Unite and Lead: Finding Disjunctive Cliques for Scheduling Problems

Konstantin Sidorov[1]
Delft University of Technology, The Netherlands

Imko Marijnissen[1]
Delft University of Technology, The Netherlands

Emir Demirović
Delft University of Technology, The Netherlands

Abstract

Constraint programming solvers have seen much success in scheduling problems owing to their efficient reasoning over constraints to solve complex problems in practice. Many algorithms have been proposed for propagating information from a *single* constraint. However, inferring and exchanging information across multiple constraints can provide deeper insight into the *global* structure of a problem. In this work, we propose to exchange information amongst constraints by inferring the disjointness of tasks in scheduling problems from *many* constraints. We do this by (i) augmenting existing propagators, such as the `Cumulative` and nogoods, to report when pairs of tasks are disjoint, and (ii) leveraging this information by introducing the `SelectiveDisjunctive` propagator which generates a lower bound on the earliest completion time of cliques of disjoint tasks to determine conflicts. This allows us to aggregate disjointness information spanning multiple constraints to gain a better global overview of the problem, as well as more precise local information. We also identify a problem structure where an LCG solver reasoning over `Cumulative` constraints separately, without any reformulations, requires an exponential amount of time to prove infeasibility, which we both justify theoretically and show empirically; on the other hand, our approach solves those instances in polynomial time. On particular known RCPSP and RCPSP/max benchmarks, our approach significantly reduces the number of conflicts required to prove optimality when resource contention is high. Additionally, we discover new lower bounds for 16 RCPSP/max instances (closing six of them) and four RCPSP instances (closing one), as well as new upper bounds for two RCPSP/max instances and four RCPSP instances. Furthermore, we empirically analyse our proposed approach to determine which features are beneficial for performance, showing that *finding* cliques is one of the main bottlenecks and that detecting disjointness during search can lead to improved bounds on certain instances, but it generally negatively impacts learning. This work paves the way for reasoning over the disjointness of tasks inferred from a variety of standard constraints to discover novel information sourced from multiple constraints during search.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming

Keywords and phrases Constraint Programming, Lazy Clause Generation, Propagation, Scheduling, Cumulative, Disjunctive

Digital Object Identifier 10.4230/LIPIcs.CP.2025.35

Supplementary Material
Software (Pumpkin and processing scripts): `https://doi.org/10.5281/zenodo.15624366`
Dataset (Instances and solver logs): `https://doi.org/10.5281/zenodo.15624416`

Funding *Konstantin Sidorov*: supported by the TU Delft AI Labs program as part of the XAIT lab.
Imko Marijnissen: supported by the NWO/OCW, as part of the Quantum Software Consortium programme (project number 024.003.037 / 3368).

[1] Both authors contributed equally to this research.

© Konstantin Sidorov, Imko Marijnissen, and Emir Demirović;
licensed under Creative Commons License CC-BY 4.0

Acknowledgements The authors would like to thank Maarten Flippo for the productive discussions on scheduling applications for constraint programming during the early phases of this project.

1 Introduction

Scheduling is one of the important classes of problems in the field of optimisation that can be loosely described as the problem of scheduling a set of tasks to satisfy certain constraints while optimising *some* objective. Constraint programming (CP) has been successful in solving scheduling problems. The key advantage of CP is its native support for constraints that capture task resource usage and efficient propagation algorithms to prune the search space, such as time-tabling [29] or energetic reasoning [2]. Another promising technique is exploiting the disjointness of tasks, as identified by Gay et al. [16] to infer bounds on tasks.

A common trait of the previously mentioned works is that they reason over a *single* resource, since reasoning jointly over multiple resources significantly increases the computational cost. However, the benefits of stronger inference obtained by aggregating and exchanging information across *multiple* resource constraints may outweigh the drawbacks, an opportunity underexplored in constraint programming for scheduling.

To illustrate the issues with single-resource reasoning, consider the following example:

▶ **Example 1.** We have two tasks on different resources and aim to minimise the latest finish time (makespan). Individually, neither resource constraint allows us to make any non-trivial inferences about the makespan. However, if we combine this with the information that the two tasks cannot be executed at the same time, then we can see that they need to be executed sequentially, which allows us to derive a tighter bound on the makespan of these two tasks.

To address this, we propose a principled solution for aggregating and exchanging information across constraints by reasoning over the disjointness of task pairs, based on both static incompatibilities (i.e., present in the original model) and dynamic incompatibilities (i.e., discovered during search). Our approach consists of three core components:

Variable creation We create Boolean variables representing whether two tasks are disjoint.

Disjointness mining We extend the existing propagators for constraints defined in the problem to infer whether two tasks should be disjoint and allow propagators to use the Boolean variables to perform additional propagation.

Conflict detection We introduce the `SelectiveDisjunctive` propagator that aggregates this disjointness information by deriving lower bounds on the earliest completion time of disjoint cliques of tasks to detect conflicts.

By doing so, we can detect inconsistent states for disjoint tasks by (a) aggregating information from the *whole* problem in contrast to the more orthodox focus on single constraints, (b) allowing propagators to use this shared information, and (c) computing conflicting task groups *during search*, thus using information obtained later in the search. As finding maximum cliques is an NP-hard problem [23], we propose a heuristic to find disjoint cliques of tasks based on minimising the growth of the interval spanned by the clique when adding a task.

To support the usefulness of our contributions, we identify an instance structure for which state-of-the-art solvers are guaranteed to incur an exponential number of conflicts unless they reformulate the problem. While state-of-the-art LCG solvers also exhibit exponential runtime in practice, our approach can solve the same instances in polynomial time.

To evaluate our approach, we implemented it in Pumpkin [12] and ran experiments on well-known scheduling benchmarks (RCPSP and RCPSP/max). Comparing our approach to baseline Pumpkin and Google OR-Tools CP-SAT [31] (a state-of-the-art CP solver that has won many editions of the MiniZinc Challenge since 2017) shows that our approach works well when resource contention is high, with some instances exhibiting improvements of *three orders of magnitude*. For RCPSP/max, our approach discovers 16 new lower bounds (and closes six instances) and two new upper bounds; whereas for RCPSP, our approach yields four new upper bounds, four new lower bounds, and one closed instance. Our ablation study shows that the main bottleneck is finding the cliques *but* that it can be crucial to find the *right* cliques, whereas disjointness mining hampers performance due to poor learning on most instances.

To summarise, we address the insufficiency of *local* information by reasoning over the disjointness of tasks to infer *global* information. We do this by (1) introducing Boolean variables representing the disjointness between two tasks and extending existing propagators to infer the values of these variables *and* use them for additional propagation, and (2) introducing the `SelectiveDisjunctive` propagator that heuristically finds disjoint cliques to detect conflicts unattainable by current approaches due to their focus on individual constraints. We identify an instance structure for which current techniques are guaranteed to incur an exponential number of conflicts, while our approach only requires a polynomial number of them. We experimentally evaluate our approach and observe that it is most beneficial for instances with high resource contention, leading to *orders-of-magnitude* improvements for specific benchmarks. Through an ablation study, we find that dynamically determining which disjoint cliques to focus on can be beneficial and that looking for disjointness during search inhibits the learning capabilities of the solver. This work paves the way for aggregating information across constraints to derive *global* information about the problem.

The rest of the paper is structured as follows. Section 2 introduces the concepts of constraint programming relevant to our work. Section 3 reviews the previous research on disjunctive reasoning. We present and motivate our strategy for discovering and reasoning about disjunctive tasks in Section 4. We evaluate our approach on a range of scheduling problems in Section 5. Finally, we conclude and review further research directions in Section 6.

2 Preliminaries

Constraint programming framework. A Constraint Satisfaction Problem (CSP) consists of a tuple $(\mathcal{X}, \mathcal{C}, \mathcal{D})$ where $\mathcal{X}$ is the set of *variables*, $\mathcal{C}$ is the set of *constraints* which specify the relations between variables, and $\mathcal{D}$ is the *domain* which specifies for each variable which values it can take; we refer to the lowest and highest domain values as *lower* and *upper* bounds and denote them $\text{LB}(x)$ and $\text{UB}(x)$, respectively [32]. A *solution* $\mathcal{I}$ is a mapping that maps each variable in $\mathcal{X}$ to a single value in the domain of that variable in $\mathcal{D}$ which satisfies all of the constraints in $\mathcal{C}$. An *atomic constraint* (i.e. *atomic predicate*) is a predicate over a single integer variable $x \in \mathcal{X}$ and value v signified by $[\![x \otimes v]\!]$ where $\otimes \in \{=, \leq, \geq, \neq\}$.

Constraint programming (CP) is a paradigm for solving CSPs; CP solvers enforce constraints through *propagators*, each represented with a function $f : \mathcal{D} \mapsto \mathcal{D}'$ (where $\mathcal{D}' \subseteq \mathcal{D}$) which removes values from $\mathcal{D}$ infeasible under the constraints in $\mathcal{C}$. After applying the propagators, the solver makes a *decision* which creates several subproblems by splitting the domain of a variable into two or more parts. This process of applying propagators and making decisions is performed until either a solution $\mathcal{I}$ is found, the problem is found to be unsatisfiable, or a termination criterion is met.

During the search process, CP solvers can use a technique called *nogood learning* [10]. A nogood is a partial assignment that cannot be extended to a full solution, typically rendered with an implication of the form $N = p_1 \wedge \cdots \wedge p_k \implies \bot$ with p_j being the atomic constraints encoding the partial assignment; a nogood is *unsatisfied* if no p_j is falsified. The purpose of adding nogoods is to prevent the search process from (re-)exploring parts of the search space. In this work, we use the lazy clause generation framework [14] to derive these nogoods. In this framework, it is required that the propagators explain their inferences in terms of atomic constraints, that is, they produce *explanations* of the form $e_1 \wedge \cdots \wedge e_m \implies q$.

`Cumulative` constraint. `Cumulative` is a constraint useful for many scheduling problems to model limited renewable resources, such as available work-hours. However, determining the satisfiability of a single `Cumulative` constraint is NP-complete [2], meaning that even reasoning efficiently over a single `Cumulative` requires reasoning over relaxations of the problem.

We define the `Cumulative` in Definition 2, the variable s_i encodes the start time of task i, and $(s_i + d_i)$ the finish time of i. Thus, in this constraint, we implicitly associate each task $i \in T$ with an interval $[s_i, s_i + d_i)$. We also introduce notation for the bounds of tasks and sets of tasks.

