


Modeling and Explaining an Industrial Workforce Allocation and Scheduling Problem

Ignace Bleukx ✉ 

Department of Computer Science, KU Leuven, Belgium

Ryma Boumazouza ✉ 

Airbus SAS, Institut de Recherche Technologique Saint-Exupery, Toulouse, France

Tias Guns ✉ 

Department of Computer Science, KU Leuven, Belgium

Nadine Laage ✉

Airbus Aerostructures GmbH, Hamburg, Germany

Guillaume Poveda ✉ 

Airbus SAS, Toulouse, France

Abstract

We present an industrial case on workforce allocation and scheduling in the aircraft manufacturing industry, where available teams need to be assigned to logistical operations. This application presents several challenges such as the scale of the problem, the need for fair workload distribution, and the need for methods for mitigating unforeseen disruptions due to technical malfunctions or incompatible weather conditions. We compare different Constraint Programming (CP) models for the allocation and scheduling problems, with extra focus on modeling the workload balancing component. Additionally, we investigate different techniques for explaining infeasibility of a disrupted schedule, such as conflict computation using Minimal Unsatisfiable Subsets (MUSes) and feasibility restoration using Minimal Correction Subsets (MCSes) or constraint relaxations. Our experimental results show that by using appropriate modeling techniques, the problem can be solved in reasonable time, thereby producing fair schedules. Additionally, we show how invalidated schedules can be explained and restored efficiently to help human operators in solving disruptions to the schedule.

2012 ACM Subject Classification Computing methodologies → Planning and scheduling; Theory of computation → Constraint and logic programming; Human-centered computing → User centered design

Keywords and phrases modeling, scheduling, fairness, explanations, feasibility restoration

Digital Object Identifier 10.4230/LIPIcs.CP.2025.6

Supplementary Material *Model (Source Code and Data)*: <https://github.com/ML-KULeuven/Explainable-Workforce-Scheduling>

Software (Source Code and Data): <https://github.com/airbus/discrete-optimization>

Funding This work was partially supported by the Europe Research and Innovation program TUPLES (101070149). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

1 Introduction

The assignment of workers to different tasks is a recurring challenge in many industries. Effective allocation of workers directly impacts key business metrics such as productivity, cost management, and overall organizational performance. Solving an industrial workforce allocation problem requires considering several factors such as employee availability, required qualifications of workers for a task, regulatory requirements, and task priorities.



© Ignace Bleukx, Ryma Boumazouza, Tias Guns, Nadine Laage, and Guillaume Poveda; licensed under Creative Commons License CC-BY 4.0

31st International Conference on Principles and Practice of Constraint Programming (CP 2025).

Editor: Maria Garcia de la Banda; Article No. 6; pp. 6:1–6:24

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we focus on a workforce-allocation problem in the aircraft manufacturing industry, solving instances involving hundreds of tasks on a daily basis. Each of these tasks consists of a logistical operation such as transporting parts from one place to another or loading and unloading cargo. Transportation tasks are executed using trucks, trailers, or specialized transportation platforms, which require trained workers. Timely execution of these tasks is critical for the production and assembly of aircraft. Currently, given an order book of tasks to be executed by a given deadline, a team of operational planners manually schedules and allocates the tasks to available workers. To alleviate some of the workload of this planning task, we investigate (partially) automating the planning and allocation of tasks by formulating the allocation and scheduling problem as a CP model.

Furthermore, we aim to offer decision support in case of disrupted schedules due to unforeseen circumstances. Examples of such disruptions can be the delay of a part delivery, human or logistical resource disruptions, adversarial weather conditions, etc. We investigate several methods for detecting and explaining the infeasibility of schedules, as well as methods for automatically resolving disruptions using techniques from feasibility restoration [30]. This allows the planners to 1) understand the implications of the disruption on the given schedule and 2) review alternatives provided by an automated system to restore the schedule to a feasible state while accounting for the disruption(s).

1.1 Use case description

We address the problem of workforce allocation for logistical tasks among worker teams at an aircraft manufacturing site. These logistical tasks are interleaved with production activities on assembly lines [4], and need to be aligned with timings of higher-level logistics involving deliveries of raw materials or components provided by external parties via trucks, ships, or cargo aircraft. These components then have to be transported to internal hangars at the right time, where they will be assembled with other aircraft parts. We incorporate these operational requirements into the allocation and scheduling problem as part of the input data. More specifically, each task has a predefined release time, duration, and deadline, and certain tasks follow precedence constraints, such as when two parts must be delivered in sequence before assembly. Apart from explicit precedence constraints, there are no requirements on the task ordering, as the duration of any task includes returning to the home base of the manufacturing plant. As a result, transition time between the end location of one task, and the starting location of the next is included in the duration of each task. This differs from the traditional definition of a workforce-scheduling and routing problem (WSRP) [5] and allows to formulate the problem as a pure scheduling problem instead. In our setting, the goal is to find an allocation of tasks to working teams, such that the total number of required teams is minimized. Additionally, the workload should be balanced fairly among the teams [12]. That is, the allocation of tasks should be done such that all teams have a similar amount of work.

Currently, transportation tasks are manually allocated and scheduled upfront by a team of planners. These planners also coordinate the fine-grained synchronization among teams, to ensure the precedences of tasks are satisfied. During the working day, they continuously monitor real-time updates and adjust schedules as needed to handle unforeseen events or disruptions. These disruptions can arise for multiple reasons, including:

- **Weather-related disruptions:** Extreme wind or rain prevents the use of some hangars.
- **External logistics disruptions:** Delays or cancellations from third-party suppliers impact the arrival time of essential parts.
- **Human resource disruptions:** Unexpected worker absences, team meetings or fire-drills in a hangar temporarily reduces workforce availability.

- **Material availability disruptions:** Vehicles like trucks or trailers may become temporarily unavailable due to technical breakdowns, accidents or maintenance operations.

Large disruptions can impact the predefined plan in such a way that the set of disrupted tasks cannot be reallocated to other teams. This requires the planners to revise the plan, which may involve multiple re-allocation and/or re-scheduling operations. When disruption occurs, given that individual activity execution times range from 15 to 60 minutes, planners have a limited window to adjust schedules. As this is tedious and time-consuming to do by hand, planners often drop disrupted tasks from the current planning horizon and move it to a next one. In order to help the planners revise the allocation in a more efficient way, we propose to automatically repair disrupted schedules and show several alternatives to the planners. This allows them to further adapt or merge proposed alternatives based on their experience or other tacit knowledge.

2 Background

A Constraint Satisfaction Problem (CSP) is defined as triple $CSP(\mathcal{X}, \mathcal{D}, \mathcal{C})$ [27]. Here, \mathcal{X} is set of discrete variables with each an associated set of values (i.e., a domain) in \mathcal{D} . The domain of variable x is written as $\mathcal{D}[x]$. A variable can be finite-domain integer ($\mathcal{D}[x] \subset \mathbb{Z}$) or Boolean ($\mathcal{D}[x] = \{True, False\}$). An assignment α maps each variable x to a value in its domain $\mathcal{D}[x]$. \mathcal{C} is a set of constraints, where each constraint maps an assignment α to True or False. An assignment that is mapped to True by all constraints in the CSP is called a *solution*. We write $\alpha(x)$ for the value of the variable x in solution α . A CSP where no α satisfies all constraints is *unsatisfiable* or *UNSAT* for short.

A Constraint Optimization Problem (COP) extends the definition of a CSP with an objective function f that maps solutions to a numerical value. A solution α is an *optimal* solution if no other solution α' has a better objective value than α .

A Multi-Objective Constraint Optimization Problem further extends the definition of a COP by allowing multiple objective functions to be optimized simultaneously [28, 7, 8]. In case there is a strict priority on the objectives, we refer to it as a Lexicographic Multi-objective Constraint Optimization Problem (LMOCOP). Such problems can be solved by decomposing the LMOCOP into individual COPs and solving these in sequence, and where the objective value of previous COPs is fixed. Example 1 illustrates this process, which can be implemented efficiently with incremental (CP-)solvers.

► **Example 1** (Solving a LMOCOP). Suppose we have a lexicographic optimization problem with objectives $f_1 \prec f_2 \prec \dots \prec f_n$. We can find an optimal solution by first optimizing $COP(\mathcal{X}, \mathcal{D}, \mathcal{C}, f_1)$ to find solution α_{f_1} . Then, we add constraint $f(\mathcal{X}) \leq f(\alpha_{f_1})$ and re-solve the problem with objective function f_2 to find α_{f_2} . This process is repeated until we find α_{f_n} where each objective f_i is optimal, given a value for objective f_{i-1} .

