-coloring problem on unit interval graphs [Bonomo *et al.*, 2012], which is equivalent to (γ, μ)-coloring with $\gamma(v) = 0$ for all $v \in V$. It would be interesting to know in this context whether the (γ, μ)-coloring problem is FPT for proper interval graphs with parameter $\max_{v \in V} \mu(v) - \gamma(v)$.

7 Conclusions

Scheduling problems are an important and well studied application of combinatorial optimization. Due to their combinatorial nature, scheduling problems are computationally hard even for very restricted cases. Surprisingly, one angle of this problem, scheduling problems under structural restrictions has received little attention in the literature. In this work, we have made some interesting steps forward in our understanding of the hardness of scheduling problems. We have shown that we can exploit structural restrictions to obtain new tractable classes of scheduling problems. For example, we proved that scheduling problems with a large number of identical tasks are tractable. On the other hand, we demonstrated that scheduling problems with uniform task lengths remain intractable even under very strong structural restrictions. To this end, we revealed an interesting connection between this class of problems and the list coloring problem on proper interval graphs. Our work raises a number of open questions, most importantly: what other interesting structural restrictions and parameters make scheduling problems tractable?

References

[Aggoun and Beldiceanu, 1993] Abderrahmane Aggoun and Nicolas Beldiceanu. Extending chip in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling*, 17(7):57 – 73, 1993.

[Artiouchine and Baptiste, 2007] Konstantin Artiouchine and Philippe Baptiste. Arc-b-consistency of the inter-distance constraint. *Constraints*, 12(1):3–19, 2007.

[Artiouchine *et al.*, 2008] Konstantin Artiouchine, Philippe Baptiste, and Christoph Dürr. Runway sequencing with holding patterns. *European Journal of Operational Research*, 189(3):1254–1266, 2008.

[Baptiste and Pape, 2000] Philippe Baptiste and Claude Le Pape. Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. *Constraints*, 5(1/2):119–139, 2000.

[Bessière *et al.*, 2008] Christian Bessière, Emmanuel Hebrard, Brahim Hnich, Zeynep Kiziltan, Claude-Guy Quimper, and Toby Walsh. The parameterized complexity of global constraints. In *AAAI 2008*, 235–240. 2008.

[Bodlaender and Fellows, 1995] Hans L. Bodlaender and Michael R. Fellows. W[2]-hardness of precedence constrained k-processor scheduling. *Operations Research Letters*, 18(2):93–97, 1995.

[Bonomo *et al.*, 2009] Flavia Bonomo, Guillermo Durán, and Javier Marenco. Exploring the complexity boundary between coloring and list-coloring. *Annals of Operations Research*, 169(1):3–16, 2009.

[Bonomo *et al.*, 2012] Flavia Bonomo, Yuri Faenza, and Gianpaolo Oriolo. On coloring problems with local constraints. *Discrete Mathematics*, 312(12-13):2027–2039, 2012.

[Caseau and Laburthe, 1996] Yves Caseau and François Laburthe. Cumulative scheduling with task intervals. In *Proc. of the Joint International Conference and Symposium on Logic Programming*, 363–377, 1996.

[Downey and Fellows, 1999] Rodney G. Downey and Michael R. Fellows. *Parameterized complexity*. Springer-Verlag, 1999.

[Fellows and McCartin, 2003] Michael R. Fellows and Catherine McCartin. On the parametric complexity of schedules to minimize tardy tasks. *Theoretical Computer Science*, 2(298):317–324, 2003.

[Fellows *et al.*, 2011a] Michael R. Fellows, Tobias Friedrich, Danny Hermelin, Nina Narodytska, and Frances A. Rosamond. Constraint satisfaction problems: Convexity makes alldifferent constraints tractable. In *IJCAI 2011*, 522–527. 2011.

[Fellows *et al.*, 2011b] Michael R. Fellows, Serge Gaspers, and Frances Rosamond. Multivariate complexity theory. In *Computer Science: The Hardware, Software and Heart of It*, chapter 13, 269–293. Springer, 2011.

[Fellows *et al.*, 2012] Michael R. Fellows, Serge Gaspers, and Frances A. Rosamond. Parameterizing by the number of numbers. *Theory of Computing Systems*, 50(4):675–693, 2012.

[Flum and Grohe, 2006] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*, volume XIV of *Texts in Theoretical Computer Science.* . Springer, 2006.

[Garey and Johnson, 1979] Michael R. Garey and David R. Johnson. *Computers and Intractability*. W. H. Freeman and Co.y, 1979.

[Gaspers and Szeider, 2011] Serge Gaspers and Stefan Szeider. Kernels for global constraints. In *IJCAI 2011*, 540–545. 2011.

[Golumbic, 1980] Martin C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, 1980.

[Krokhin *et al.*, 2001] Andrei Krokhin, Peter Jeavons, and Peter Jonsson. Reasoning about temporal relations: The tractable subalgebras of Allen's interval algebra. *Journal of the ACM*, 50:2003, 2001.

[Lenstra, 1983] Hendrik W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

[Lokshtanov, 2009] Daniel Lokshtanov. *New Methods in Parameterized Algorithms and Complexity*. PhD thesis, University of Bergen, 2009.

[Marx, 2006] Dániel Marx. Precoloring extension on unit interval graphs. *Discrete Applied Mathematics*, 154(6):995–1002, 2006.

[Marx, 2011] Dániel Marx. Fixed-parameter tractable scheduling problems. In *Packing and Scheduling Algorithms for Information and Communication Services (Dagstuhl Seminar 11091)*. 2011.

[Mercier and Hentenryck, 2008] Luc Mercier and Pascal Van Hentenryck. Edge finding for cumulative scheduling. *INFORMS Journal on Computing*, 20(1):143–153, 2008.

[Niedermeier, 2006] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

[Quimper *et al.*, 2008] Claude-Guy Quimper, Alejandro López-Ortiz, and Gilles Pesant. A quadratic propagator for the inter-distance constraint. *Constraint Programming Letters*, 3:21–35, 2008.

[Régin, 1994] Jean-Charles Régin. A filtering algorithm for constraints of difference in CSPs. In *Proc. of the 12th National Conference on Artificial Intelligence (AAAI 1994)*, 362–367. 1994.

[Régin, 1996] Jean-Charles Régin. Generalized arc consistency for global cardinality constraint. In *AAAI/IAAI, Vol. 1*, 209–215, 1996.

[Régin, 1997] Jean-Charles Régin. The global minimum distance constraint. Technical report, ILOG, 1997.

[Samer and Szeider, 2011] Marko Samer and Stefan Szeider. Tractable cases of the extended global cardinality constraint. *Constraints*, 16(1):1–24, 2011.

[Schutt and Wolf, 2010] Andreas Schutt and Armin Wolf. A new $O(n^2 \log n)$ not-first/not-last pruning algorithm for cumulative resource constraints. In *Proc. of CP-2010*, 445–459, 2010.

[Spieksma and Crama, 1992] Frits C.R. Spieksma and Yves Crama. The complexity of scheduling short tasks with few starting times. Reports in operations research and systems theory M92-06, University of Limburg, 1992.

[Tovey, 1984] Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85 – 89, 1984.

[Vilím, 2011] Petr Vilím. Timetable edge finding filtering algorithm for discrete cumulative resources. In *CPAIOR'11*, 230–245, 2011.