▶ **Definition 2.** *Let T be a set of* ***tasks****, and let a task $i \in T$ be defined by its start variable s_i, resource usage $r_i \in \mathbb{Z}_{\geq 0}$, and duration $d_i \in \mathbb{Z}_{\geq 0}$. Finally, let $C \in \mathbb{Z}_{\geq 0}$ be the* ***capacity*** *of the resource. Then the `Cumulative` constraint is the condition that at any time point τ the cumulative resource usage of intervals $[s_i, s_i + d_i)$ covering τ does not exceed the capacity:*

$$\forall \tau \in \mathbb{Z} \ : \sum_{i \in T \ : \ s_i \leq \tau < s_i + d_i} r_i \leq C. \tag{1}$$

Given a task $i \in T$, we denote its: ***earliest start time*** $\mathrm{EST}_i = \mathrm{LB}(s_i)$*,* ***earliest completion time*** $\mathrm{ECT}_i = \mathrm{LB}(s_i) + d_i$*,* ***latest start time*** $\mathrm{LST}_i = \mathrm{UB}(s_i)$*,* ***latest completion time*** $\mathrm{LCT}_i = \mathrm{UB}(s_i) + d_i$*, and* ***energy*** $e_i = r_i \times d_i$*.*

Given a ***set*** *of tasks $\Omega \subseteq T$, we denote its:* ***total duration*** $d_\Omega = \sum_{i \in \Omega} d_i$*,* ***earliest start time*** $\mathrm{EST}_\Omega = \min_{i \in \Omega} \mathrm{LB}(s_i)$*,* ***latest completion time*** $\mathrm{LCT}_\Omega = \max_{i \in \Omega} \mathrm{UB}(s_i) + d_i$*,* ***earliest completion time*** $\mathrm{ECT}_\Omega = \max_{\Omega' \subseteq \Omega} \mathrm{EST}_{\Omega'} + d_{\Omega'}$*, and* ***total energy*** $e_\Omega = \sum_{i \in \Omega} e_i$

An important concept for the inference over `Cumulative` constraints is the *mandatory part* (Figure 1a), which is a time interval that is covered by a task regardless of its placement:

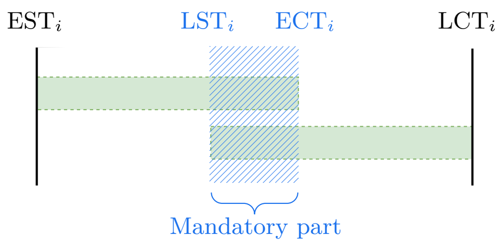
▶ **Definition 3.** *Given a `Cumulative` constraint over tasks T with capacity C, we denote the* ***mandatory part*** MP_i *of a task $i \in T$ as the interval $\mathrm{MP}_i := [\mathrm{LST}_i, \mathrm{ECT}_i)$. We define a* ***height*** *at a time point τ as total consumption of mandatory parts covering τ, that is,* $\mathrm{Height}(\tau) := \sum_{i \in T \ : \ \tau \in \mathrm{MP}_i} r_i$*. We also define a* ***reduced height*** *at a time point τ without $\Omega \subseteq T$ as* $\mathrm{Height}^-(\tau,\ \Omega) := \sum_{i \in T \setminus \Omega \ : \ \tau \in \mathrm{MP}_i} r_i$*. Finally, we define a* ***profile*** *Pr_x as a rectangle (a_x, b_x, h_x) where $\forall \tau \in [a_x, b_x]$: $\mathrm{Height}(\tau) = h_x$ and we assume that all profiles are maximal (i.e. there are no adjacent profiles with the same height).*

`Disjunctive` (`No-Overlap`) constraint. In this paper, we focus on a special case of `Cumulative` where the resource has unit capacity $C = 1$ and each task has unit resource usage $r_i = 1$, known as `Disjunctive`. Many inference procedures that are intractable for `Cumulative` constraints can be efficiently executed for `Disjunctive` constraints. The core inference that we use is infeasibility checking (Theorem 4), as illustrated by Figure 1b.

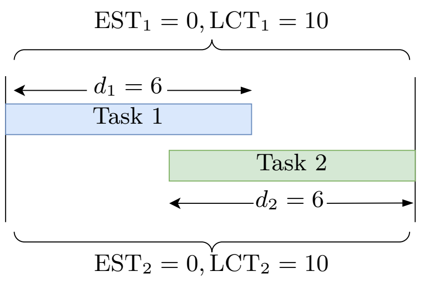
▶ **Theorem 4** (Overload checking). *Let $\Omega \subseteq T$ be a set of variables bound by a `Disjunctive` constraint such that $d_\Omega > \mathrm{LCT}_\Omega - \mathrm{EST}_\Omega$. Then the `Disjunctive` constraint has no feasible solutions [37], and any feasible solution of the CSP satisfies*

$$\left(\bigwedge_{i \in \Omega} [\![s_i \geq \mathrm{EST}_\Omega - \delta^-]\!] \right) \wedge \left(\bigwedge_{i \in \Omega} [\![s_i \leq \mathrm{LCT}_\Omega - d_\omega + \delta^+]\!] \right) \implies \bot \tag{2}$$

for arbitrary $\delta^-, \delta^+ \in \mathbb{Z}_{\geq 0}$ *satisfying* $\mathrm{LCT}_\Omega - \mathrm{EST}_\Omega + \delta^- + \delta^+ < d_\Omega$ *[38].*



(a) Mandatory part illustration for a *single* task i.


$\mathrm{EST}_1 = 0, \mathrm{LCT}_1 = 10$
$d_1 = 6$
Task 1
Task 2
$d_2 = 6$
$\mathrm{EST}_2 = 0, \mathrm{LCT}_2 = 10$

(b) Overload checking leads to a conflict since $d_1 + d_2 > \mathrm{LCT}_{\{1,2\}} - \mathrm{EST}_{\{1,2\}}$.

Figure 1 Key concepts related to `Cumulative` and `Disjunctive` constraints.

3 Related Work

Much work focused on inferring information based on the `Cumulative` efficiently [29, 34]. One of the approaches of primary interest in this work is edge-finding [29] as it is similar in its conflict detection procedure (also known as *input/output consistency tests* [13, Chapter 4]). A common trait of these approaches [22, 18] is that the earliest completion time of $\Omega \subseteq T$ is used to make inferences. For example, overload checking enforces the condition $\mathrm{ECT}_\Omega > \mathrm{LCT}_\Omega \implies \perp$, while edge finding extends this rule to $\mathrm{ECT}_{\Omega \cup \{i\}} > \mathrm{LCT}_\Omega \implies \Omega \lessdot i$, updating the bounds of t such that it does not overflow the interval bounded by the tasks in Ω. While edge-finding takes into account the earliest start times and latest finish times of the tasks (and horizontally elastic edge-finding [18] takes into account the maximum resource usage of the tasks), this type of reasoning does not take into account the disjointness of tasks and its influence on the earliest completion time of a set of tasks. Furthermore, edge-finding reasons over a single `Cumulative`, whereas we propose gathering information across several `Cumulative` constraints to infer the earliest completion time.

Detecting disjoint pairs of tasks within a `Cumulative` constraints is also a viable reasoning strategy [16], known as time-table disjunctive reasoning. This strategy can detect disjoint task pairs dynamically by taking into account the time points at which other tasks are guaranteed to consume *some* amount of resource. When it detects such disjointness using time-tabling [29], it determines when the bounds of one of the tasks can be updated. The main difference with our approach is that time-table disjunctive reasoning only reasons about a single `Cumulative` constraint while we reason jointly over multiple constraints, and our approach infers disjointness from other constraints besides the `Cumulative`.

To the best of our knowledge, the only work that addresses the reasoning over multiple `Cumulative` constraints is the work by Beldiceanu and Carlsson [5]. This work introduces a multi-resource `Cumulatives` constraint as a generalisation of the `Cumulative`, admitting negative resource consumption and lower/upper bounds on the cumulative resource usage. However, their reasoning makes limited use of the additional information provided by reasoning over multiple `Cumulative` constraints, nor does it take into account disjointness.

Besides reasoning about `Cumulative` constraints, we consider other works that reason about disjunctive tasks. One area is branch-and-bound algorithms for scheduling problems. A common example is the *resource-constrained scheduling problem* (RCPSP), in which one of the lower bounds (LB4) on makespan is the sum of durations in a group of pairwise disjunctive tasks; with similar bounds being proposed for activity groups with at most two or at most k of them running in parallel [24]. Thus, our approach generalises that strategy while also working in the constraint programming context, where the tasks can be (a) arbitrarily bounded by the solver and (b) involved in other constraints. This type of reasoning can also be embedded into the search tree node representation, for example, as a scheduling scheme [7], which accounts for the pairs of tasks that are either disjunctive or running in parallel.

We thus note that there are several methods that attempt to find relations between variables and to exploit these relations to infer information about the bounds. However, the relations discovered by the majority of state-of-the-art techniques are local to individual resource constraints or do not reason over the disjointness of tasks; contrary to that, our work aims to discover this disjointness *dynamically during search* from multiple constraints and use information *across* propagators to infer the earliest completion time of a set of tasks.

4 Our Contribution: Finding Disjunctive Cliques

This section will introduce our approach for aggregating information across constraints. We propose the following workflow:

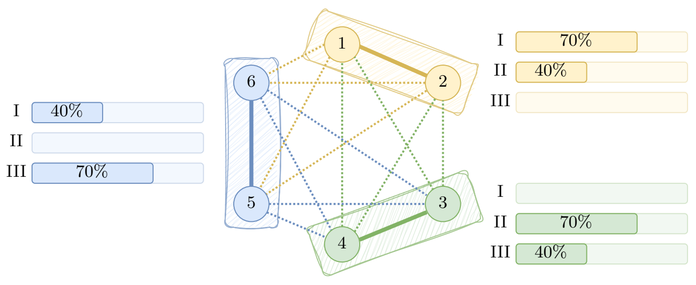
1. Given a set of tasks T (bound by `Cumulative` constraints), introduce new variables δ_{ij} for all $i, j \in T$ and reify them as $\delta_{ij} \rightarrow [s_i, s_i + d_i) \cap [s_j, s_j + d_j) = \emptyset$. (Subsection 4.2)
2. Introduce the reification variables δ into other constraints (such as the `Cumulative` and difference logic) and extend their propagation logic to derive implications of the form $[\![\text{variable bounds}]\!] \implies \delta_{ij}$, and propagate variable bounds based on δ. (Subsection 4.4)
3. Introduce our novel `SelectiveDisjunctive` propagator which derives lower bounds on the earliest competion time of sets of pairwise disjoint tasks given (a) pairs of tasks $\{i, j\}$ such that $\delta_{ij} = 1$, and (b) the bounds on start times to discover conflicts. (Subsection 4.3)

Our approach allows us to gain a *global* view across multiple constraints rather than relying on single constraints to independently make local inferences which causes limited information exchange between propagators. Thus, our approach allows them to communicate more elaborate facts between propagators and a CP solver, such as the disjointness of a pair of tasks, as opposed to existing methods focusing on a *local* view.

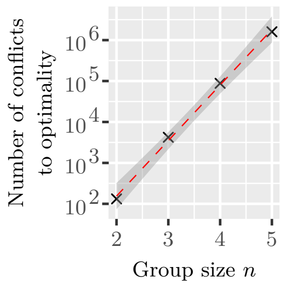
4.1 Motivating Example: $3n$ Problem

Let us first look at a case where reasoning over a single resource is insufficient: Suppose that we are scheduling six tasks, each with unit duration. At every time unit, we are given a fixed amount of resources of types I, II, and III. The amount of resources consumed by each task is illustrated in Figure 2a. *How much time has to pass before all six tasks are completed?*

This is possible to do in six units of time by sequentially scheduling the tasks in any order. What cannot be inferred by reasoning over single constraints (such as time-tabling or energetic reasoning) is that this schedule is the fastest possible (in terms of the latest completion time). To see why, observe that the two yellow tasks cannot be run in parallel due to the shortage of resource I, and the same is true for both the two green tasks and the two blue tasks by similar reasoning. On the other hand, scheduling a yellow and a green task in parallel is not possible, as it would overflow the resource II; the same holds for a green and a blue task, as well as for a blue and a yellow task. By this point, we have considered *all* pairs of tasks and concluded for each of them that they cannot be run in parallel, making it impossible to improve upon the sequential schedule.



(a) Conflict graph of the motivating example. Edge colour indicates resource, solid edges connect tasks jointly consuming 140%, and dotted edges connect tasks jointly consuming 110%. All tasks of the same colour have the same resource usage.



(b) Number of conflicts generated by CP-SAT solving $3n$ problem instances with $d = 1$.

Figure 2 Motivating example: scheduling problem with tasks that are pairwise disjoint due to *various* resource constraints and how CP-SAT performs on these instances.