2.1 Scheduling-problems in CP

We consider problems involving scheduling and allocating sets of tasks. The CP-community has developed several *global constraints* for such problems, and in this paper we use the NOOVERLAP (also called DISJUNCTIVE) and CUMULATIVE global constraints. These global constraints reason over uninterruptible tasks, which are represented as *intervals* using an INTERVAL(s, d) variable with start s and duration d . The NOOVERLAP(I) constraint ensures intervals I are scheduled such that they do not overlap. CUMULATIVE(I, D, c) is a generalization of this constraint where a set of intervals with a corresponding set of resource requirements D are scheduled in such a manner that the total capacity c of the resource is never exceeded [29, 32, 33].

An extension to interval variables was introduced by Laborie et al. [21], namely *optional* intervals. Such an interval is declared as $\text{OPTINTERVAL}(s, d, p)$ where the Boolean variable p controls whether the interval (that is the task it represents) is present in the solution.

2.2 Explainable Constraint Solving

For explaining and resolving infeasible schedules due to disruptions, we will rely on techniques from explainable constraint solving. In particular, we compute Minimal Unsatisfiable Subsets (MUS) [23] to explain the user *why* the allocation problem is unsatisfiable after disruptions, and compute Minimal Correction Subsets (MCS) [24] to pinpoint the user to a set of constraints that should be relaxed or removed in order to resolve the conflicts in the problem.

► **Definition 2** (Minimal Unsatisfiable Subset [18]). *A Minimal Unsatisfiable Subset (MUS) is an unsatisfiable subset $U \subseteq \mathcal{C}$ for which it holds that each strict subset $U' \subsetneq U$ is satisfiable.*

Note that several MUSes may exist, and one may have a preference on which conflict to compute [20]. E.g., it is common to compute a *smallest* MUS [19], or *optimal* MUS [15].

► **Definition 3** (Minimal Correction Subset [24]). *A Minimal Correction Subset (MCS) is a subset $M \subseteq \mathcal{C}$ for which $\mathcal{C} \setminus M$ is satisfiable, and for any $M' \subsetneq M$, $\mathcal{C} \setminus M'$ is unsatisfiable.*

Similar to MUSes, a problem may contain multiple MCSes. To find an optimal MCS, we can reify each constraint and minimize the number of falsified indicator variables.

3 Formal specification of the constraint models

We now introduce the mathematical formulation of the workforce-allocation COP. Section 3.1 focuses on the pure-allocation setting where each task has a fixed start-and end-time. In Section 3.2, we investigate how this formulation can be adapted to a scheduling-and-allocation formulation where each of the tasks can be shifted within a given time-window.

3.1 Allocation

This problem consists of assigning each task to a compatible team. Teams can be (in)compatible with a task due to their working hours, or due to specific technical qualifications required to execute the task. No team can execute two tasks at once, and hence overlapping tasks should be assigned to different teams. Finally, some tasks are required to be assigned to the same team, e.g., when two tasks involve transporting the same part from one place to another.

Input data

- \mathcal{T} : a set of n tasks to assign to teams.
- \mathcal{W} : a set of m worker-teams to assign tasks to.
- $\text{Comp} \in \mathbb{B}^{n \times m}$ where $\text{Comp}_{t,w}$ indicates team w and task t are compatible.
- S : the fixed start times of each task, with $s_i \in S$ the start time of task i .
- D : the fixed durations of each task, with $d_i \in D$ the duration of task i .
- \mathcal{G} : set of task groups where tasks in group $g \subseteq \mathcal{T}$ should be executed by the same team.

Decision variables and constraints

The variables in this model consist of a $n \times m$ Boolean matrix where each variable $x_{t,w}$ indicates whether team w is assigned to task t .

Each task is allocated to a team:

$$\sum_{w \in \mathcal{W}} x_{t,w} = 1 \quad \forall t \in \mathcal{T} \quad (1)$$

Tasks are only assigned to compatible teams:

$$\neg \text{Comp}_{t,w} \Rightarrow \neg x_{t,w} \quad \forall t \in \mathcal{T}, \forall w \in \mathcal{W} \quad (2)$$

A given group of tasks g should be assigned to the same team:

$$\text{ALLEQUAL}(\{x_{t,w} \mid t \in g\}) \quad \forall g \in \mathcal{G}, \forall w \in \mathcal{W} \quad (3)$$

A team cannot be allocated to overlapping tasks:

$$\sum_{t' \in \mathcal{T}}^n x_{w,t'} \cdot (s_t \leq s_{t'} \wedge s_t + d_t > s_{t'}) \leq 1 \quad \forall t \in \mathcal{T}, \forall w \in \mathcal{W} \quad (4)$$

Auxiliary variables and objectives

We introduce auxiliary variables $used_w \in \mathbb{B}^m$ that indicate whether team w is used, and integer variables $time_w \in \mathbb{N}^m$ that indicate the time worked by each team. These auxiliary variables relate to the decision variables x according to the following constraints:

A team is used if a task is allocated to it:

$$\bigvee_{t \in \mathcal{T}} x_{t,w} \Rightarrow used_w \quad \forall w \in \mathcal{W} \quad (5)$$

Definition of the time worked by a team, only enforced if the team is used:

$$used_w \Rightarrow \sum_{t \in \mathcal{T}} x_{t,w} \cdot d_t = time_w \quad \forall w \in \mathcal{W} \quad (6)$$

The objectives of the problem are to minimize the number of teams used: $\sum_{w \in \mathcal{W}} used_w$ and to minimize the time-dispersion as described next.

Modeling improvement: different formulations for time-dispersion

Tasks should be distributed *fairly* among the working teams. That is, we aim to minimize the *dispersion* of the workload which we define as: $\max_{w \in \mathcal{W}}(time_w) - \min_{w \in \mathcal{W}}(time_w)$ and corresponds to a *spread-based* fairness measure from the literature [6, 12]. However, we noticed that combinatorial solvers are often stuck on proving optimality when using this objective. Therefore, in Sec. 5.1, we compare the following formulations of the time-dispersion:

- **max_minus_min**: exact formulation: $\max_{w \in \mathcal{W}}(time_w) - \min_{w \in \mathcal{W}}(time_w)$
- **max_diff**: exact formulation, with pairwise differences $\max_{w,w' \in \mathcal{W}}(time_w - time_{w'})$
- **proxy_max_min**: maximize the minimum time worked: $-\min_{w \in \mathcal{W}}(time_w)$
- **proxy_min_max**: minimize the maximum time worked: $\max_{w \in \mathcal{W}}(time_w)$
- **proxy_sum**: minimize the sum of absolute differences $\sum_{w \in \mathcal{W}, w' \in \mathcal{W}}(|time_w - time_{w'}|)$

Modeling improvement: symmetry breaking over equivalent teams

Although each team has a specific set of qualifications that result in (in)compatibility with certain tasks, some teams are still equivalent for the given time-horizon of the schedule. Therefore, we can detect and break symmetries between teams. We do this by detecting sets of teams that can be assigned to the exact same set of tasks in the input data. For every consisting set of equivalent teams $\{t_1, t_2, \dots, t_p\}$, we add the following constraint:

$$used_{t_1} \leq used_{t_2} \leq used_{t_3} \leq \dots \leq used_{t_p}$$

Comparison to literature

The allocation problem described in this paper is related to the “list coloring” problem, a variant of the well-known graph-coloring problem where each node can only be assigned a subset of colors. However, in addition to minimizing the number of teams (colors) used, we also aim to minimize the time dispersion. This is specific to our use case and, to the best of our knowledge, it is not considered in the literature of similar list-coloring problems. Constraint (4) would implement a “clique” constraint on a list coloring problem. In our case, these cliques can easily be detected due to the structure of the problem.

3.2 Scheduling

In the second variant of the problem, the tasks are not yet fixed in time and should be scheduled in a predefined time-window. That is, each task has a earliest time it can be started (the release time), and a deadline by which it should be completed. Moreover, some tasks depend on one another and therefore share a precedence constraint. Also, each of the worker teams has fixed working-hours and hence, any task assigned to a given team, must start and finish within the working hours of that team.

Additional data

- \mathcal{R} : the release time of each task, with $r_t \in \mathcal{R}$ the release time of task t .
- \mathcal{E} : the deadline of each task, with $e_t \in \mathcal{E}$ the deadline of task t .
- $Prec$: a set of tuples (t, t') where task t should finish before the start of t' .
- Cal_w : a set of tuples (c_s, c_d) that represent intervals during which team w is off-duty.

Additional variables and constraints

Our CP-model for this problem is inspired by flexible-jobshop problems [1, 9], and the main idea of this model is to duplicate each task for each team using an optional interval variable. Each interval is defined using a new variable $s_t \in \mathbb{N}$ for representing the start time of each task, and we use the Boolean matrix x to control which intervals are indeed present in the schedule, i.e., which intervals (tasks) are allocated to what team.

The constraint model builds further on the “pure allocation” formulation above. That is, we re-use constraints (1), (2) and (3), but replace constraint (4) with constraint (7), which ensures that no tasks assigned to the same team overlap. Additionally, constraint (7) ensures no tasks are assigned during the “off-hours” of a team as defined by their calendar. Finally, we add constraints (8) and (9) to enforce a legal schedule of the tasks.

