

Trustworthy and Explainable Decision-Making for Workforce allocation

Guillaume Povéda ✉ 

Airbus SAS, FR

Andreas Strahl ✉

Airbus Aerostructures, GmbH

Mark Hall ✉

Airbus Operations Ltd, UK

Ryma Boumazouza ✉ 

Airbus SAS, FR

Santiago Quintana-Amate ✉ 

Airbus Operations Ltd, UK

Nahum Alvarez ✉ 

Airbus SAS, FR

Ignace Bleukx ✉ 

DTAI, KU Leuven, Leuven, Belgium

Dimos Tsouros ✉ 

DTAI, KU Leuven, Leuven, Belgium

Hélène Verhaeghe ✉ 

DTAI, KU Leuven, Leuven, Belgium

Tias Guns ✉ 

DTAI, KU Leuven, Leuven, Belgium

Abstract

In industrial contexts, effective workforce allocation is crucial for operational efficiency. This paper presents an ongoing project focused on developing a decision-making tool designed for workforce allocation, emphasizing the explainability to enhance its trustworthiness. Our objective is to create a system that not only optimises the allocation of teams to scheduled tasks but also provides clear, understandable explanations for its decisions, particularly in cases where the problem is infeasible. By incorporating human-in-the-loop mechanisms, the tool aims to enhance user trust and facilitate interactive conflict resolution. We implemented our approach on a prototype tool/digital demonstrator intended to be evaluated on a real industrial scenario both in terms of performance and user acceptability.

2012 ACM Subject Classification Human-centered computing → User centered design

Keywords and phrases CP, Explainable CP, Trustworthy AI

Acknowledgements This work has been partially funded by the European Union's Horizon Europe Research and Innovation program under the grant agreement TUPLES No 101070149

1 Introduction

In industrial contexts, effective workforce allocation is a cornerstone of operational efficiency, directly impacting productivity, cost management, and overall organizational performance. The complex nature of workforce allocation involves balancing numerous constraints, such as employee availability, skill levels, regulatory requirements, and task priorities. As industries

increasingly rely on automated decision-making tools to manage these complexities, the need for trustworthiness and explainability in these systems becomes paramount.

This paper introduces an ongoing project dedicated to the development of a decision-making tool tailored to workforce allocation. The core objective of this tool is to not only optimise the allocation of teams to scheduled tasks but also to ensure that the decision-making process is transparent and understandable to users. Current industrial workforce allocation often functions as a black box, primarily due to the complexity and opacity of the underlying processes. This lack of transparency hinders the general understandability of the solution and is detrimental to the development and deployment of automatic solutions using AI tools. Our work aims to address this problem by improving transparency and explainability of workforce allocation systems. Complicating workforce allocation processes, are the need for real-time adaptation of the workforce under disruptions. The necessary knowledge to manage these disruptions is often implicit, ‘hidden’ in the planners’ heads, making it difficult for AI-generated solutions to gain acceptance unless they can clearly explain their rationale. Our approach not only seeks to enhance the transparency of workforce allocation but also aims to ensure that AI solutions can effectively communicate their decision-making processes, thereby increasing trust and acceptance among human planners.

Another significant challenge in workforce allocation is the occurrence of infeasible situations, where the constraints cannot be satisfied simultaneously. Traditional systems may simply fail or produce sub-optimal solutions without providing clear explanations, leading to user frustration and mistrust. To overcome this, our tool incorporates human-in-the-loop mechanisms, enabling users to interact with the system to understand and resolve infeasibilities. These explainability features are designed to enhance user trust and facilitate effective conflict resolution, making the decision-making process more collaborative and reliable.

In summary, this paper presents an integrated approach to workforce allocation, emphasizing the importance of trustworthiness and explainability. By integrating interactive features and human-in-the-loop mechanisms, we aim to create a decision-making tool that is not only effective but also transparent and user-friendly, paving the way for more reliable and collaborative industrial operations.

Looking ahead, future plans include evaluating the tool’s effectiveness. This evaluation will focus on assessing the tool’s impact on operational efficiency, user understandability and acceptance, and its ability to handle real-world workforce allocation scenarios.