The previous reasoning relies only on the partition of tasks into three groups such that choosing one task from every two groups implies disjointness; this property holds if we consider groups of size n. We generalise the reasoning above not only in terms of group size n but also by duration d and capacity parameters p, q, M:

▶ **Definition 5.** *Given a group size $n \in \mathbb{N}_{\geq 1}$, duration $d \in \mathbb{N}_{\geq 1}$, and parameters $p, q, M \in \mathbb{N}_{\geq 1}$ with $2p < M < p + q < 2q$, the* ***3n problem instance*** *is the constraint satisfaction problem defined over variables X, Y, Z with $|X| = |Y| = |Z| = n$ and domains $\mathcal{D}(\cdot) \equiv [1, (3n-1)d]$:*

$$\texttt{cumulative}(X + Y; [d]^{2n}; [p]^n + [q]^n; M) \quad \texttt{cumulative}(Y + Z; [d]^{2n}; [p]^n + [q]^n; M)$$
$$\texttt{cumulative}(Z + X; [d]^{2n}; [p]^n + [q]^n; M)$$

As we established, any $3n$ instance is unsatisfiable, but to establish this without search, a solver needs information about all resources. However, each resource individually only implies that any solution occupies some time segment of duration $2n \times d$, and the remaining $(n-1)d$ time units have to be resolved by search. This suggests that any valid sequence of inferences that operates on `Cumulative` constraints separately has exponential length in terms of n.

We prove this, and the $3n$ problem thus serves a role similar to pigeonhole principle formulas [19], but for the `Cumulative` models instead of propositional formulas. We show that any unsatisfiability proof of a $3n$ problem in the C-RES proof system [21] has length exponential in n, which shows that any LCG solver requires an exponential number of steps to prove unsatisfiability, as long as it introduces neither new constraints nor new variables:

▶ **Theorem 6** ($3n$ problem intractability)**.** *Let $\mathcal{P}$ be a C-RES proof of a CSP instance with* ***all*** *clauses valid for any single constraint in Definition 5 for $d = 1$. Then $|\mathcal{P}| = \Omega(1.05^n)$.*

Proof Sketch. First, we show that $\mathcal{P}$ can be encoded, with extra assumptions, as a pigeonhole problem proof over $3n$ variables (Appendix A). Next, we adapt the argument by Beame and Pitassi [4] to verify the exponential lower bound (Appendix B). ◀

We evaluate this insight empirically by running $3n$ instances for $n \leq 10$ and $d \leq 5$ with five solvers: two LCG solvers (Google OR-Tools CP-SAT [31] and Chuffed [9]), a non-learning CP solver (Gecode [17]), and two branch-and-cut solvers (CBC [15] and HiGHS [20]). We ran each solver with a 12-hour time limit, 1 CPU, and 4000 MB of RAM. Both the LCG and non-learning solvers exhibit exponential behaviour, as shown on Figure 2b for CP-SAT, and only one of the runs $n \geq 6$ proved unsatisfiability. These results suggest that LCG solvers indeed resort to exhaustive enumeration on the $3n$ problem. Branch-and-cut solvers have exhibited better performance in solving those instances, demonstrating another similarity with pigeonhole principle formulas. HiGHS solved all instances with $d \leq 3$ in seven minutes. CBC has performed worse, as it has failed to solve instances with $n \geq 7$, however, it has successfully solved $d = 1$ for $n \leq 6$ in less than two seconds.

4.2 `SelectiveDisjunctive` Constraint

Now that we have motivated the necessity for using *global* information, we describe how we use information concerning the disjointness of tasks across *multiple* constraints by introducing the `SelectiveDisjunctive` constraint. The `SelectiveDisjunctive` constraint (i) generalises the disjunctive constraint in a way that allows other propagators to indicate disjoint task pairs, and (ii) restricts the variables indicating this disjointness to be true when a pair of intervals in the assignment is disjoint. We formalise this intuition in Definition 7.

▶ **Definition 7.** *Let T be a set of tasks, and Δ be the $n \times n$ matrix of Boolean variables indexed by T. Then a `SelectiveDisjunctive`(T, Δ) constraint is a condition that for any two tasks $i, j \in T, i \neq j$ either $\delta_{ij} = 0$ or $[s_i, s_i + d_i) \cap [s_j, s_j + d_j) = \emptyset$.*

Introducing these variables and `SelectiveDisjunctive` does not prune any valid solutions, but it allows performing two new operations: (a) fixing the variables from Δ and (b) using these Δ variables in other propagations. This allows us to encode facts – such as 'a pair of tasks cannot overlap given a partial assignment of other tasks' – that are not possible to encode concisely using atomic constraints over the original model variables. More specifically, we can now observe that Definition 7 embeds the standard `Disjunctive` constraint (or even exponentially many of them); to formalise this, we first need the following notion:

▶ **Definition 8.** *Given a search state with domains $\mathcal{D}'$ and a `SelectiveDisjunctive` constraint, a **conflict graph** $\mathcal{G}|_{\mathcal{D}'}$ is a graph with a vertex for each task $i \in T$ and an edge for tasks $\{i, j\}$ such that $\mathcal{D}'(\delta_{ij}) = \{1\}$ (ommitting $\mathcal{D}$ when it is clear from context).*

We can make two key observations based on Definition 8: (1) a conflict graph can be seen as an *exponentially large* collection of `Disjunctive` constraints, and (2) this is also a *dynamically updating* collection of `Disjunctive` constraints, provided that the other parts of a constraint solving engine can propagate the domains of reification variables. On top of this, it suggests that any reasoning for the conventional `Disjunctive` constraint can be deployed for any clique of the conflict graph. In Subsection 4.3, we discuss how to find these cliques in the conflict graph and how to apply the overload checking rule for the `Disjunctive` in this context.

4.3 Overload checking for `SelectiveDisjunctive`

To clarify the intuition behind our inference strategy, consider the following example:

▶ **Example 9.** Let i, j, k be tasks with possible start times $s_i \in [0, 2], s_j \in [0, 2], s_k \in [2, 3]$ and durations $d_i = d_j = 2, d_k = 3$ bound by a `SelectiveDisjunctive` constraint such that $\delta_{ij} = \delta_{jk} = \delta_{ik} = 1$ in the current search state. We can observe that any feasible assignment also satisfies `Disjunctive`$([s_i, s_j, s_k], [d_i, d_j, d_k])$. However, this constraint is infeasible by overload checking for $\{s_i, s_j, s_k\}$: these three tasks jointly cover seven time units, but their EST is 0 and LCT is 6, leaving only six time units among these three tasks.

▶ Note 10. The example above purposefully does not specify *why* the Δ-variables are true: the same reasoning can be carried out as soon as those intervals are established to be disjoint. We discuss the specific strategies for deriving those facts in Subsection 4.4.

We formalise this intuition in Proposition 11 by reformulating Theorem 4 in the context of the `SelectiveDisjunctive` constraint.

▶ **Proposition 11** (Selective overload checking). *Let T be the set of tasks bound by a `SelectiveDisjunctive` constraint, and suppose that $\Omega \subseteq T$ is a clique in the conflict graph $\mathcal{G}$ induced by the current variable domains. Then this constraint is infeasible if $d_\Omega > \mathrm{LCT}_\Omega - \mathrm{EST}_\Omega$, and any feasible solution of the CSP satisfies*

$$\left(\bigwedge_{i,j\in\Omega,\ i\neq j} \delta_{ij}\right) \wedge \left(\bigwedge_{\omega\in\Omega} [\![s_\omega \geq \mathrm{EST}_\Omega - \delta^-]\!]\right) \wedge \left(\bigwedge_{\omega\in\Omega} [\![s_\omega \leq \mathrm{LCT}_\Omega - d_\omega + \delta^+]\!]\right) \implies \perp \quad (3)$$

for arbitrary $\delta^-, \delta^+ \in \mathbb{Z}_{\geq 0}$ satisfying $\mathrm{LCT}_\Omega - \mathrm{EST}_\Omega + \delta^- + \delta^+ < d_\Omega$.

Unlike the `Disjunctive` overload checking, propagation of the `SelectiveDisjunctive` requires an additional step before overload checking can be performed: finding cliques. Finding *a* clique leading to a conflict, even if it exists, is NP-complete[2] and we thus propose a heuristic in Algorithm 1 that dynamically explores some of the cliques present in the conflict graph to balance the trade-off between finding “ good” cliques and the time spent finding them. The heuristic algorithm considers each node as a root for a clique $\mathcal{C}$ and then adds tasks $j \in T \setminus \mathcal{C}$ which minimise $\mathrm{LCT}_{\mathcal{C}\cup\{j\}} - \mathrm{EST}_{\mathcal{C}\cup\{j\}}$ (breaking ties in favour of tasks with a longer duration) while retaining the clique property. The time complexity of Algorithm 1 is $\mathcal{O}(|T|^4)$.

4.4 Disjointness mining

We described how to deploy the `SelectiveDisjunctive` propagator when some of the reification variables have been assigned, but we have not yet described how to infer the domain of these reification variables. In this section, we describe strategies for *disjointness mining*, that is, discovering, possibly during the search, pairs of tasks $i, j \in T$ that are inferred to be disjoint.

Domain disjointness. One direct source of disjointness is the current bounds of variables. The rule can be formally stated as $\forall i, j \in T\ :\ [\mathrm{EST}_i, \mathrm{LCT}_i) \cap [\mathrm{EST}_j, \mathrm{LCT}_j) = \emptyset \implies [\![\delta_{ij} = 1]\!]$. We perform this check during the selective overload checking (Subsection 4.3). Given a propagation of δ_{ij} via this detection, we explain the propagation according to Equation 4.

$$\begin{cases} [\![s_i \leq \mathrm{EST}_j - d_i]\!] \wedge [\![s_j \geq \mathrm{EST}_j]\!] & \text{if } \mathrm{LCT}_i \leq \mathrm{EST}_j \\ [\![s_j \leq \mathrm{EST}_i - d_j]\!] \wedge [\![s_i \geq \mathrm{EST}_i]\!] & \text{otherwise} \end{cases} \implies [\![\delta_{ij} = 1]\!] \quad (4)$$

Difference logic. Scheduling problems often include tasks dependent on the execution of other tasks. These dependencies can be provided in the form of precedence constraints, which provide static information about the disjointness. For example, to encode that j starts δ time units *after* i, we can use the difference logic constraint $s_i + \delta \leq s_j$.