Teams cannot perform overlapping tasks and can only work when available.

$$\text{NOOVERLAP}(\{\text{OPTINTERVAL}(s_t, d_t, x_{t,w}) \mid t \in \mathcal{T}\} \cup \{\text{INTERVAL}(c_s, c_d) \mid (c_s, c_d) \in \text{Cal}_w\}) \quad \forall w \in \mathcal{W} \quad (7)$$

Tasks are scheduled within their allowed time time-frame:

$$s_t \geq r_t \wedge s_t + d_t \leq e_t \quad \forall t \in \mathcal{T} \quad (8)$$

Enforce precedences between tasks:

$$s_t + d_t \leq s_{t'} \quad \forall (t, t') \in \text{Prec} \quad (9)$$

Typically, the objective in flexible job-shop scheduling is to minimize the makespan of the schedule. However, in this problem, we re-use the objectives from Section 3.1 and aim to minimize the number of teams used and to minimize the dispersion.

Modeling improvement: computing a lower-bound on the number of required teams

The constraint model above can be solved for large instances by state-of-the art constraint solvers. However, we noticed that when minimizing the number of required teams, the solver is often stuck on proving optimality of the solution. We want to help the solver by enforcing a reasonable lower-bound on the number of teams. To compute this lower-bound, we propose the following relaxation of the problem, where the pool of working teams is modeled as a cumulative resource:

$$\text{CUMULATIVE}(\{\text{INTERVAL}(s_t, d_t) \mid t \in \mathcal{T}\}, c)$$

In this relaxation, each team is treated as equivalent, and hence the relaxation does not consider task-compatibility requirements, nor the availabilities of each team. As we will see in Section 5.2, computing a solution that minimizes the capacity c , is often much faster than solving the full problem. We add the lower-bound on c found by the solver to the full-problem as a constraint, which can help the solver prove optimality faster.

Comparison to literature

The scheduling problem considered in this paper is most closely related to the Flexible-Jobshop scheduling problem (FJSP), where a set of tasks is to be scheduled and assigned to a set of resources (e.g., machines). However, compared to the flexible-jobshop problem, we also consider the constraint where some tasks have to be assigned to the same team (resource in FJSP). Moreover, in jobshop-scheduling, it is usually the goal to minimize the makespan; this is not the case for our use case. We consider as primary objective to minimize the number of resources (teams) required to complete the schedule. Additionally, we ensure a fair distribution using the time-dispersion objective. While load balancing is considered in FJSP literature, it is more common to minimize the maximum workload of a machine given the time-horizon as not to wear out any machines more than others. This is different from our objective and corresponds to the `proxy_min_max` formulation from Section 3.1.

4 Interactive and explainable allocation

In this section, we outline our efforts to develop an explainable decision-making tool for allocation of tasks to teams, with the potential for generalization to similar use cases. Our primary application of explanation techniques will be to handle disruption scenarios, which invalidate the precomputed schedule and require the planner to re-allocate and/or re-schedule the set of tasks. Section 4.1 outlines several types of disruptions and Section 4.2 investigates what types of explanation techniques can be useful for the planners.

4.1 Disruption scenarios

From the planner's perspective, the scenario is as follows: an initial schedule and allocation is established for upcoming shifts, with an estimate of available teams. This schedule can be computed using the CP-models from the previous section or manually constructed by the planners. As discussed in Section 1.1, various disruptions can affect the plan and possibly make the foreseen allocation infeasible. In response, the planner must first understand the implications of the disruption, and then adapt and reallocate the available resources such that as many tasks as possible can still be allocated, without wasting resources.

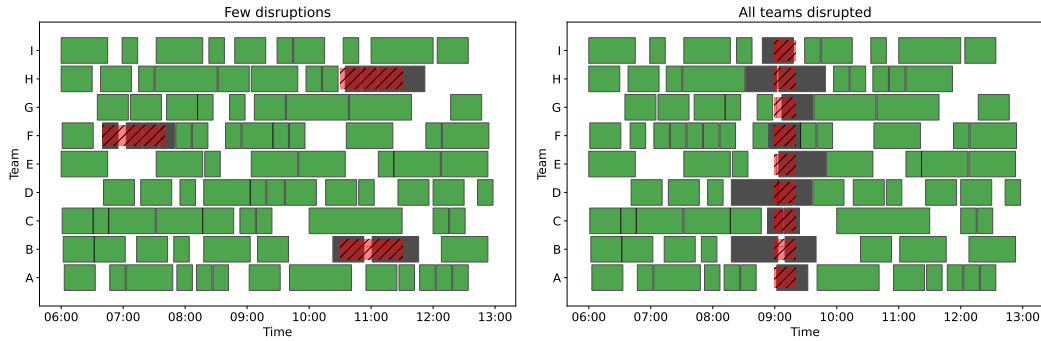
To evaluate our explanation methods in a broad range of disruption scenarios, we use a generator with adjustable parameters for simulating disruptions. These generator parameters are designed in coordination with the real planners, to cover a range of realistic and complex scenarios that arise in their daily practice. Table 1 details the parameters used for generating resource disruption scenarios based on three key factors: the number of disruptions, the number of affected teams, and the duration of the disruptions. As an illustration, the **all teams affected** scenario corresponds to a situation where all working teams are unavailable for a given amount of time, which can occur during extreme weather conditions. In less extreme cases, such as the **few disruptions** scenario, some teams are unavailable for a period of time caused by team meetings, transport accidents, or unexpected worker absences. A single disruption can affect multiple teams when they need to share a certain resource in the manufacturing site, e.g., when a hangar has to be evacuated due to a fire-drill. Figure 1 visualizes two instances of disruptions.

■ **Table 1** Parameters for generating disruptions of each scenario type.

Scenario description	No. of disruptions	No. of teams affected	Duration (Minutes)
few disruptions	$\{1 \dots 3\}$	$\{1 \dots \frac{ \mathcal{W} }{2}\}$	$\{15, 30, 60, 120\}$
long disruptions	$\{1 \dots 3\}$	1	$\{120, 240, 360\}$
many disruptions	$\{5 \dots 10\}$	$\{1 \dots \mathcal{W} \}$	$\{15, 30, 60, 120\}$
many teams affected	1	$\{\frac{ \mathcal{W} }{2} \dots \mathcal{W} \}$	$\{15, 30, 60, 120\}$
all teams affected	1	$ \mathcal{W} $	$\{15, 30, 60, 120\}$
one team dropped	1	1	Horizon
two teams dropped	1	2	Horizon
delay	-	-	$Poisson(\lambda = 5)$

After a disruption in the schedule has occurred, the planners will try to reallocate tasks to other, free teams. However, in practice, this is often not possible as task-compatible teams are usually already occupied with other tasks. In the next two sections, we explore how to assist the planner in resolving these disruptions.

► **Example 4** (Example disruptions to a schedule). Figure 1 illustrates two classes of disruption scenarios. Disruptions are shown as striped boxes, and the affected tasks are colored black.



■ **Figure 1** Examples of different disruption scenarios on the same base instance of 1 working shift.

4.2 Conflict detection: explaining why the allocation is unsatisfiable

In this first setting, we aim to explain to the planner *why* it was not possible to reallocate all the tasks affected by the disruption. That is, we compute a minimal *conflict* in the constraints by using a Minimal Unsatisfiable Subset (MUS)-extraction algorithm. As is typical in MUS-extraction, we split the problem constraints into a set of *foreground* and *background* constraints [18]. Foreground constraints are those that are explicitly presented as part of the conflict while background constraints are assumed to always hold. In our setting, we treat constraints that require tasks to be allocated (1), constraints that require tasks to be allocated to the same team (3) and no-overlap constraints (4) as foreground constraints. The remaining constraints are background, in combination with an encoding of the disruptions. As a result, a MUS corresponds to a minimal set of overlapping tasks that cannot be executed given the available teams. Section 6.1 experimentally compares the SHRINK and SMUS algorithms for computing MUSes for this setting. SMUS computes the smallest MUS in terms of the number of constraints [19], while SHRINK computes an arbitrary MUS.

Figure 2a visualizes a set of *task is allocated* (3) constraints that together form a MUS. Indeed, this irreducible set of tasks cannot be allocated given available teams and disruptions.

4.3 Feasibility restoration: explaining how to revise the model

After understanding the conflict in the allocation problem, the next step is to revise the model such that a new schedule can be computed. Model diagnosis [26] can be used to iteratively resolve all conflicts in the problem by sequentially resolving each individual MUS. Indeed, a conflict represented by a MUS is resolved after the removal of one of its constraints. However, by solving each conflict locally, in the end, more constraints may be removed from the problem than strictly necessary. Therefore, we investigate two types of feasibility restoration methods that can globally resolve all conflicts in the model.