1.1 Overview of workforce allocation challenges

In the industrial landscape, efficient workforce allocation or task scheduling is a critical component of operational success. We consider here the operational problem of assigning teams of workers, to a set of *already scheduled* tasks, in a manner that optimises workers utilization and meets various operational constraints. Furthermore, workers have different availability slots; in real scenarios, uncertainty (represented by accidents, illnesses or simply time delays in other tasks) may further modify this pre-established availability.

To address this, we have already developed a decision-making tool relying on constraint programming (CP) [34], a powerful paradigm well-suited for solving complex allocation problems. While we will describe this tool in detail in the next section, it is important to note that even with a highly performant solver, eXplainable AI (XAI) is essential to ensure the trustworthiness and acceptance of AI solutions in workforce allocation.

Despite the technical robustness of CP solvers, their adoption in industrial settings is often hindered by a perceived lack of transparency and lack of user interaction capabilities.

Decision-makers and end-users frequently struggle to understand the rationale behind the solver's outputs, particularly when the problem is infeasible. This can lead to mistrust and underusage of the technology, ultimately diminishing its potential benefits. Also, the actual modelling of the problem may be challenging as the modelling experts are often not the final users of the decision-making tool.

To overcome these challenges, our project focuses on integrating explainability and trustworthiness into the CP-based decision-making tool. By providing clear, comprehensible explanations for the solver's decisions and highlighting reasons for infeasibilities, we aim to build greater user trust and facilitate more effective human-computer collaboration. Interactive features are also being developed to allow users to engage with the tool, explore alternative solutions, and iteratively restore feasibility when conflicts arise.

This paper outlines our ongoing efforts to create an explainable and trustworthy workforce allocation tool. We demonstrate the implementation of interactive conflict resolution mechanisms and discuss our plans for evaluating these features.

1.2 Explainability in Constraint Programming

Explainability in AI has evolved significantly over time, driven by the need to make AI systems more transparent, trustworthy, and user-friendly. The authors in [6] broadly categorized the questions that explanations in AI aim to answer into three classes: *What and Why* (What made/Why did the system reach this outcome?), *Why not and What if* (Why did the system not reach a different outcome? What if different information were used?), and *How* (How can I modify the system to obtain a more desirable outcome with the existing information?). This categorization helps understand the progression and focus of explainability efforts in various AI methodologies, including machine reasoning (MR) and machine learning (ML). Different methodologies have addressed these explainability questions (see [10, 16, 18, 19, 24, 26, 27, 28, 29, 33, 35, 37]).

The remainder of this section focuses on the specific application of XAI techniques within the domain of constraint programming, especially in workforce allocation and scheduling problems.

Constraint Programming is a powerful method at the intersection of AI and OR, for solving combinatorial problems. CP involves specifying constraints that need to be satisfied and finding solutions that meet these constraints. Explainability is crucial in CP, particularly for workforce allocation and scheduling, where decision-makers need to understand the rationale behind the allocation decisions. Different existing methods are used to enhance explainability in CP and can be categorized as:

- Explanation of Constraints: Making the constraints and their roles in the decision-making process clear to users.
- Solution Traceability: Allowing users to trace back the steps and decisions made by the CP solver to understand how a particular solution was reached (e.g., [1]).
- Conflict Explanation: Identifying and explaining conflicts or infeasibilities when no solution can be found, which is particularly important for iterative problem-solving and debugging (e.g., [22, 25]).

A significant focus within explainable constraint solving is on the latter and is about explaining why a set of constraints is unsatisfiable. Many of these methods [4, 15, 17, 20, 21, 22, 23, 25] aim to identify a minimal unsatisfiable subset (MUS) - an irreducible subset of constraints which causes the model to be unsatisfiable.

► **Definition 1** (Minimal Unsatisfiable Subset [23]). *Given an unsatisfiable set of constraints C , a subset $U \subseteq C$ is a Minimal Unsatisfiable Subset if and only if U is unsatisfiable and every strict subset $U' \subsetneq U$ is satisfiable*

Such explanations are interesting because they pinpoint the exact constraints responsible for the inconsistency, allowing users to focus their efforts on resolving specific issues. Recently, research has also been directed towards advising users on how to restore feasibility [12, 36], notably by identifying the minimal correction subset (MCS) [3].