[2] This can be shown by reduction from the maximum clique problem

Algorithm 1 Heuristic conflict discovery procedure.

```
Data: A set of tasks $T$ with intervals $[\mathrm{EST}_i, \mathrm{LCT}_i)$ and durations $d_i$ for all $i \in T$.
Data: A conflict graph $\mathcal{G} = (T, E)$.
Result: A set $\Omega \subseteq T$ satisfying $d_\Omega > \mathrm{LCT}_\Omega - \mathrm{EST}_\Omega$ or $\bot$ if none was found.
if $\exists e \in E : d_e > \mathrm{LCT}_e - \mathrm{EST}_e$ then
    return $e$ ;                                   /* Check for binary conflicts */
for $t_0 \in T$ do
    $\Omega \leftarrow \{t_0\}$ ;                              /* $\Omega$ is a clique containing $t_0$ */
    $S, T \leftarrow \mathrm{EST}_{t_0}, \mathrm{LCT}_{t_0}$ ;            /* $[S, T)$ is an interval covered by $\Omega$ */
    while $d_\Omega \leq \mathrm{LCT}_\Omega - \mathrm{EST}_\Omega$ do
        // Collect tasks that are disjoint with $\Omega$
        $\Omega^+ \leftarrow \{j \in T \setminus \Omega : [S, T) \cap [\mathrm{EST}_j, \mathrm{LCT}_j) \neq \emptyset \wedge \forall i \in \Omega, \{i, j\} \in E\}$;
        if $\Omega^+ = \emptyset$ then
            break ;  /* Cannot extend $\Omega$ into a clique, try another root */
        $t^+ \leftarrow$ an element of $\arg\min_{j \in \Omega^+} \Big(\max(T, \mathrm{LCT}_j) - \min(S, \mathrm{EST}_j), -d_j\Big)$;
        $\Omega \leftarrow \Omega \cup \{t^+\},\ S \leftarrow \min(S, \mathrm{EST}_t^+),\ T \leftarrow \max(T, \mathrm{LCT}_t^+)$;   /* Extend $\Omega$ */
    if $d_\Omega > \mathrm{LCT}_\Omega - \mathrm{EST}_\Omega$ then
        return $\Omega$;
return $\bot$;
```

These constraints encode *stronger* conditions than disjointness; for example, precedence can be seen as disjointness with ordering. Thus, we can deduce the disjointness variables from the difference constraint for appropriate parameter values (this occurs at the root level and requires no explanation). We formalise this in the following proposition:

▶ **Proposition 12.** *A pair of intervals $[s_i, s_i + d_i), [s_j, s_j + d_j)$ is disjoint (i.e., $\delta_{ij} = 1$ in any feasible solution) if any of the following constraints are implied by the CSP: (a) $s_j - s_i \geq \delta$ for some constant $\delta \geq d_i$, or (b) $s_i - s_j \geq \delta$ for some constant $\delta \geq d_j$.*

`Cumulative`. There are several `Cumulative` propagation techniques that reason about when the combination of scheduling a task i and a set of tasks $\Omega \subseteq T \setminus \{i\}$ leads to resource overflows to infer relations between tasks. In this work, we use resource profiles to determine when a pair of tasks is disjoint. An example of this reasoning can be seen in Example 13.

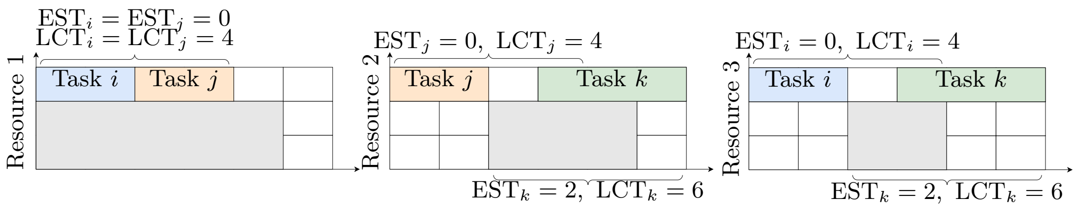

$\mathrm{EST}_i = \mathrm{EST}_j = 0$
$\mathrm{LCT}_i = \mathrm{LCT}_j = 4$
Resource 1
Task i
Task j
$\mathrm{EST}_j = 0,\ \mathrm{LCT}_j = 4$
Resource 2
Task j
Task k
$\mathrm{EST}_k = 2,\ \mathrm{LCT}_k = 6$
$\mathrm{EST}_i = 0,\ \mathrm{LCT}_i = 4$
Resource 3
Task i
Task k
$\mathrm{EST}_k = 2,\ \mathrm{LCT}_k = 6$

Figure 3 Example where profiles (marked in grey) cause disjointness between all tasks i, j, k.

▶ **Example 13.** In addition to the assumptions of Example 9, suppose that the problem has three `Cumulative` constraints as visualised in Figure 3. We can derive from Resource 1 that i and j are disjoint ($\delta_{ij} = 1$); if they are not, then they overlap at some point $\tau^* \in [\max(\mathrm{EST}_i, \mathrm{EST}_j) = 0, \min(\mathrm{LCT}_i, \mathrm{LCT}_j) = 4]$. However, this would cause an overflow of the capacity, implying disjointness. Similar reasoning derives that $\delta_{jk} = 1$ and $\delta_{ik} = 1$.

In this example, just as in the motivating example (Subsection 4.1), neither of the three constraints is infeasible; in fact, each can only imply one disjointness relation. However, the *intersection* of those constraints is correctly declared infeasible by (i) `Cumulative` mining and (ii) `SelectiveDisjunctive` overload checking. Thus, combining the mining strategies with `SelectiveDisjunctive` inferences can generate conflicts that are out of reach for the conventional techniques for the `Cumulative` constraint. ⌟

We adapt the time-table disjunctive reasoning [16] by going over all pairs of tasks $i, j \in T$, and calculating their overlap o_{ij}. Then we check $\forall \tau \in o_{ij}$ if there exists a profile such that scheduling the tasks together with that profile would lead to an overflow. A simple implementation of this approach has time complexity $\mathcal{O}(|T|^2|TT|)$, where TT is the set of existing profiles (i.e, the time-table). This reasoning is formally stated in the following proposition:

▶ **Proposition 14.** *Consider a* `Cumulative` *constraint on tasks T with capacity C. Let the overlap between two tasks $i, j \in T$ be given by $o_{ij} = [\max(\mathrm{EST}_i, \mathrm{EST}_j), \min(\mathrm{LCT}_i, \mathrm{LCT}_j))$. Then δ_{ij} is true for any feasible solution as long as there is no point $\tau \in o_{ij}$ such that $\mathrm{Height}^-(\tau, \{i, j\})$ plus the resource consumption of i and j fits within the capacity:*

$$\forall \tau \in o_{ij} \;:\; r_i + r_j + \mathrm{Height}^-(\tau, \{i, j\}) > C \implies [\![\delta_{ij} = 1]\!]$$

Moreover, given a propagation of δ_{ij} by a set of profiles $\mathcal{P} = \{(a_0, b_0, h_0), ..., (a_m, b_m, h_m)\}$, $a_0 < \cdots < a_m$, we explain the propagation according to Equation 5, assuming without loss of generality that $\mathrm{EST}_i \leq \mathrm{EST}_j$. For $\mathcal{E}_{pr}$, we use the big-step explanation [33].

$$\begin{cases} [\![s_j \geq a_0]\!] \wedge [\![s_j \leq b_m - d_j + 1]\!] & \text{if } \mathrm{LCT}_i \geq \mathrm{LCT}_j \\ [\![s_i \leq b_m - d_i + 1]\!] \wedge [\![s_j \geq a_0]\!] & \text{otherwise} \end{cases} \wedge \left(\bigwedge_{Pr \in \mathcal{P}} \mathcal{E}_{pr}(Pr) \right) \implies [\![\delta_{ij} = 1]\!] \quad (5)$$

The first part of the explanation in Equation 5 is based on the observation that either one task subsumes the entire interval of the other (in which case the explanation depends only on the bounds of s_j), or there is a partial overlap where one task starts at the same time or before the other and ends at the same time or before the other (in which case the overlapping part is only defined by the upper bound on s_i and the lower bound on s_j).

Additionally, we adapt the propagation by Gay et al. [16] and propose a rule using δ in Equation 6 (the rule for updating s_i is symmetric). The rule in Equation 6 and the rule proposed by Gay et al. [16] do not subsume one another since Equation 6 can propagate based on information gained from other constraints, while the rule by Gay et al. can propagate when two tasks are not *fully* disjoint.

$$[\![\delta_{ij} = 1]\!] \wedge \mathrm{ECT}_i > \mathrm{EST}_j \wedge \mathrm{LST}_i < \mathrm{ECT}_j \implies [\![s_j \geq \mathrm{ECT}_i]\!] \quad (6)$$

Nogood. Last, we describe how to derive disjointness from nogoods: given an unsatisfied nogood N, tasks $i, j \in T$ are disjoint if falsifying each of the unassigned atomic constraints implies δ_{ij}. An example can be seen in Example 15 and it is formalised in Proposition 16.

▶ **Example 15.** Given two tasks $i, j \in T$ such that $s_i \in [0, 9], s_j \in [7, 20]$ with durations $d_i = 2, d_j = 5$, consider a nogood $N = \{p_i = [\![s_i \geq 5]\!], p_j = [\![s_j \leq 11]\!],\; ...\}$. Suppose that in the current search state, only the first two atomic constraints are unassigned, while the rest are satisfied. Thus, in any feasible solution either $\neg p_i = [\![s_i \leq 4]\!]$ or $\neg p_j = [\![s_j \geq 12]\!]$ holds. But in either case, we can derive δ_{ij} via domain disjointness:

- If $\neg p_i$ is true, then $\mathrm{LCT}_i = 6 \leq \mathrm{EST}_j = 7$, implying δ_{ij}.
- If $\neg p_j$ is true, then $\mathrm{LCT}_j = 11 \leq \mathrm{EST}_i = 12$, also implying δ_{ij}.

▶ **Proposition 16.** *Let $\mathcal{D}'$ be a set of domains, and suppose that a nogood N has no falsified atomic constraints in this domain. Then a pair of tasks i, j is disjoint if for every atomic constraint $p \in N$ **not** satisfied with respect to $\mathcal{D}'$, δ_{ij} is implied by domain disjointness.*

Our procedure is a simplified version of Proposition 16 based on the two-watcher scheme [28, 27]. Whenever a watcher is updated, we perform a scan to determine whether the conditions of Proposition 16 hold while reasoning over domain disjointness; this misses out on some propagations, but the time required is significantly reduced. The time complexity of this approach is $\mathcal{O}(|N|)$ for each nogood N. We explain this propagation due to an unsatisfied nogood N as $\mathcal{E}_{sn}(s_i, s_j, p_i, p_j) \wedge \left(\bigwedge_{p^+ \in N^+} p^+\right) \implies [\![\delta_{ij} = 1]\!]$, where $N^+ \subseteq N$ is the set of satisfied atomic constraints, and $p_x = [\![s_x \otimes v]\!]$ is one of the two undecided atomic constraints. See Appendix C for the definition of $\mathcal{E}_{sn}$.

5 Experiments

We implemented our approach as part of Pumpkin [12]. First, we present an ablation study in Subsection 5.1, which shows that (a) dynamic disjointness mining hinders the learning of the solver but that it can be beneficial on specific instances, (b) an alternative efficient sorting heuristic has a significant impact on the search but also leading to massive slowdowns on some instances, and (c) enabling the propagation in Equation 6 does not consistently change the performance. Next, we run a comparison on RCPSP and RCPSP/max benchmarks in Subsection 5.2 with baseline Pumpkin and CP-SAT, which shows that our approach can accelerate search by orders of magnitude on certain instances. Additionally, we compare our results with bounds reported in the literature and show that our approach discovers new lower bounds on 16 RCPSP/max instances and four RCPSP instances, as well as new upper bounds on two RCPSP/max instances and four RCPSP instances.

Experiment Setup. The implementation of our approach with the infrastructure for running the experiments is available in the supplementary material. We ran our experiments on *DelftBlue* [11], with each run of an instance being allocated a single core of an Intel Xeon E5-6248R 24C 3.0GHz processor and 4000 MB of RAM with a time limit of one hour.

For our evaluation, we use two models that minimise the latest completion time of all tasks (makespan). One is **RCPSP/max**, a collection of `Cumulative` constraints and precedence relations encoded by the constraints $s_i + \gamma_{ij} \leq s_j$ with *arbitrary* constants $\gamma_{ij} \in \mathbb{Z}$; we use the test suites distributed by PSPLIB [25], namely, C, D, UBO, and SM suites. We also use **RCPSP**, a special case of RCPSP/max where all precedence relations have the form $s_i + d_i \leq s_j$; we use the data files provided in the MiniZinc benchmarks [36], which correspond to the AT [30], BL [3], Pack [8], Pack-d and KSD15-D [26], and PSPLIB [25] suites.