4.3.1 Removing constraints with Minimal Correction Subsets

In this first approach to resolving disruptions, we compute a Minimal Correction Subset (MCS). Such a correction subset presents the planner with a minimal set of constraints that should be removed, so that the resulting model will admit a solution. Similar to computing

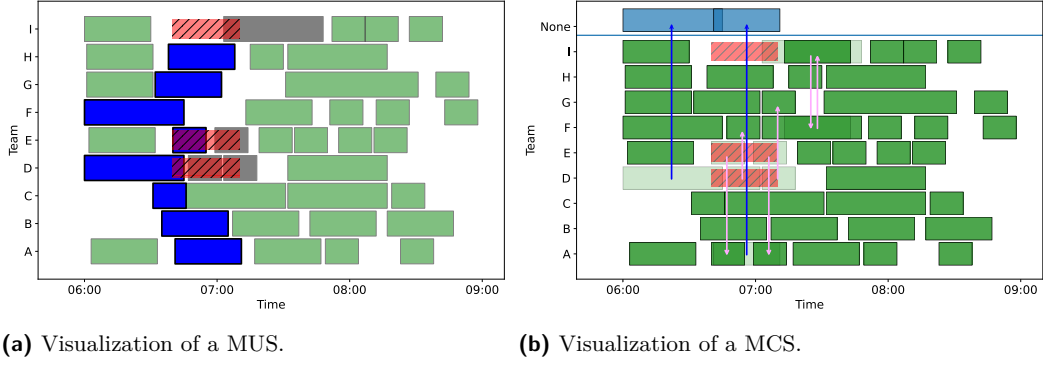


Figure 2 Visualization of two explanation methods for the same working shift and disruption.

a MUS, the model is split into foreground and background constraints. Here, the foreground constraints are those that can be removed, while the background constraints are always enforced. In this setting, we only allow to remove tasks from the schedule (1) or remove *same-allocation* constraints (3). All other constraints are considered background. Figure 2b visualizes an MCS and the resulting solution where two tasks (in blue) can be dropped from the schedule in order to resolve the disruption. Section 6.1 experimentally investigates two types of MCS-computation algorithms: we explore a greedy GROW-based algorithm to find an arbitrary MCS, and a reified MAX-CSP formulation for finding an optimal MCS.

4.3.2 Rescheduling using multi-objective feasibility restoration

Using MCS-based feasibility restoration can cause drastic changes to the model [3]. Indeed, as constraints (or tasks) are removed from the model in order to restore feasibility, the resulting solution may drastically change compared to the initial assignment. Moreover, a disruption may only overlap with a task for a short period of time and, instead of removing the task, it could be slightly shifted to a later time. In this section, we investigate such fine-grained alterations to the schedule, which are not possible by computing MCSes on the infeasible allocation model. That is, we convert the original allocation problem to a allocation-and scheduling-formulation as outlined in Section 3.2. Then, based on the original assignment α , which is now infeasible, we define several *similarity* objectives. These objectives serve to ensure the newly computed solution is as close as possible to the original assignment [10, 14, 30]. We define 5 such similarity objectives:

Maximize the amount of tasks that are still allocated to a team (**#done**):

$$\text{maximize} \quad \left(\sum_{t \in \mathcal{T}} \left(\sum_{w \in \mathcal{W}} x_{t,w} \right) \right) = \left(\sum_{w \in \mathcal{W}} \alpha(x_{t,w}) \right) \quad (10)$$

Minimize the tasks that are re-allocated to a different team (**#realloc**):

$$\text{minimize} \quad \sum_{w \in \mathcal{W}} \sum_{t \in \mathcal{T}} (\alpha(x_{t,w}) \neq x_{t,w}) \quad (11)$$

Minimize the number of shifted tasks (**#shift**).

$$\text{minimize} \quad \sum_{t \in \mathcal{T}} (\alpha(s_t) \neq s_t) \quad (12)$$

Minimize the total shift in time of the tasks (**sum-shift**)

$$\text{minimize} \quad \sum_{t \in T} |\alpha(s_t) - s_t| \quad (13)$$

Minimize the maximum shift in time of the tasks (**max-shift**)

$$\text{minimize} \quad \max_{t \in T} |\alpha(s_t) - s_t| \quad (14)$$

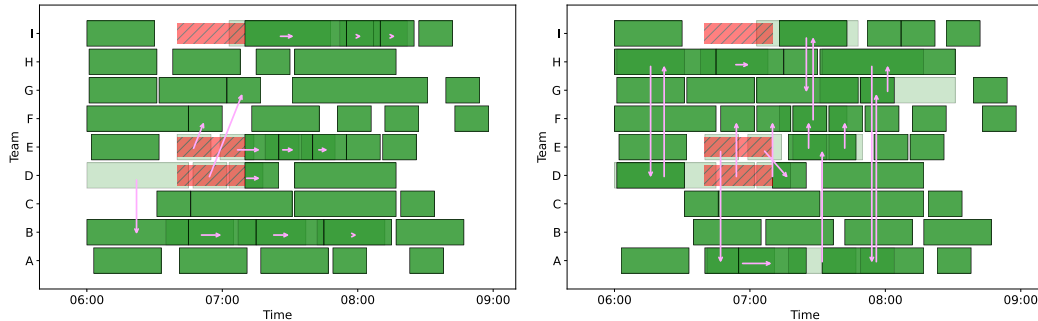
Instead of optimizing only one of these similarity objectives, we propose to combine these into a LMOCOP. This allows us to adjust the ordering of objectives, which, after solving with techniques similar to Example 1, produce differently repaired schedules after disruptions. In Table 2, we propose several orderings of the similarity objectives that can be used as sub-objectives to the LMOCOP.

■ **Table 2** Optimization orders used for feasibility restoration with rescheduling.

Optimization order	Lexicographic ordering
Reallocation first (A)	#done < # <u>realloc</u> < #shift < sum-shift < max-shift
Shift first (B)	#done < # <u>shift</u> < sum-shift < max-shift < #realloc
Sum of shifts first (C)	#done < <u>sum-shift</u> < max-shift < #shift < #realloc
Max-shift first (D)	#done < <u>max-shift</u> < sum-shift < #shift < #realloc

Figure 3 shows two repaired schedules for different orders of the sub-objectives on the same instance as Figure 2. Depending on the ordering of the similarity objectives, a different trade-off between re-allocating and re-scheduling tasks is found by the solver.

By optimizing the LMOCOP for each lexicographic ordering, we can present the decision maker with multiple alternatives on how to resolve the disruption; as can be seen, many of these resolutions go further than shifting or re-allocating single tasks, and repairing the allocation can involve a chain of moves in order to ensure all tasks can be allocated to a non-disrupted team; which is tedious and time-consuming for planners to do by hand.



(a) Using order A: #realloc = 3, #shift = 12. (b) Using order B: #realloc = 14, #shift = 4.

■ **Figure 3** Repaired schedule after disruptions using different optimization orders.

5 Computational results on solving the allocation/scheduling problem

In this section, we investigate the efficiency of the different modeling approaches for solving the allocation/scheduling problem and Section 6 investigates the efficiency of the explanation techniques. More concretely, this section aims to answer the following questions:

- EQ1.** How do Constraint Programming (CP), Integer Linear Programming (ILP) and Pseudo-Boolean (PB) solvers compare for solving the allocation problem?
- EQ2.** Which objective function yields the best fairness in terms of time-dispersion?
- EQ3.** What is the effect of adding a lower-bound from the problem relaxation in the scheduling problem?

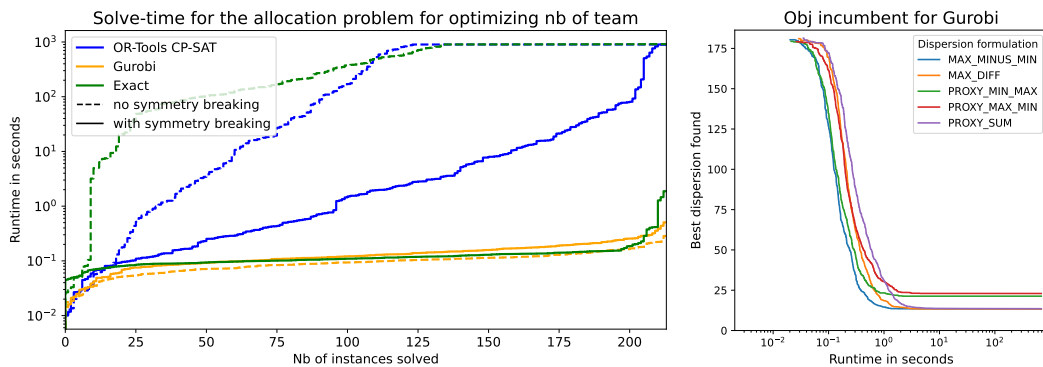
We implement the models of Section 3.1 and 3.2 in the CPMpy [17] library version 0.9.23 and use as underlying solvers OR-Tools CP-SAT v.9.11; the ILP-solver Gurobi v12.0.0 and the Pseudo-Boolean solver Exact v2.1.0 [11, 13]. The experiments in this section were ran on a single thread of a Intel(R) Xeon(R) Silver 4514Y and 256GB of RAM. For the allocation problem, we set OR-Tools to not use any sub-solvers to evaluate a pure CP-approach of solving the problem and for the scheduling problem, we allow it to use 8 threads. We consider instances of 8h horizons over a period of 3 months. This corresponds to one shift of the worker-teams and an average number of 67.6 tasks to be scheduled and allocated. All instances and models are made publicly available on <https://www.github.com/ML-KULeuven/Explainable-Workforce-Scheduling>.

5.1 Solving the allocation problem

We first compare different solvers and models for allocating tasks with *fixed* start-times.

Minimizing the number of teams

In Figure 4, left side, we plot the number of instances solved to optimality by each solver, with and without symmetry breaking as described in Section 3.1. We notice that both CP and PB-solving can benefit greatly from breaking symmetries in the model. Indeed, both OR-Tools and Exact can solve ± 125 instances when symmetry breaking is disabled. However, by imposing an order on the equivalent teams for the given time horizon, OR-Tools solves ± 200 instances, and Exact solves all instances in our dataset. Interestingly, we notice that Gurobi does not benefit from breaking the symmetries in the model, and the performance slightly degrades instead. Still, in both settings, Gurobi solves all instances in < 1 seconds.



■ **Figure 4** Computational results on the allocation use-case (time-horizon of 8h).

Minimizing time-dispersion

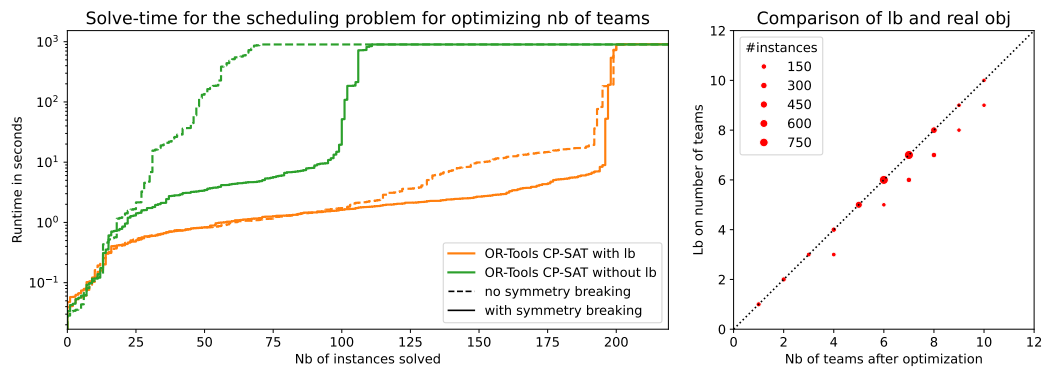
We found that computing optimal solutions for the time-dispersion using the straightforward formulation $\max_{w \in \mathcal{W}}(time_w) - \min_{w \in \mathcal{W}}(time_w)$ is a challenging problem, and solvers are often stuck proving optimality. Indeed, as shown in Section A, no solver was able to prove optimality for more than 30% of the instances. In Figure 4, right side, we plot the *true* dispersion during optimization for each of the dispersion-formulations outlined in Section 3.1 for Gurobi with results for other solvers in Section A. Interestingly, while Gurobi does not prove the optimality of the solution, the time-dispersion drops to $\pm 15min$ on average within 10s of solve-time. Furthermore, although more instances using the proxy-metrics are solved to optimality, this does not actually contribute to a better overall time-dispersion. From this experiment, we can conclude that although optimality is not proven for most instances, exact formulations of the time-dispersion work well for allocating the tasks fairly among the teams.

5.2 Solving the scheduling problem

We now turn to the much more computationally intensive case of *scheduling*, where tasks are not yet fixed in time. We only use OR-Tools CP-SAT as it is the only solver we tested with support for the appropriate global constraints. Figure 5, left side, shows the runtime for optimizing the number of teams in the scheduling problem. While OR-Tools is able to solve only ± 100 instances to optimality in the bare-bones model, adding the lower-bound of the relaxed problem greatly improves the performance of the solver and ± 100 extra instances are solved. The right-side of Figure 5 reveals that the lower-bound found by the solver is close to or equal to the true optimal number of teams required to allocate all tasks and hence, by adding the lower-bound to the model, we alleviate a lot of the work in proving optimality. In 83% of instances, the computed lower-bound is a proven optimal for the relaxed problem.

6 Computational results on explanation-generation

In this section, we investigate how effective the explanation methods of Section 4 are for explaining and correcting infeasible schedules due to disruptions. More precisely, we first investigate the computation time and size of MUSes and MCSes and subsequently, we investigate how infeasible schedules can be repaired using feasibility restoration methods. We use the generator from Section 4.1 to generate 4663 instances, summarized in Section B. All experiments were ran on a 2.8GHz AMD CPU with 16 cores and 64GB of RAM.



■ **Figure 5** Computational results averaged over 5 seeds on the scheduling use-case.

6.1 Computation of MUS and MCS explanations

The results presented in this section were obtained by running default implementations of MUS and MCS-algorithms as implemented in CPMpy v0.9.23, on the unsatisfiable scenarios listed in Table 7. In particular, we use SHRINK [23] for computing *any* MUS, SMUS [19] for computing a smallest MUS, GROW [24, 25] for finding *any* MCS and finally we use a reified MAX-CSP formulation for finding an optimal MCS. In all MUS and MCS-experiments we use Exact [11] as solver, as it supports incremental and assumption-based solving [34].

Runtime of explanation-computation

In our tool, we want to be able to quickly compute explanations when disruptions occur, such that the planners can interact with these explanations. For computing MUSes, there is a clear runtime-difference between a smallest MUS, and computing a MUS using SHRINK as the runtime is doubled on average. Moreover, computing optimal MUSes times-out in 5.81% of the instances and yields significantly higher runtime for some disruption types such as the **two teams dropped** scenario, shown in Figure 3, left side. This is likely due to the inner workings of the SMUS algorithm which iteratively computes MCSes. In this scenario, many tasks can be re-allocated and hence, many MCSes may be enumerated.

Computing an MCS using GROW and optimal MCSes using MAX-CSP shows less difference in runtime, and both methods compute MCSes in under 1 second on average.

So, while computing MUSes and MCSes can be done quickly for this application, the potential time-out for SMUS does not outweigh the slight decrease in explanation size compared to computing MUSes using the simpler SHRINK algorithm.

Explanation size

From an application perspective, it is preferable to show small explanations are they are typically easier and quicker to understand by the planner. Figure 6b shows the size of MUSes and MCSes found by various methods for all disruption scenarios. Here, we clearly see a large

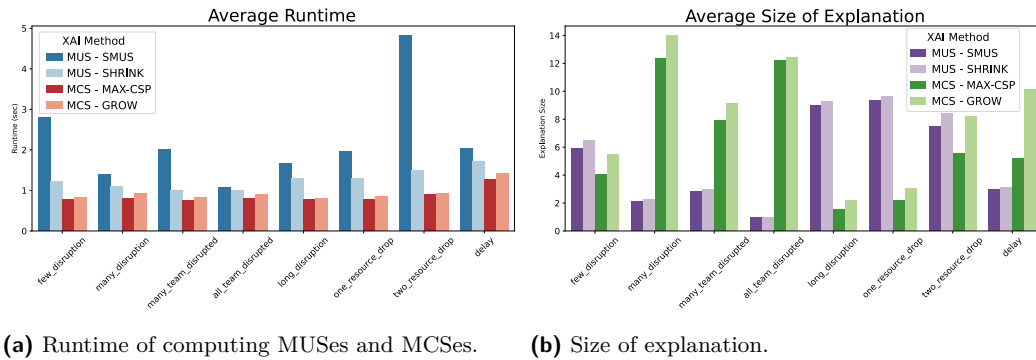


Figure 6 Comparison of runtime and explanation quality for different scenarios.

Table 3 Summary of averaged performance metrics for XAI methods on the allocation use-case. The number of constraints requiring a task to be allocated in explanations is shown in brackets.

Method	Time(s)	MUS size	%Timeout	Method	Time(s)	MCS size	%Timeout
SMUS	2.06	4.42 (3.72)	5.81	MAX-CSP	0.82	6.67 (6.29)	0.00
SHRINK	1.13	4.55 (3.99)	0.04	GROW	0.88	7.99 (7.31)	0.00

difference in the size of MUSes depending on the scenarios. For example, when all teams are disrupted, a MUS consists of just a single task that is allocated within that disruption period. Conversely, when one or two teams are dropped from the available set of teams, a MUS contains a set of tasks that fully saturates the original set of available teams, and one of these tasks was allocated to a now dropped team. Additionally, we can observe that in all scenarios, the size of an optimal MUS is close to the size of an arbitrary MUS computed by SHRINK. This suggests that all found MUSes have more or less the same size for our setting, given a disruption scenario.