Given a minimisation objective $\mathcal{O}$, each of the solvers is run with one of the two search directions. In the **primal search**, the solver generates a series of problems with an extra assumption $[\![\mathcal{O} \leq o_n]\!]$ for decreasing values of o_n. In the **dual search**, the solver generates a series of problems with an extra assumption $[\![\mathcal{O} \leq o_n]\!]$ for increasing values of o_n.

We discard all the instances solved by the Pumpkin baseline within five seconds in either search direction, leaving us with 736 RCPSP instances and 349 RCPSP/max instances. The models we used and the data files are available in the supplementary material. We evaluate the rate of progress of a solver towards the best-known bound with the following metrics. First, if $M(t)$ is the lowest makespan discovered at time $t \in [0, T]$, and M^* is the lowest discovered makespan for this problem, then the **primal integral** is $\int_0^T \frac{M(t) - M^*}{M(t)}$ and measures how fast the solver progresses towards good solutions [6]. Conversely, if $B(t)$ is the highest lower bound discovered at time $t \in [0, T]$, and B^* is the highest discovered lower bound for this problem, the **dual integral** is defined as $\int_0^T \frac{B^* - B(t)}{B^*}$.

5.1 Ablation Study

To evaluate the influence of individual components on the performance of the whole approach, we ran the disjunctive approach, both in primal and dual search directions, with *all* sixteen combinations of enabling and disabling the following features:

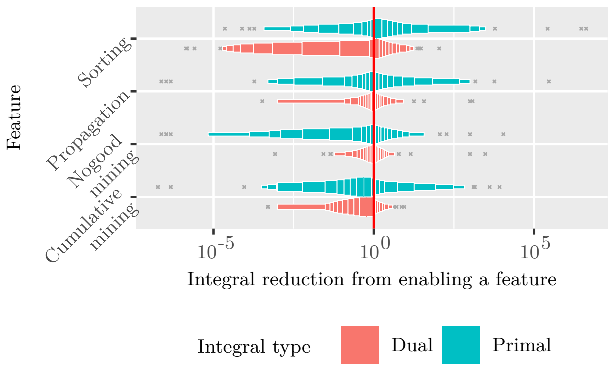
Cumulative mining Enable the mining for `Cumulative` as described in Subsection 4.4.

Nogood mining Enable the mining for nogoods as described in Subsection 4.4.

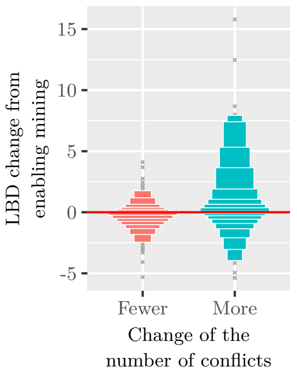
Sorting Restrict Ω^+ in Algorithm 1 to tasks j with $\mathrm{LCT}_j \leq \min_{i \in \Omega} \mathrm{LCT}_i$.

Propagation Execute the propagation shown in Equation 6 on discovered disjunctive pairs.

Figure 4a reports the relative gains from enabling each feature. For each feature, we consider *all* combinations of the remaining three features and evaluate the ratio between the aforementioned integrals when adding the feature to this configuration and *without* the feature.



(a) Distribution of ratio of integrals of enabling each evaluated feature and *not* enabling the feature. The vertical line in the middle corresponds to no observed difference, a ratio larger than 1 corresponds to an improvement by using the feature, and values smaller than 1 correspond to performance degradation.



(b) Change in LBD split by change in # conflicts with cumulative mining compared to baseline; a negative value indicates a decrease in the LBD when using mining.

Figure 4 Impact of enabling the listed features on the solver performance.

One of the major takeaways is that cumulative mining has a predominantly negative impact on search performance. As we understand, this happens because nogood learning derives statements overly specific to the current search state and thus leads to less general nogoods. To support this conclusion, we compare the average literal block distance (LBD) [1] of nogoods produced[3] with and without cumulative mining in Figure 4b, as this metric indicates learned nogood quality. In 47% of all pairs of runs, mining leads to fewer conflicts, while having a relatively small impact on LBD; however, in the other half of run pairs, mining not only increases the number of conflicts but also exhibits a much larger increase in the LBD.

[3] If two runs reached different solutions, we compare the number of conflicts to the best common bound.

On the positive side, we have discovered that cumulative mining *can* be a beneficial strategy for instances where many *triples* of tasks – but few *pairs* of them – are mutually disjoint. In that case, fixing the position of a few tasks can uncover a variety of disjoint pairs, aiding the conflict discovery. In particular, this is the case for several instances from the Pack-d collection; we ran the mining configuration of our approach on those instances and observed that the non-mining implementation discovers few conflicts (and thus makes little progress), whereas the mining implementation discovers several bounds tighter than reported by all the other approaches (see Appendix D).

Another takeaway is that the sorting heuristic has a significant influence on the search. Enabling it reduces integrals for most instances, suggesting that the bottleneck of our approach is the clique search, and most of the iterations of Algorithm 1 do not yield conflicts. However, there is a group of instances for which the *dual* integral degrades by three orders of magnitude or more; for this reason, we only use the sorting heuristic when the disjunctive approach runs in the primal search direction.

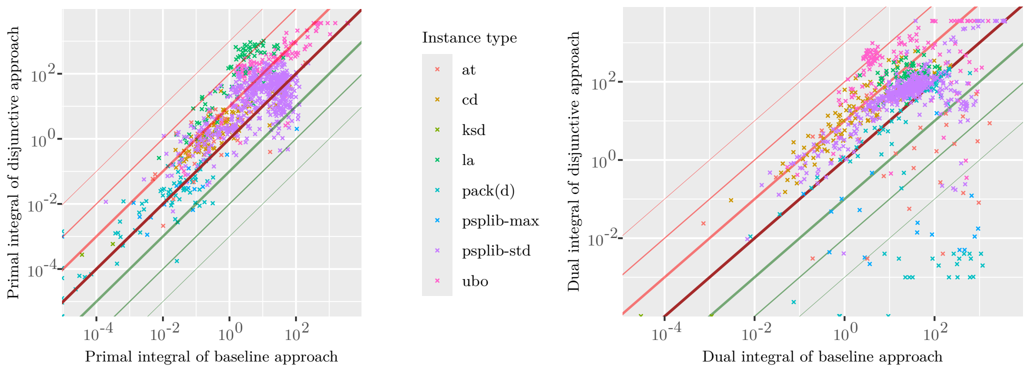
Last, enabling propagation does not exhibit a consistent change in any direction; the primal search can substantially benefit from propagation, but fails to do so consistently, and the dual search is left virtually uninfluenced by the propagation. A similar conclusion holds for using nogood mining in dual search, albeit the performance of primal search becomes consistently worse with nogood mining enabled.

As a result, we use the following configurations: enable sorting and disable propagation and mining for *primal search*, and disable all additional features for *dual search*.

5.2 Scheduling Problem Evaluation

In order to establish whether our approach is beneficial, we compare the disjunctive approach with baseline Pumpkin and CP-SAT v9.11 using the aforementioned integrals.

We start by evaluating the influence of adding disjointness reasoning to the baseline Pumpkin. As seen from Figure 5: (1) the disjunctive approach proves much more successful when using dual search as opposed to primal search, and (2) the disjunctive approach commonly translates to *much* larger improvements in cases where it does not hinder the search, while it can also degrade the performance on a number of instances.



(a) Primal integrals. **(b)** Dual integrals.

Figure 5 Comparison of integrals of baseline and disjunctive approaches. Both axes are logarithmic; lines are evenly spaced and correspond to a tenfold relative change between the integrals.

Figure 5b suggests that the disjunctive approach can accelerate the search by at least *three orders of magnitude*, which indicates that our approach can improve state-of-the-art bounds. Indeed, our approach discovers a variety of lower bounds that are both better than the bounds previously reported and the bounds discovered by all other evaluated approaches. Appendix D introduces a precise criterion for reporting the bounds and the complete list of the novel bounds.

Remarkably, our approach is able to close five of the previously open RCPSP/max J30 instances in *under a second*. For each of those instances, there is a disjoint clique Ω at the root level with d_Ω equal to the optimal bound. In all five cases, these cliques correspond to various resource constraints, making them similar to the aforementioned $3n$ instances; fittingly, CP-SAT has been able to certify only one of them after ten minutes of search, whereas the other instances time out with all other approaches. To clarify this point, we have reproduced and justified such a clique for one of those instances in Appendix E, thereby closing one of the previously open instances without any search.

Both plots in Figure 5 suggest that instance features have a major influence on the change in performance. To establish which instance properties aid our approach, we introduce a measure of similarity of a `Cumulative` constraint to a `Disjunctive` constraint.

▶ **Definition 17.** *For a* ***`Cumulative`*** *constraint over a resource with capacity C, and tasks T, we call the quantity* $\mathrm{RC} = \frac{1}{|T|}\sum_{i \in T} \frac{r_i}{C}$ *the* ***constrainedness*** *of that* ***`Cumulative`***. *Given a CSP, we say that its* ***resource constrainedness*** RC *is the maximum constrainedness among its cumulative constraints. A larger value indicates more resource contention.*

Figure 6 demonstrates the impact of the resource constrainedness: there is exactly *one* instance with an RC under 15% which benefits from using the disjunctive approach, whereas instances with a " scarce" resource (i.e., RC $\geq$ 40%) account for the largest improvements and have the most variability. The performance on the instances between those two values mostly degrades, with a small number of instances seeing minor improvements.

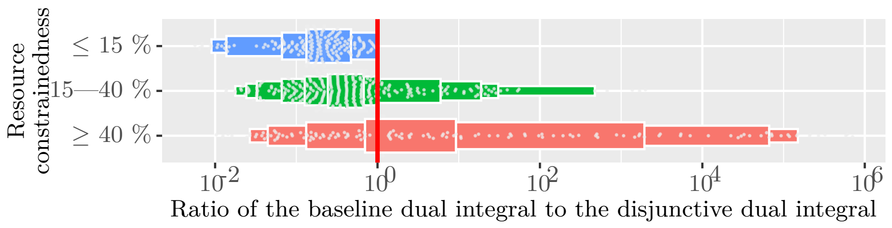
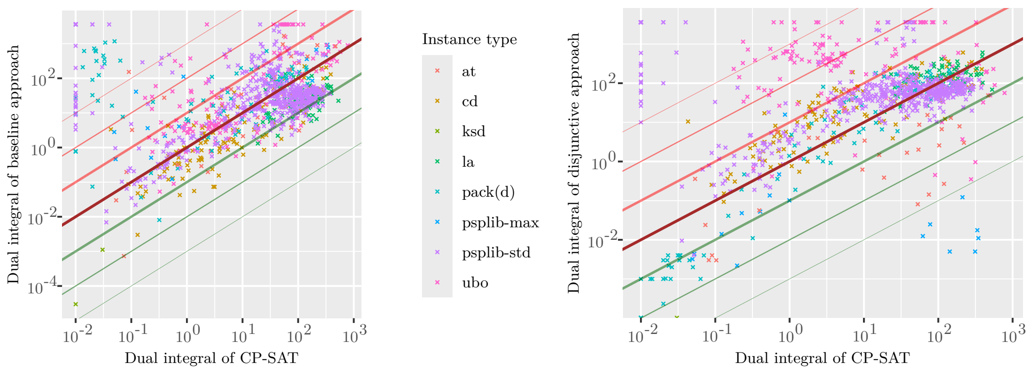


Figure 6 Distribution of dual integral ratio of baseline and disjunctive approach for various values of resource constrainedness. Points to the right of the vertical line correspond to improvements by the disjunctive approach, and points to the left correspond to deterioration.

Last, we evaluate our approach against CP-SAT. Figure 7a shows that the baseline consistently performs better on PSPLIB, with worse performance on pack(d) instances. On the other hand, Figure 7b indicates that the disjunctive approach has a much smaller edge against CP-SAT, yet it can improve certain instances by many orders of magnitude.

6 Conclusions

We have presented a novel way to aggregate information about the disjointness of tasks in scheduling problems across multiple constraints. We do this by creating new variables representing the disjointness between pairs of tasks, mining for this disjointness during search,



(a) Baseline. **(b)** Disjunctive.

Figure 7 Comparison of Pumpkin and CP-SAT dual integrals. All axes are logarithmic; lines on the plots are evenly spaced and correspond to a tenfold relative change.

and heuristically finding disjoint cliques that lead to conflicts. We show that the "standard" approaches require exponentially many conflicts to prove optimality on crafted instances, while our approach can show their infeasibility instantaneously. We also show that our approach provides improvements in the order of magnitudes for benchmark instances with high resource contention, indicating that reasoning across multiple constraints *during search* is of vital importance for inferring the *global* structure of instances.

One direction for future work would be determining what is a *good* clique. Another direction would be to look into incorporating more `Disjunctive` reasoning over the cliques. Additionally, future work could focus on mining *jointness* rather than *dis*jointness.

References

1 Gilles Audemard and Laurent Simon. Refining restarts strategies for SAT and UNSAT. In *Proceedings of the 18th international conference on principles and practice of constraint programming*, pages 118–126, Berlin, Heidelberg, 2012. Springer-Verlag. `doi:10.1007/978-3-642-33558-7_11`.

2 Philippe Baptiste, Claude Le Pape, and Wim Nuijten. Satisfiability tests and time-bound adjustments for cumulative scheduling problems. *Annals of operations research*, 92:305–333, 1999. `doi:10.1023/a:1018995000688`.

3 Philippe Baptiste and Claude Le Pape. Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. *Constraints: an international journal*, 5(1/2):119–139, January 2000. `doi:10.1023/a:1009822502231`.

4 Paul Beame and Toniann Pitassi. Simplified and improved resolution lower bounds. In *Proceedings of the 37th annual symposium on foundations of computer science*, FOCS '96, page 274, USA, 1996. IEEE Computer Society.

5 Nicolas Beldiceanu and Mats Carlsson. A new multi-resource cumulatives constraint with negative heights. In *Proceedings of the 8th international conference on principles and practice of constraint programming*, CP '02, pages 63–79, Berlin, Heidelberg, 2002. Springer-Verlag. `doi:10.1007/3-540-46135-3_5`.

6 Timo Berthold. Measuring the impact of primal heuristics. *Operations research letters*, 41(6):611–614, 1 November 2013. `doi:10.1016/j.orl.2013.08.007`.

7 Peter Brucker, Sigrid Knust, Arno Schoo, and Olaf Thiele. A branch and bound algorithm for the resource-constrained project scheduling problem. *European journal of operational research*, 107(2):272–288, 1 June 1998. `doi:10.1016/s0377-2217(97)00335-4`.

8 Jacques Carlier and Emmanuel Néron. On linear lower bounds for the resource constrained project scheduling problem. *European journal of operational research*, 149(2):314–324, 1 September 2003. doi:10.1016/s0377-2217(02)00763-4.

9 Geoffrey Chu, Peter J Stuckey, Andreas Schutt, Thorsten Ehlers, Graeme Gange, and Kathryn Francis. Chuffed: The chuffed CP solver. https://github.com/chuffed/chuffed.

10 Rina Dechter. Learning while searching in constraint-satisfaction-problems. In *Proceedings of the fifth AAAI national conference on artificial intelligence*, AAAI'86, pages 178–183, Philadelphia, Pennsylvania, 1986. AAAI Press.

11 Delft High Performance Computing Centre. DelftBlue, 2024.

12 Emir Demirović, Maarten Flippo, Imko Marijnissen, Konstantin Sidorov, and Jeff Smits. Pumpkin: a lazy clause generation constraint solver in Rust. https://github.com/ConSol-Lab/Pumpkin, 2024.

13 Ulrich Dorndorf. Consistency tests. In *Project scheduling with time windows*, chapter 4, pages 31–65. Physica-Verlag HD, Heidelberg, 2002. doi:10.1007/978-3-642-57506-8_4.

14 Thibaut Feydy and Peter J Stuckey. Lazy clause generation reengineered. In *Proceedings of the 15th international conference on principles and practice of constraint programming*, CP'09, pages 352–366, Berlin, Heidelberg, 2009. Springer-Verlag. doi:10.1007/978-3-642-04244-7_29.

15 John Forrest, Ted Ralphs, Stefan Vigerske, Haroldo Gambini Santos, John Forrest, Lou Hafer, Bjarni Kristjansson, jpfasano, EdwinStraver, Jan-Willem, Miles Lubin, rlougee, a-andre, jpgoncal, Samuel Brito, h-i-gassmann, Cristina, Matthew Saltzman, tosttost, Bruno Pitrus, Fumiaki Matsushima, Patrick Vossler, Ron @ Swgy, and to-st. coin-or/Cbc: Release releases/2.10.12, 2024. doi:10.5281/ZENODO.13347261.

16 Steven Gay, Renaud Hartert, and Pierre Schaus. Time-table disjunctive reasoning for the cumulative constraint. In *Integration of AI and OR techniques in constraint programming*, Lecture notes in computer science, pages 157–172, Cham, 2015. Springer International Publishing. doi:10.1007/978-3-319-18008-3_11.

17 Gecode Team. Gecode: Generic constraint development environment. http://www.gecode.org, 2006.

18 Vincent Gingras and Claude-Guy Quimper. Generalizing the edge-finder rule for the cumulative constraint. In *Proceedings of the twenty-fifth international joint conference on artificial intelligence*, IJCAI'16, pages 3103–3109. AAAI Press, 9 July 2016. doi:10.5555/3061053.3061056.

19 Armin Haken. The intractability of resolution. *Theoretical computer science*, 39:297–308, 1985. doi:10.1016/0304-3975(85)90144-6.

20 Q Huangfu and J A J Hall. Parallelizing the dual revised simplex method. *Mathematical programming computation*, 10(1):119–142, March 2018. doi:10.1007/s12532-017-0130-5.

21 Joey Hwang and David G Mitchell. 2-way vs. *d*-way branching for CSP. In *Principles and practice of constraint programming - CP 2005*, Lecture notes in computer science, pages 343–357, Berlin, Heidelberg, 1 October 2005. Springer Berlin Heidelberg. doi:10.1007/11564751_27.

22 Roger Kameugne, Laure Pauline Fotso, Joseph Scott, and Youcheu Ngo-Kateu. A quadratic edge-finding filtering algorithm for cumulative resource constraints. *Constraints: an international journal*, 19(3):243–269, July 2014. doi:10.1007/s10601-013-9157-z.

23 Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103, Boston, MA, 1972. Springer US. doi:10.1007/978-1-4684-2001-2_9.

24 Robert Klein and Armin Scholl. Computing lower bounds by destructive improvement: An application to resource-constrained project scheduling. *European journal of operational research*, 112(2):322–346, January 1999. doi:10.1016/s0377-2217(97)00442-6.

25 Rainer Kolisch and Arno Sprecher. PSPLIB - a project scheduling problem library. *European journal of operational research*, 96(1):205–216, January 1997. doi:10.1016/s0377-2217(96)00170-1.

26 Oumar Koné, Christian Artigues, Pierre Lopez, and Marcel Mongeau. Event-based MILP models for resource-constrained project scheduling problems. *Computers & operations research*, 38(1):3–13, 1 January 2011. doi:10.1016/j.cor.2009.12.011.

27 Joao Marques-Silva, Ines Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In *Handbook of satisfiability*, Frontiers in artificial intelligence and applications, chapter 4. IOS Press, 2 February 2021. doi:10.3233/faia200987.

28 Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: engineering an efficient SAT solver. In *Proceedings of the 38th design automation conference*. ACM, 2002. doi:10.1109/dac.2001.935565.

29 Wim Nuijten. *Time and resource constrained scheduling: a constraint satisfaction approach*. PhD thesis, Technische Universiteit Eindhoven, Eindhoven, 1994. doi:10.6100/IR431902.

30 Ramón Alvarez-Valdés Olaguíbel and José Manuel Tamarit Goerlich. Heuristic algorithms for resource-constrained project scheduling: A review and an empirical analysis. *Advances in project scheduling*, pages 113–134, 1989.

31 Laurent Perron and Frédéric Didier. CP-SAT, 7 May 2024.

32 Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of constraint programming*. Foundations of artificial intelligence. Elsevier Science, London, England, 18 August 2006. doi:10.1016/s1574-6526(06)x8001-x.

33 Andreas Schutt, Thibaut Feydy, Peter J Stuckey, and Mark G Wallace. Explaining the cumulative propagator. *Constraints: an international journal*, 16(3):250–282, July 2011. doi:10.1007/s10601-010-9103-2.

34 Andreas Schutt, Thibaut Feydy, Peter J Stuckey, and Mark G Wallace. Solving RCPSP/max by lazy clause generation. *Journal of scheduling*, 16(3):273–289, June 2013. doi:10.1007/s10951-012-0285-x.

35 Peter J Stuckey. RCPSP. https://people.eng.unimelb.edu.au/pstuckey/rcpsp/. Accessed: 2025-3-29.

36 Peter J Stuckey, Thibaut Feydy, Andreas Schutt, Guido Tack, and Julien Fischer. The MiniZinc Challenge 2008–2013. *AI magazine*, 35(2):55–60, 1 June 2014. doi:10.1609/aimag.v35i2.2539.

37 Petr Vilím. $o(n \log n)$ filtering algorithms for unary resource constraint. In Jean-Charles Régin and Michel Rueher, editors, *Proceedings of CP-AI-OR 2004*, volume 3011 of *Lecture Notes in Computer Science*, pages 335–347, Nice, France, April 2004. Springer-Verlag. doi:10.1007/978-3-540-24664-0_23.

38 Petr Vilím. Computing explanations for the unary resource constraint. In Roman Barták and Michela Milano, editors, *Integration of AI and OR techniques in constraint programming for combinatorial optimization problems, second international conference, CPAIOR 2005, Prague, Czech Republic, May 30 - June 1, 2005*, volume 3524 of *Lecture Notes in Computer Science*, pages 396–409. Springer, 2005. doi:10.1007/11493853_29.

39 Petr Vilím. Timetable edge finding filtering algorithm for discrete cumulative resources. In *Proceedings of the 8th international conference on integration of Ai and OR techniques in constraint programming for combinatorial optimization problems*, CPAIOR'11, pages 230–245, Berlin, Heidelberg, 2011. Springer-Verlag. doi:10.1007/978-3-642-21311-3_22.

40 Petr Vilím, Philippe Laborie, and Paul Shaw. Failure-directed search for constraint-based scheduling. In *CPAIOR '15: proceedings of the 12th international conference on integration of AI and OR techniques in constraint programming for combinatorial optimization problems*, Barcelon, Spain, 2015. Springer-Verlag.

A Conversion of $3n$ problem proofs into pigeonhole principle pseudo-proofs

For convenience, we reproduce the definition of the C-RES proof system subject to the notation differences between our paper and the work of Hwang and Mitchell [21].

▶ **Definition 18.** *Given a set of variables $\mathcal{X}$ with domains $\mathcal{D}$, a* ***clause*** *is a constraint of the form* $\left(\bigvee_{j=1}^{N} [\![x_j^+ = v_j^+]\!]\right) \vee \left(\bigvee_{j=1}^{M} \neg[\![x_j^- = v_j^-]\!]\right)$ *with* $x_j^\pm \in \mathcal{X}, v_j^\pm \in \mathcal{D}(x)$.