Comparing the size of MCSes, there is overall a larger difference between the optimal MCS found by MAX-CSP and one found by GROW. For example, in the `delay` scenario, the Max-CSP approach on average finds a set of 6 constraints to drop, compared to 10 in the grow-based approach. Interestingly, the scenarios where one full team is dropped from the set of available teams do not necessarily correspond to larger MCSes, as only a few tasks need to be dropped from the allocation. This suggests that the full set of teams is only required at a few points in the working-shift, and most tasks can be allocated to different teams by (potentially many) re-allocations, instead of being dropped. Conversely, when many teams are affected by the same disruption, most tasks that overlap with this disruption have to be dropped from the planning horizon.

6.2 Multi-objective feasibility restoration

We now investigate the effectiveness of restoring feasibility by computing fine-grained alterations to the schedule as described in Section 4.3.2. We use OR-Tools CP-SAT with 10 search workers to solve the lexicographic optimization models as introduced in Section 4.3.1, using the sequential solving approach outlined in Section 2. For each of the 5 objectives, we set a time-limit of 5 seconds, such that the total time for feasibility restoration is limited to 25s. Table 4 summarizes the final objective value for each similarity objective for each of the optimization orders and across all scenario types. The number of not-done tasks (`#not-done`) is derived from the `#done` similarity objective.

■ **Table 4** Averaged objective value for feasibility restoration, grouped per lexicographic ordering.

Optimization order	<code>#not-done</code>	<code>#realloc</code>	<code>#shift</code>	<code>sum-shift</code>	<code>max-shift</code>
Reallocation first (A)	1.00	8.37	23.21	289.21	24.04
Shift first (B)	0.96	22.60	17.59	244.75	21.40
Sum Shift first (C)	0.98	22.48	16.66	198.93	16.15
Max Shift first (D)	0.98	22.95	17.54	199.71	14.91

Comparing the number of dropped tasks in the above table to the MCS-results in Table 3, we can clearly see that more-fine grained feasibility restoration leads to less tasks dropped. Indeed, repaired schedules using the optimal MCS drops on average 6.29 tasks, whereas allowing for re-scheduling of tasks only drops a single task on average.

As expected, there is a clear trade-off between shifting tasks and re-allocating tasks after a disruption. In “Reallocation first (A)”, which prioritizes minimization of re-allocated tasks, 8.37 tasks are re-allocated on average, while for the other orders between 22 and 23 tasks are re-allocated. Conversely, prioritizing minimization of shift-related metrics amounts to a lower number of shifted tasks (16-17) at the cost of more re-allocations.

Table 4 shows that by altering the order of the objectives, the resulting repaired schedule can be influenced significantly, and hence, we can use these orderings to propose several alternatives to the planners.

■ **Table 5** Fraction of instances solved to optimality for each sub-objective.

Optimization order	#not-done	#realloc	#shift	sum-shift	max-shift
Reallocation first (A)	0.97	0.96	0.84	0.90	0.89
Shift first (B)	0.98	0.63	0.60	0.68	0.66
Sum shift first (C)	0.97	0.64	0.43	0.80	0.67
Max shift first (D)	0.97	0.65	0.59	0.88	0.68

Interestingly, optimizing the total magnitude of shifted tasks, results in less tasks that are shifted on average, compared to minimizing **#shift** directly. Table 5 reveals that this is due to **#shift** being a more difficult objective to optimize compared to **sum-shift**, thereby often reaching the time-limit of 5s. Indeed, in only 60% of the instances, OR-Tools was able to provide an optimal solution when optimizing this continuity objective first. In “Sum Shift First (C)”, the **sum-shift** continuity objective was proven optimal in 80% of the instances and, by minimizing this simpler objective first, the resulting repaired schedule contains less shifted tasks overall.

7 Conclusions and future work

We have presented an industrial setting of a workforce-allocation and scheduling problem. We have showed that, using modeling techniques such as symmetry breaking, both PB and ILP-solvers can solve the allocation problem efficiently, and produce fair solutions regarding time-dispersion. For the scheduling variant, additional techniques such as adding the lower-bound of a relaxed problem are required in order to find optimal schedules. In future work, we plan to explore allowing for direct transfers from one location to another, which requires extending our formulation to a full allocation-and-routing problem [5].

Additionally, we have identified several disruption scenarios and investigated how to support a decision maker using explainable constraint solving techniques; we have experimentally shown that explanations such as MUSes and feasibility restoration can be computed efficiently. Hence, we deem it feasible to deploy such an interactive system as a decision support system for the planners. We are currently setting up a user-study to evaluate how effective the different generated explanations are in practice. Such user-study will enable to further tune the type of explanations generated by the system and going further, could allow us to learn preferences of the planners in repairing the schedule [16, 2, 31].

A future, alternative setting for dealing with possible disruptions in the schedule, is to ensure the original solution is *robust* [22]. A robust solution remains feasible, or even optimal when the parameters of the constraint model change. A robust schedule can for example introduce larger breaks in between tasks to account for possible delays. That robustness of a solution typically comes at the cost of higher computation times, and the need for specialized solving algorithms, which makes integration with production systems nontrivial.

Finally, we want to explore how we can assist planners by generating explanations for the full scheduling variant as well, similar to the explanations we developed for the allocation problem in this paper. This raises new challenges such as how to efficiently use global constraints in MUS and MCS-computation algorithms.

References

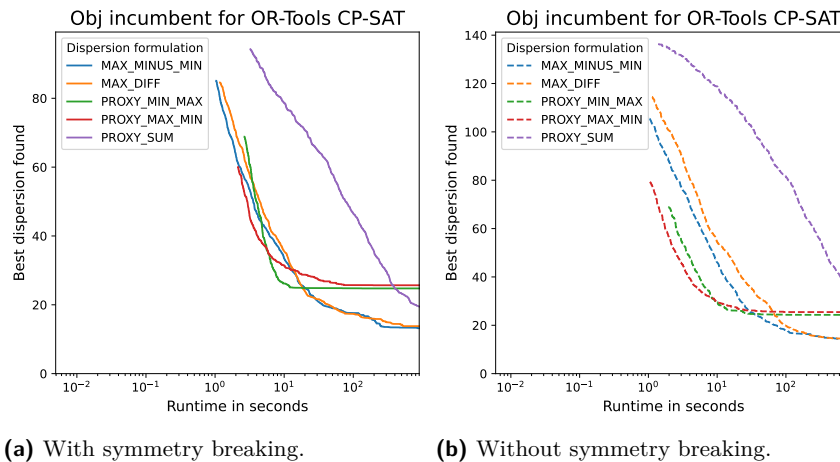
- 1 J Christopher Beck and Mark S Fox. Constraint-directed techniques for scheduling alternative activities. *Artificial Intelligence*, 121(1-2):211–250, 2000. doi:10.1016/S0004-3702(00)00035-7.
- 2 Nawal Benabbou and Thibaut Lust. An interactive polyhedral approach for multi-objective combinatorial optimization with incomplete preference information. In *International Conference on Scalable Uncertainty Management*, pages 221–235. Springer, 2019. doi:10.1007/978-3-030-35514-2_17.
- 3 Ignace Bleukx, Tias Guns, and Dimos Tsouros. Explainable Constraint Solving: A hands-on tutorial, February 2024. doi:10.5281/zenodo.10694140.
- 4 Nils Boysen, Philipp Schulze, and Armin Scholl. Assembly line balancing: What happened in the last fifteen years? *European Journal of Operational Research*, 301(3):797–814, 2022. doi:10.1016/J.EJOR.2021.11.043.
- 5 J Arturo Castillo-Salazar, Dario Landa-Silva, and Rong Qu. Workforce scheduling and routing problems: literature survey and computational study. *Annals of Operations Research*, 239:39–67, 2016. doi:10.1007/S10479-014-1687-2.
- 6 Violet Xinying Chen and JN Hooker. A guide to formulating equity and fairness in an optimization model. *Preprint*, pages 162–174, 2021.
- 7 João Cortes, Inês Lynce, and Vasco Manquinho. New core-guided and hitting set algorithms for multi-objective combinatorial optimization. In Sriram Sankaranarayanan and Natasha Sharygina, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings, Part II*, volume 13994 of *Lecture Notes in Computer Science*, pages 55–73. Springer, 2023. doi:10.1007/978-3-031-30820-8_7.
- 8 João Cortes, Inês Lynce, and Vasco Manquinho. Slide&drill, a new approach for multi-objective combinatorial optimization. In Paul Shaw, editor, *30th International Conference on Principles and Practice of Constraint Programming, CP 2024, September 2-6, 2024, Girona, Spain*, volume 307 of *LIPICs*, pages 8:1–8:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICs.CP.2024.8.
- 9 Giacomo Da Col and Erich C Teppan. Industrial-size job shop scheduling with constraint programming. *Operations Research Perspectives*, 9:100249, 2022.
- 10 Broes De Cat, Bart Bogaerts, Maurice Bruynooghe, Gerda Janssens, and Marc Denecker. Declarative logic programming. In Michael Kifer and Yanhong Annie Liu, editors, *Declarative Logic Programming: Theory, Systems, and Applications*, chapter Predicate Logic As a Modeling Language: The IDP System, pages 279–323. Association for Computing Machinery and Morgan & Claypool, New York, NY, USA, 2018. doi:10.1145/3191315.3191321.
- 11 Jo Devriendt. Exact solver, 2023. URL: <https://gitlab.com/JoD/exact>.
- 12 Zehranaz Dönmez, Merve Ayyıldız, Benay Uslu, Ozlem Karsu, and Bahar Y Kara. Fairness in humanitarian logistics: State of the art and future directions, 2022.
- 13 Jan Elffers and Jakob Nordström. Divide and conquer: Towards faster pseudo-Boolean solving. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 1291–1299. ijcai.org, 2018. doi:10.24963/ijcai.2018/180.
- 14 Maria Fox, Alfonso Gerevini, Derek Long, and Ivan Serina. Plan stability: Replanning versus plan repair. In Derek Long, Stephen F. Smith, Daniel Borrajo, and Lee McCluskey, editors, *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling, ICAPS 2006, Cumbria, UK, June 6-10, 2006*, pages 212–221. AAAI, 2006. URL: <http://www.aaai.org/Library/ICAPS/2006/icaps06-022.php>.
- 15 Emilio Gamba, Bart Bogaerts, and Tias Guns. Efficiently explaining cps with unsatisfiable subset optimization. *J. Artif. Intell. Res.*, 78:709–746, 2023. doi:10.1613/JAIR.1.14260.

- 16 Andreia P. Guerreiro, João Cortes, Daniel Vanderpooten, Cristina Bazgan, Inês Lynce, Vasco Manquinho, and José Rui Figueira. Exact and approximate determination of the pareto front using minimal correction subsets. *Comput. Oper. Res.*, 153:106153, 2023. doi:10.1016/J.COR.2023.106153.
- 17 Tias Guns. Increasing modeling language convenience with a universal n-dimensional array, cppy as python-embedded example. In *Proceedings of the 18th workshop on Constraint Modelling and Reformulation at CP (Modref 2019)*, volume 19, 2019.
- 18 Sharmi Dev Gupta, Begum Genc, and Barry O’Sullivan. Explanation in constraint satisfaction: A survey. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 4400–4407. ijcai.org, 2021. doi:10.24963/ijcai.2021/601.
- 19 Alexey Ignatiev, Alessandro Previti, Mark H. Liffiton, and João Marques-Silva. Smallest MUS extraction with minimal hitting set dualization. In Gilles Pesant, editor, *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings*, volume 9255 of *Lecture Notes in Computer Science*, pages 173–182. Springer, 2015. doi:10.1007/978-3-319-23219-5_13.
- 20 Ulrich Junker. Quickxplain: Conflict detection for arbitrary constraint propagation algorithms. In *IJCAI’01 Workshop on Modelling and Solving problems with constraints*, volume 4. Citeseer, 2001.
- 21 Philippe Laborie, Jerome Rogerie, Paul Shaw, and Petr Vilím. Reasoning with conditional time-intervals. part II: an algebraical model for resources. In H. Chad Lane and Hans W. Guesgen, editors, *Proceedings of the Twenty-Second International Florida Artificial Intelligence Research Society Conference, May 19-21, 2009, Sanibel Island, Florida, USA*. AAAI Press, 2009. URL: <http://aaai.org/ocs/index.php/FLAIRS/2009/paper/view/60>.
- 22 Jheisson López, Alejandro Arbelaez, and Laura Climent. Large neighborhood search for robust solutions for constraint satisfaction problems with ordered domains. In Christine Solnon, editor, *28th International Conference on Principles and Practice of Constraint Programming, CP 2022, July 31 to August 8, 2022, Haifa, Israel*, volume 235 of *LIPICs*, pages 33:1–33:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.CP.2022.33.
- 23 João Marques-Silva. Minimal unsatisfiability: Models, algorithms and applications (invited paper). In *40th IEEE International Symposium on Multiple-Valued Logic, ISMVL 2010, Barcelona, Spain, 26-28 May 2010*, pages 9–14. IEEE Computer Society, 2010. doi:10.1109/ISMVL.2010.11.
- 24 João Marques-Silva, Federico Heras, Mikolás Janota, Alessandro Previti, and Anton Belov. On computing minimal correction subsets. In Francesca Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 615–622. IJCAI/AAAI, 2013. URL: <http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6922>.
- 25 Carlos Mencía and Joao Marques-Silva. Efficient relaxations of over-constrained cps. In *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*, pages 725–732. IEEE, 2014.
- 26 Raymond Reiter. A theory of diagnosis from first principles. *Artificial intelligence*, 32(1):57–95, 1987. doi:10.1016/0004-3702(87)90062-2.
- 27 Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006. URL: <https://www.sciencedirect.com/science/bookseries/15746526/2>.
- 28 Pierre Schaus and Renaud Hartert. Multi-objective large neighborhood search. In Christian Schulte, editor, *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, volume 8124 of *Lecture Notes in Computer Science*, pages 611–627. Springer, 2013. doi:10.1007/978-3-642-40627-0_46.

- 29 Andreas Schutt, Thibaut Feydy, and Peter J Stuckey. Explaining time-table-edge-finding propagation for the cumulative resource constraint. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 10th International Conference, CPAIOR 2013, Yorktown Heights, NY, USA, May 18-22, 2013. Proceedings 10*, pages 234–250. Springer, 2013. doi:10.1007/978-3-642-38171-3_16.
- 30 Ilankaikone Senthoran, Matthias Klapperstück, Gleb Belov, Tobias Czauderna, Kevin Leo, Mark Wallace, Michael Wybrow, and Maria Garcia de la Banda. Human-centred feasibility restoration in practice. *Constraints An Int. J.*, 28(2):203–243, 2023. doi:10.1007/S10601-023-09344-5.
- 31 Federico Toffano, Michele Garraffa, Yiqing Lin, Steven Prestwich, Helmut Simonis, and Nic Wilson. A multi-objective supplier selection framework based on user-preferences. *Annals of Operations Research*, pages 1–32, 2022.
- 32 Petr Vilím. Edge finding filtering algorithm for discrete cumulative resources in $O(kn \log n)$. In *CP*, volume 5732 of *Lecture Notes in Computer Science*, pages 802–816. Springer, 2009. doi:10.1007/978-3-642-04244-7_62.
- 33 Petr Vilím. Max energy filtering algorithm for discrete cumulative resources. In *CPAIOR*, volume 5547 of *Lecture Notes in Computer Science*, pages 294–308. Springer, 2009. doi:10.1007/978-3-642-01929-6_22.
- 34 Siert Wieringa. *Incremental satisfiability solving and its applications*. PhD thesis, Aalto University School of Science, Department of Computer Science and Engineering, 2014.

A Detailed results on optimizing dispersion

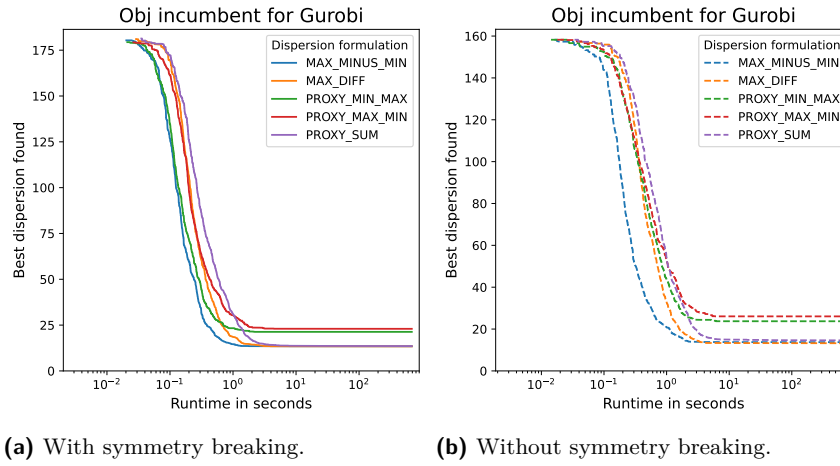
Below, we show the detailed objective incumbent for each of the solver/model configurations. Note that we show the *true* dispersion value according to the exact formulation.