▶ **Definition 19.** *Given a CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, a sequence of clauses $\mathcal{P}$ is an (unsatisfiability)* ***proof*** *of this CSP if the last clause is empty, and every clause $\omega \in \mathcal{P}$ is produced by one of the following rules:*

Resolution *ω can be represented as $\omega' \vee \omega''$ for some clauses ω', ω'' such that $\omega' \vee [\![x = v]\!]$ and $\omega'' \vee \neg[\![x = v]\!]$ are clauses encountered earlier in the proof. We denote the result of this operation as $\omega' \diamond \omega''$.*

Domain clause *$\omega = \bigvee_{v \in \mathcal{D}(x)} [\![x = v]\!]$ for some variable x.*

Unique value clause *$\omega = \neg[\![x = u]\!] \vee \neg[\![x = v]\!]$ for some variable x and two distinct values $u, v \in \mathcal{D}(x)$.*

Constraint clause *ω is* ***implied*** *by some constraint $c \in \mathcal{C}$, that is, any solution satisfying c also satisfies ω.*

Each step of the proof is either a " consistency" clause encoding that any variable is assigned to exactly one value, a constraint clause (which in our case corresponds to a propagation), or a resolution of two clauses (which typically corresponds to branching or clause learning). The original C-RES definition by Hwang and Mitchell also restricts $\mathcal{C}$ to only contain nogoods, and thus the constraint clauses are the negations of nogoods in $\mathcal{C}$. Again, that difference does not impact the proof definition, because any constraint can be replaced by a set of all nogoods implied by it, which impacts the size of $\mathcal{C}$ *but not the proof length.*

In this appendix, we show that the proofs of $3n$ instances can be encoded as the proofs of the *pigeonhole formula* [19] with $3n$ variables, with clauses encoding the pigeonhole subformulas for up to $2n$ variables. We use the following definition of a pigeonhole problem in this text:

▶ **Definition 20.** *A* ***pigeonhole formula*** PHP_m *is the unsatisfiable propositional formula on $(m-1) \times m$ variables $p_{i,j}, 1 \le i \le m-1, 1 \le j \le m$ having the following clauses:*

Pigeon clauses *Any pigeon is placed in a hole: $p_{1,i} \vee p_{2,i} \vee \cdots \vee p_{m-1,i}$ $\forall 1 \le i \le m-1$.*

Hole clauses *No hole contains two pigeons: $\bar{p}_{i,j} \vee \bar{p}_{i,k}$ $\forall 1 \le i \le m-1, 1 \le j < k \le m$.*

To simplify the description of the encoding, we switch to a more lenient notion of a proof proposed by Beame and Pitassi [4]:

▶ **Definition 21.** *Given a formula PHP_m, an assignment α is called* ***critical*** *if some $(m-1)$ pigeons are assigned to $(m-1)$ distinct holes, that is, $p_{1,\pi_1}, \ldots, p_{m-1,\pi_{m-1}}$ are satisfied by α for some pairwise different $\pi_j \in [1, m], 1 \le j \le m-1$; if additionally the only assigned pigeon has index k, then α is called* ***k-critical****.*

A pair of clauses ω', ω'' is ***congruent*** *if ω' and ω'' are equal for any critical assignment. If any critical assignment satisfying clauses ω' and ω'' also satisfies another clause ρ, then this clause is a* ***critical implication*** *by ω' and ω''.*

One of the immediate simplifications resulting from the notion of congruency is that we can assume without loss of generality that all clauses defined on pigeonhole variables are defined without negations:

▶ **Proposition 22** (Beame and Pitassi [4], Section 3). *Given a formula PHP_m, any clause ω is congruent to a* ***positive clause*** *ω^+ obtained by replacing all literals $\bar{p}_{i,j} \in \omega$ with a negative polarity by the conjunction $\bar{p}_{i,j} \mapsto \bigvee_{k \in [1,m], k \ne i} p_{i,k}$ of variables over all pigeons except the i-th one.*

The core result of this appendix is the reduction from $3n$ proofs to pigeonhole proofs:

▶ **Lemma 23.** *Any proof $\mathcal{P}$ of a $3n$ problem instance can be encoded into a sequence of clauses $\mathcal{P}'$ with $|\mathcal{P}'| \leq |\mathcal{P}|$ such that the final clause is empty and any clause is either a pigeon clause*[4] *of* PHP_{3n}*, a critical implication of previous clauses, or a* ***subproblem*** *clause of the form*

$$\sigma(K, V) := \bigvee_{k\in[1,3n-1]\setminus K, v\in V} p_{k,v}, |V| > |K| \tag{7}$$

with $1 < |V| \leq 2n$.

We start by replacing the original $3n$ problem with a simpler formulation that does not invalidate unsatisfiability proofs of $3n$ problems.

▶ **Definition 24.** *Given a set of variables X, the* `Alldifferent`(X) *constraint is true for the solutions in which any two distinct variables $x, y \in X$ are assigned to different values.*

▶ **Lemma 25.** *Any proof of a $3n$ problem instance is also a proof of a CSP on the same variables and constraints* `Alldifferent`(G) *with $G \in \{X \cup Y, Y \cup Z, Z \cup X\}$.*

Proof. Let $\mathcal{P}$ be a proof of the $3n$ problem instance, and ω is a constraint clause in $\mathcal{P}$ with respect to the original $3n$ instance. We show that this is also a constraint clause with respect to the CSP in the lemma statement. This is sufficient to show that $\mathcal{P}$ is also a valid proof of the new problem, because the resolution steps only depend on the previous clauses. As for the domain and unique value clauses, they only depend on the underlying variable definitions, which are the same between two problem instances, and are also valid for the new problem instance.

Without loss of generality, suppose that a clause ω is implied by the first constraint in Definition 5. Consider an arbitrary solution $\mathcal{I}$ that falsifies ω and thus also falsifies the `Cumulative` constraint in question. Any overflow of such a constraint can be described as either scheduling two tasks from X in parallel, scheduling one task from X and one task from Y in parallel, or scheduling three tasks from Y in parallel. In each case, this assignment violates the `Alldifferent`$(X \cup Y)$. Thus, `Alldifferent`$(X \cup Y)$ is false whenever ω is false, or, in other words, ω is true whenever `Alldifferent`$(X \cup Y)$ is true. ◀

We now complete the reduction to pigeonhole proofs as follows:

Proof of Lemma 23. Introduce a one-to-one mapping between atomic predicates in $\mathcal{P}$ and pigeonhole variables with $[\![x_j = v]\!] \leftrightarrow p_{v,j}$ for some ordering of variables $\mathcal{X} = \{x_1, \ldots, x_{3n}\}$; we further refer to this as the canonical mapping. We show that the proof $\mathcal{P}'$ produced by (i) replacing atomic constraints in C-RES clauses with canonical mapping and (ii) discarding tautologies satisfies the lemma conditions. More specifically, we show that each clause $\omega_P \in \mathcal{P}'$ produced from a C-RES clause $\omega_C \in \mathcal{P}$ either falls into one of the three stated categories or is tautologically true. Additionally, we enforce the following invariant: if a non-tautological clause ω_C is falsified by an assignment $\mathcal{I}$ of $(3n-1)$ variables to $(3n-1)$ different values, then the critical assignment produced from it by canonical mapping falsifies ω_P.

We start with the " consistency" clauses. If $\omega_C = [\![x_k = 1]\!] \vee \cdots \vee [\![x_k = 3n-1]\!]$ is a *domain* clause, then $\omega_P = p_{1,k} \vee \cdots \vee p_{3n-1,k}$ is a valid clause in $\mathcal{P}'$ as a pigeon clause of PHP_{3n}. Additionally, if $\mathcal{I}$ falsifies ω_P, that means that x_k is unassigned; the canonical mapping leaves

[4] Hole clauses are tautological with respect to critical assignments.

k-th pigeon unassigned, which falsifies ω_P. On the other hand, if $\omega_C = \neg[\![x_k = u]\!] \vee \neg[\![x_k = v]\!]$ is a *unique value* clause, then $\omega_P = \bar{p}_{u,k} \vee \bar{p}_{v,k}$ is true for any *critical* assignment and is thus a tautology.

Next, consider the case when ω_C is a *constraint* clause. By Lemma 25, we can assume without loss of generality that $\omega_C \in \mathcal{P}$ is a constraint clause with respect to `Alldifferent`$(X \cup Y)$. Thus, if ω_C is false, then any assignment of $X \cup Y$ assigns the same values to some two variables in this set. By Hall's theorem, that means that there is a set of variables $Q \subseteq X \cup Y$ with indices V and a set of values $K \subset [1, 3n-1]$ such that $|K| < |V| \leq 2n$ and assigning all variables in Q to values in K falsifies ω_C. Conversely, that means that assigning *some* variable in Q to a value outside of K, equivalently, in $\bar{K} = [1, 3n-1] \setminus K$ satisfies ω_C. We have encoded a condition $\bigvee_{v \in V}(x_v \notin K) = \bigvee_{v \in V}(x_v \in \bar{K})$ implying ω_C, which after applying the canonical mapping becomes $\omega_P = \bigvee_{k \in \bar{K}, v \in V} p_{k,v}$ and can be directly added to the proof, since it coincides with $\sigma(K, V)$ from Equation 7.

Last, suppose that $\omega_C = \omega''_C \diamond \omega''_C$ is a *resolution* of some previously introduced propositional clauses ω'_C, ω''_C. Let ω'_P and ω''_P be the corresponding clauses in $\mathcal{P}'$; suppose first that neither of those clauses was skipped. In that case, we observe the encoding ω_P of ω_C is critically implied by ω'_P and ω''_P: otherwise, there would be a critical assignment α satisfying both ω'_P and ω''_P but falsifying ω_P, and applying the canonical mapping to it yields an assignment $\mathcal{I}$ satisfying ω'_C and ω'_C but falsifying ω_C, contradicting the soundness of the resolution step. In case when one (or both) of the original clauses is a tautology, the same reasoning holds, because resolving with a tautological clause does not increase the set of falsifying assignments. ◀

B Exponential lower bound on pigeonhole proofs

We start with an auxiliary fact that establishes a lower bound on the width of a valid proof:

▶ **Lemma 26.** *Let $n \in \mathbb{N}_{\geq 1}, 0 < \gamma \leq 1$, and consider a sequence $\mathcal{P}$ such that any clause in it is either a pigeon clause of* $\mathrm{PHP}_{(2+\gamma)n}$*, a critical implication of the previous clauses, or a subproblem clause $\sigma(K, V)$, $1 \leq |K| < |V| \leq 2n$. Then $\mathcal{P}$ has a clause with at least $2\gamma n^2$ literals.*

Proof. Given a clause ω, let $\mathrm{Complexity}(\omega)$ be the smallest number of pigeon clauses that imply ω on all critical assignments; in particular, $\mathrm{Complexity}(\omega) = 1$ for all pigeon clauses $\omega \in \mathrm{PHP}_{(2+\gamma)n}$, $\mathrm{Complexity}(\bot) = (2+\gamma)n$, and $\mathrm{Complexity}(\sigma(K, V)) = |V|$.

Observe that a proof contains a clause $\hat{\omega}$ of complexity greater than γn and not greater than $2n$. To verify this, let $V^* \leq 2n$ be the largest cardinality of a set V used in a subproblem clause. If $V^* > \gamma n$, then the subproblem clause that achieves this bound also satisfies the complexity bound. Otherwise, assume that this is not the case and that all proof clauses have complexity that is either at most γn or more than $2n$. Observe that if ω is critically implied by ω' and ω'', then $\mathrm{Complexity}(\omega) \leq \mathrm{Complexity}(\omega') + \mathrm{Complexity}(\omega'')$, since the implication relation is transitive. As the formula clauses have complexity 1 and subproblem clauses have complexity at most γn, then *all* their resolvents ω have to have the complexity of at most γn: otherwise, $\mathrm{Complexity}(\omega) \leq \gamma n + \gamma n \leq 2n$ violates our assumption. But then the same bound has to hold for the resolvents of those resolvents, and so on until the empty clause, which has complexity $(2+\gamma)n > 2n$, which is a contradiction.

We complete the proof by demonstrating that this clause satisfies the lemma conditions. Let $\hat{\Omega}$ be the set of formula clauses that implies $\hat{\omega}$ such that $\gamma n < |\hat{\Omega}| \leq 2n$; observe that $\hat{\omega}$ has at least $|\hat{\Omega}|\big((2+\gamma)n - |\hat{\Omega}|\big)$ literals [4, Lemma 1], and this expression achieves the smallest value $2\gamma n^2$ at the endpoints of the interval on $|\hat{\Omega}|$. ◀

We now derive an exponential bound on the proof length of PHP_{3n} with extra clauses:

▶ **Lemma 27.** *Let $n \in \mathbb{N}_{\geq 1}$, and consider a sequence of clauses $\mathcal{P}$ such that any clause is either a pigeon clause of PHP_{3n}, a resolution of the previous clauses, or a subproblem clause $\sigma(K, V)$ with $1 \leq |K| < |V| \leq 2n$. Then $\mathcal{P}$ has at least $2^{0.08n}$ clauses for sufficiently large n.*

Proof. Suppose this is not the case, and we discovered a proof of length $L < 2^{0.08n}$. For the rest of the proof, we say that the clause is *long* if it has at least n^2 literals. By pigeonhole principle, there is a variable $x_{i,j}$ that is contained in at least $\frac{Ln^2}{3n(3n-1)} \geq \frac{1}{9}L$ long clauses. Observe that setting $x_{i,j} = 1$ with $x_{k,j} = x_{i,\ell} = 0$ for all $k \neq i, \ell \neq j$ and discarding all satisfied clauses from both the proof and the formula yields a proof of a PHP_{3n-1} with the same assumptions on subproblem clauses but having at most $\frac{8}{9}L$ long clauses.

Repeating this variable elimination procedure for $T = \lceil \log_{9/8} L \rceil$ steps yields a proof of PHP_{3n-T} with at most $\left(\frac{8}{9}\right)^T L < 1$ long clause, or, in other words, *no* long clauses. On the other hand, we have a pigeonhole formula with $3n - T = 3n - \lceil 0.08n \times \log_{9/8} 2 \rceil \geq \left(2 + \frac{1}{2}\right) n$ variables and a proof with no clauses of length at least n^2. However, we can now rewrite $3n - T = \hat{\gamma}n$ with $\hat{\gamma} \geq 1/2$ and apply Lemma 26, which implies that the proof in question has to have a clause with at least $2\hat{\gamma}n^2 \geq n^2$ literals. That is a contradiction, therefore, $\mathcal{P}$ has to have length of at least $2^{0.08n}$. ◀

The derivation of the intractability theorem bound is a compilation of the results above:

Proof of Theorem 6. Let $\mathcal{P}'$ be the pigeonhole formula proof produced from $\mathcal{P}$ from Lemma 23 with $|\mathcal{P}| \geq |\mathcal{P}'|$. By Lemma 27, we can rewrite the bound further as $|\mathcal{P}| = \Omega(2^{0.08n})$, and the theorem bound follows from the $1.05 < 2^{0.08}$ inequality. ◀

C Explanations

Assume without loss of generality that $\mathrm{EST}_i \leq \mathrm{EST}_j$, as otherwise the tasks i and j can be swapped. Then we generate the explanations $\mathcal{E}_{sn}(s_i, s_j, p_i, p_j)$ as prescribed by Table 1.

Table 1 List of explanations $\mathcal{E}_{sn}(s_i, s_j, p_i, p_j)$ produced for each pair of atomic constraints p_i, p_j.

Atomic predicate p_i	Atomic predicate p_j	Explanation $\mathcal{E}_{sn}(s_i, s_j, p_i, p_j)$
$[\![s_i \leq v_i]\!]$	$[\![s_j \geq v_j]\!]$	$[\![s_j \geq v_i + d_i]\!] \wedge [\![s_i \leq v_j - d_i]\!]$
$[\![s_i \leq v_i]\!]$	$[\![s_j \neq v_j]\!]$	$[\![s_j \geq v_i + d_i]\!] \wedge [\![s_i \leq v_j - d_i + 1]\!]$
$[\![s_i \leq v_i]\!]$	$[\![s_j = v_j]\!]$	$[\![s_j \geq v_i + d_i]\!] \wedge [\![s_i \leq v_j - d_i]\!]$
$[\![s_i \neq v_i]\!]$	$[\![s_j \geq v_j]\!]$	$[\![s_j \geq v_i + d_i - 1]\!] \wedge [\![s_i \leq v_j - d_i]\!]$
$[\![s_i \neq v_i]\!]$	$[\![s_j \neq v_j]\!]$	$[\![s_j \geq v_i + d_i - 1]\!] \wedge [\![s_i \leq v_j - d_i + 1]\!]$
$[\![s_i \neq v_i]\!]$	$[\![s_j = v_j]\!]$	$[\![s_j \geq v_i + d_i - 1]\!] \wedge [\![s_i \leq v_j - d_i]\!]$
$[\![s_i = v_i]\!]$	$[\![s_j \geq v_j]\!]$	$[\![s_j \geq v_i + d_i]\!] \wedge [\![s_i \leq v_j - d_i]\!]$
$[\![s_i = v_i]\!]$	$[\![s_j \neq v_j]\!]$	$[\![s_j \geq v_i + d_i]\!] \wedge [\![s_i \leq v_j - d_i + 1]\!]$
$[\![s_i = v_i]\!]$	$[\![s_j = v_j]\!]$	$[\![s_j \geq v_i + d_i]\!] \wedge [\![s_i \leq v_j - d_i]\!]$

The common idea behind the listed explanations is to lift the bounds in a way that ensures the absence of overlap when one of the two predicates is true. For example, if $p_i = [\![s_i \leq v_i]\!]$ and $p_j = [\![s_j \neq v_j]\!]$, then the first predicate $[\![s_j \geq v_i + d_i]\!]$ in the explanation states the following fact: 'if p_i is true then s_j starts only after task i has ended'. Similarly, the second predicate $[\![s_i \leq v_j - d_i + 1]\!]$ in the explanation states that task i should end before task j starts; we add an extra unit of time since we know that $[\![s_j \neq v_j]\!] \implies [\![s_j \geq v_j + 1]\!]$.

D Novel bounds

We report the bounds discovered by our approach if they are *both* better than the previous reported bounds and are not directly reproducible without using our approach. More precisely, we report bounds that are simultaneously (a) tighter than the bounds reported in the previous sources known to us that used the same benchmarks [25, 40, 39, 35], and (b) either tighter than any bound derived by non-disjunctive approaches (CP-SAT and baseline Pumpkin) or matches it but was derived at least ten times faster than with any other approach.

Novel upper bounds (makespans) are reported in Table 2, and novel lower bounds are reported in Table 3; the same data is available as supplementary materials. All durations reported in both tables are in `MM:SS` format; Table 3 additionally reports closed instances and bounds derived with mining. To indicate the remaining optimality gap, we also state the best known lower bound in Table 2 and the best known makespan in Table 3; in either case, that bound is the tightest among the previously reported values and the values discovered by non-disjunctive approaches.

Table 2 Novel upper bounds derived with our approach.

Problem	Collection	#	Ref. objective	Our objective	Time	Best bound
RCPSP/max	C	61	378	374	07:03	345
RCPSP/max	C	69	380	371	08:47	356
RCPSP	J90	5-4	103	102	01:02	101
RCPSP	J120	27-7	125	124	08:21	122
RCPSP	J120	46-10	188	187	57:34	184
RCPSP	J120	9-4	87	86	06:40	85

E Optimality proof for the RCPSP/max instance #64 of collection J30

In this appendix, we support the optimality claim on instance # 64 from the J30 collection by showing that the tasks with indices $\Omega = \{1, \dots, 9, 11, \dots, 30\}$ are pairwise disjoint, which implies optimality since $\sum_{t \in \Omega} d_t = 169$. This instance has five resources, each with the capacity of five units.

To decrease the number of considered pairs, we partition the claimed clique into six parts and record for each of them the lowest amount of each resource consumed by a task:

1. $\Omega_1 = \{6, 11, 21, 24\}$; lowest resource consumption is $(2, 4, 1, 2, 2)$.
2. $\Omega_2 = \{18, 23, 28\}$; lowest resource consumption is $(2, 1, 2, 2, 4)$.
3. $\Omega_3 = \{30\}$; lowest resource consumption is $(4, 1, 3, 2, 1)$.
4. $\Omega_4 = \{1, 2, 3, 4, 5, 8, 9, 12, 13, 14, 15, 16, 17, 19, 20, 25, 29\}$; lowest resource consumption is $(1, 1, 1, 4, 1)$.
5. $\Omega_5 = \{7\}$; lowest resource consumption is $(3, 5, 1, 1, 2)$.
6. $\Omega_6 = \{22, 26, 27\}$; lowest resource consumption is $(1, 1, 5, 1, 2)$.

The last two groups consume 100% of either resource #2 or resource #3; since all other tasks consume some amount of all resources, any pair of tasks involving a task in $\Omega_5 \cup \Omega_6$ is disjoint. The same holds for any pair of tasks involving Ω_4 and any task in $\Omega_1 \cup \dots \cup \Omega_4$ due to the overflow of resource #4. Thus, it remains to show that any pair of tasks in $\Omega_1 \cup \Omega_2 \cup \Omega_3$ is disjoint. Table 4 handles the six cases for all remaining pairs of task groups.

Table 3 Novel lower bounds derived with our approach. Rows with an asterisk highlight closed instances; rows with a dagger highlight objective values derived with enabled mining.

Problem	Collection	#	Ref. bound	Our bound	Time	Best objective
RCPSP/max	J30	64	141	169*	< 1s	169
RCPSP/max	J30	65	144	162*	< 1s	162
RCPSP/max	J30	151	142	157*	< 1s	157
RCPSP/max	J30	153	163	176*	< 1s	176
RCPSP/max	J30	155	125	154*	< 1s	154
RCPSP/max	UBO50	10	154	186*	00:05	186
RCPSP/max	UBO100	4	303	365	44:07	376
RCPSP/max	UBO100	7	281	373	51:47	395
RCPSP/max	UBO100	8	364	376	29:06	385
RCPSP/max	UBO100	32	353	414	53:52	434
RCPSP/max	UBO100	33	328	397	47:04	406
RCPSP/max	UBO200	35	610	747	57:27	823
RCPSP/max	UBO200	62	621	702	56:46	796
RCPSP/max	UBO500	36	1 285	1 423	26:37	1 908
RCPSP/max	UBO500	61	1 296	1 533	59:50	1 944
RCPSP/max	UBO500	64	1 329	1 489	55:08	1 932
RCPSP	J120	46-9	157	159	18:58	166
RCPSP	Pack-d	2	745	746†	25:39	747
RCPSP	Pack-d	3	624	625†*	13:54	625
RCPSP	Pack-d	47	2 740	2 742†	52:35	2 745

Table 4 Disjointness justifications for every pair of tasks in $\Omega_1 \cup \Omega_2 \cup \Omega_3$.

	Ω_1	Ω_2	Ω_3
Ω_1	#2: $4 + 4$	#5: $2 + 4$	#1: $2 + 4$
Ω_2	—	#5: $4 + 4$	#5: $2 + 4$
Ω_3	—	—	#1: $4 + 4$