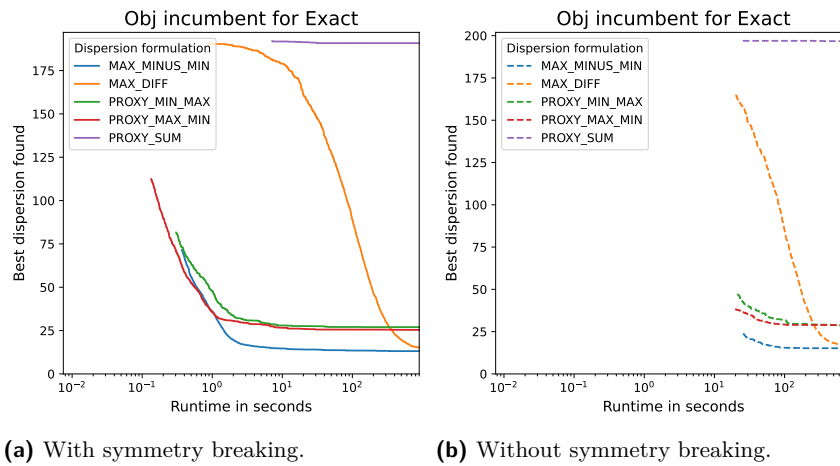
■ **Figure 7** Dispersion objective incumbent for OR-Tools CP-SAT.

■ **Table 6** Percentage of instances solved to optimality.

Solver	Break symmetries?	Dispersion variant	%solved to optimality
OR-Tools CP-SAT	No	MAX_DIFF	7.41
		MAX_MINUS_MIN	8.52
		PROXY_MAX_MIN	5.93
		PROXY_MIN_MAX	8.52
		PROXY_SUM	7.04
	Yes	MAX_DIFF	9.63
		MAX_MINUS_MIN	9.63
		PROXY_MAX_MIN	5.93
		PROXY_MIN_MAX	8.89
		PROXY_SUM	7.41
Gurobi	No	MAX_DIFF	17.41
		MAX_MINUS_MIN	18.89
		PROXY_MAX_MIN	78.15
		PROXY_MIN_MAX	78.15
		PROXY_SUM	13.33
	Yes	MAX_DIFF	30.37
		MAX_MINUS_MIN	33.33
		PROXY_MAX_MIN	78.52
		PROXY_MIN_MAX	78.52
		PROXY_SUM	15.56
Exact	No	MAX_DIFF	7.41
		MAX_MINUS_MIN	8.89
		PROXY_MAX_MIN	5.19
		PROXY_MIN_MAX	5.56
		PROXY_SUM	4.81
	Yes	MAX_DIFF	10.00
		MAX_MINUS_MIN	14.44
		PROXY_MAX_MIN	5.93
		PROXY_MIN_MAX	11.85
		PROXY_SUM	22.22

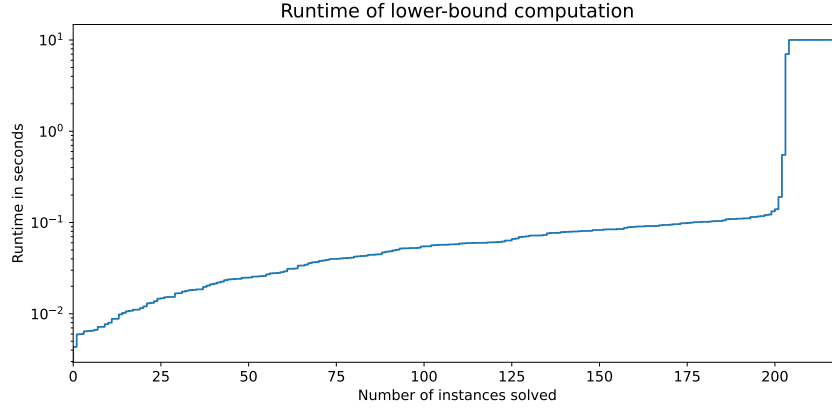


■ **Figure 8** Dispersion objective incumbent for Gurobi.



■ **Figure 9** Dispersion objective incumbent for Exact.

Figure 10 shows the number of instances and solve-time for finding a lower-bound of the scheduling problem by solving the relaxed problem from Section 3.2.



■ **Figure 10** Runtime for solving the relaxed problem to optimality.

B Detailed results on explanation-generation

Table 7 summarizes the instances used for all explanation-generation methods for each scenario.

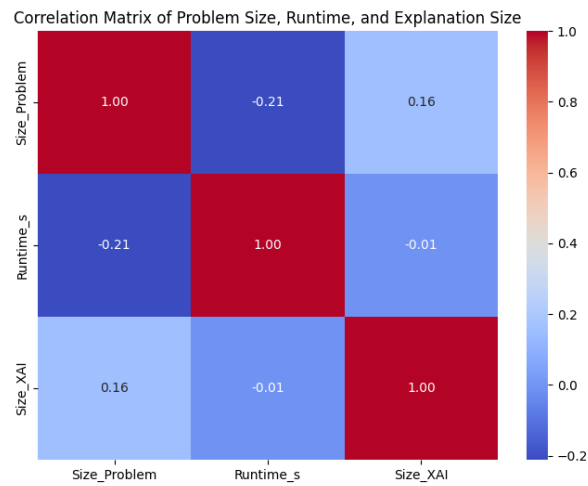
■ **Table 7** Overview of generated scenarios, relative value to total number of tasks in parenthesis.

Scenario description	#Instances count	#Disr. mean	#Teams(%) mean	\sum duration mean	#Tasks Disr(%) mean
few disruptions	1167	2.71	5.66 (0.6)	411.56	11.23 (0.11)
long disruptions	411	2.45	2.27 (0.26)	582.19	6.48 (0.06)
many disruptions	128	6.16	8.72 (1)	1355.74	17.88 (0.13)
many teams affected	1049	1	7.83 (0.83)	361.74	9.11 (0.08)
all teams affected	974	1	9.46 (1)	383.59	10.58 (0.09)
one team dropped	334	1	1 (0.11)	744.43	13.27 (0.11)
two teams dropped	306	1	2 (0.21)	1480.78	26.48 (0.21)
delay	294	—	—	—	—

Table 8 provides a summary of the number of instances, average runtime, and average explanation size, categorized by scenario type and XAI method. The explanation size corresponds to the number of constraints involved in the generated explanations.

■ **Table 8** Total number of instances, average runtime, and average explanation size, number of timeout, broken down by scenario type and XAI method (MCS vs. MUS).

Scenario description	Explanation type	No. Of instances	Runtime (s) ↓	Size XAI ↓	Timeout
few disruptions	MCS	2334	0.80	4.75	0
	MUS	2186	1.95	6.23	148
many disruptions	MCS	256	0.86	13.18	0
	MUS	254	1.24	2.16	2
many teams affected	MCS	2098	0.80	8.55	0
	MUS	2072	1.50	2.89	26
all teams affected	MCS	1948	0.84	12.31	0
	MUS	1948	1.03	1.00	0
long disruptions	MCS	822	0.79	1.87	0
	MUS	821	1.48	9.12	1
one team dropped	MCS	668	0.82	2.63	0
	MUS	666	1.63	9.49	2
two teams dropped	MCS	612	0.92	6.90	0
	MUS	518	2.86	8.05	94
delay	MCS	588	1.35	7.67	0
	MUS	588	1.87	3.05	0



■ **Figure 11** Correlation between problem size, runtime, and explanation size.

C Detailed results on rescheduling per scenario type

Table 9 display the detailed results for each scenario type, using the objective order A. We see that the scenario type with the most task that are dropped is the **all teams affected**. On other continuity objectives, the **two teams dropped** implies more shifts.

■ **Table 9** Rescheduling results per scenario type (only with optimisation order **A**).

Scenario description	#not-done	#realloc	#shift	sum-shift	max-shift
few disruptions	0.21	8.19	26.36	347.48	26.43
many disruptions	1.80	8.81	29.08	392.32	25.85
many teams affected	0.94	5.55	22.83	302.94	25.85
all teams affected	2.87	5.15	24.21	305.54	24.85
long disruptions	0.19	8.23	11.62	140.29	21.94
one team dropped	0.42	14.47	13.47	156.26	22.69
two teams dropped	0.67	28.07	36.95	507.95	26.09
delay	0.01	2.00	20.51	57.04	6.97

Table 10 details the optimality proof per scenario type, highlighting notably that the scenario **two teams dropped** is the most difficult.

■ **Table 10** Fraction of optimality proof for each lexico-phase, for each scenario type.

Scenario description	#not-done mean	#realloc mean	#shift mean	sum-shift mean	max-shift mean
few disruptions	0.98	0.54	0.41	0.69	0.53
many disruptions	0.92	0.66	0.58	0.87	0.72
many teams affected	0.95	0.83	0.73	0.93	0.87
all teams affected	0.97	0.86	0.79	0.98	0.89
long disruptions	0.98	0.75	0.58	0.73	0.71
one team dropped	0.99	0.71	0.58	0.77	0.73
two teams dropped	0.96	0.34	0.12	0.44	0.24
delay	0.99	0.99	0.99	0.96	0.99