► **Definition 2** (Minimal Correction Subset [23]). *Given an unsatisfiable set of constraints C , a subset $M \subseteq C$ is a Minimal Correction Subset if and only if $C \setminus M$ is satisfiable, and for every strict subset $M' \subsetneq M$, $C \setminus M'$ is unsatisfiable.*

An MCS is particularly useful because it identifies an irreducible set of constraints that, when modified or relaxed, can restore the feasibility of the entire system. By focusing on such a minimal set, users can implement the least disruptive changes necessary to resolve conflicts, which helps maintain the integrity of the original constraint problem as much as possible. However, there remains a shortage of tools that effectively explain why a problem is inconsistent.

2 Problem definition

The problem consists of assigning teams of workers to tasks in a large-scale industrial setting, involving several hundreds of daily activities. We will consider the set of tasks to accomplish as already scheduled in time, each of them needs to be allocated to a team of workers. Any given team of workers can't be allocated to two activities at the same time neither do 2 tasks in a row when there is some geographical constraint such transportation time that makes it impossible. Each team has its own calendar of availability or set of skills that can restrict the set of activities it can be allocated to. In this section, we will introduce the needed notations and formulate the base constraint model implemented to solve it:

2.1 Notations

1. $\mathcal{A}$ the set of activities to accomplish
2. $\mathcal{W}$ the set of worker teams available
3. $\forall a \in \mathcal{A}$, $start_a \in \mathbb{N}$, $end_a \in \mathbb{N}$, the start and end time of the activity a
4. $\forall a \in \mathcal{A}$, $comp_a \in 2^{\mathcal{W}}$ stores the subset of worker teams compatible with the activity a . Similarly we can define binary indicator $comp_binary_{a,w} \in \{0, 1\}$, $\forall a \in \mathcal{A}$, $\forall w \in \mathcal{W}$ storing the same information.
5. $\mathcal{S}$ is a list of activity pair (a_i, a_j) that should be allocated to the same team.

2.2 Constraint model

In this section, we detail the CP formulation implemented for the problem. A Boolean formulation showed the best performance using the different solvers we tested in our backend application (like Ortools CP-SAT [32], Exact [7], and Gurobi [13]).

Variables

1. Let $\forall a \in \mathcal{A}$, $w \in \mathcal{W}$, $alloc_{a,w} \in \{0, 1\}$ be the allocation variable. A value of 1 will correspond to given worker team w being allocated to the activity a .

- 174 2. Let $\forall w \in \mathcal{W}, used_w \in \{0, 1\}$, be the Boolean variable indicating if a given team w is
 175 allocated to any of activities $a \in \mathcal{A}$

176 Constraints

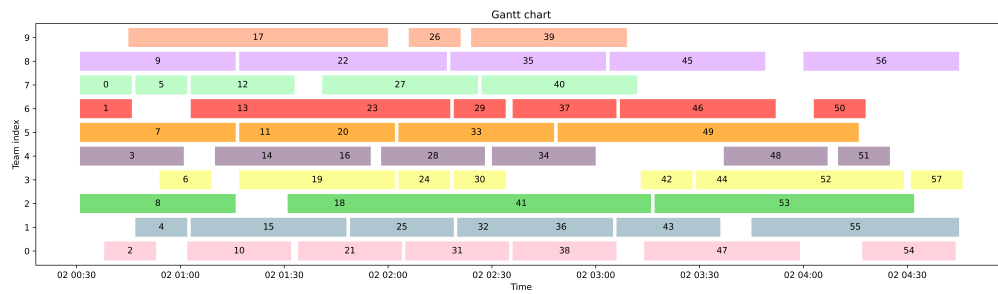
- 177 1. Each task is allocated : $\forall a \in \mathcal{A}, \sum_{w \in \mathcal{W}} alloc_{a,w} = 1$
 178 2. Non-Overlapping constraint :
 179 $\forall a \in \mathcal{A}$ we denote $neigh(a) = \{a' \in \mathcal{A} \text{ s.t. } (end_{a'} > start_a) \wedge end_a \geq start_{a'}\}$ the set of
 180 overlapping activities of activity a , then $\forall w \in \mathcal{W}, a' \in neigh(a), alloc_{a,w} + alloc_{a',w} \leq 1$
 181 3. Compatibility constraint : $\forall a \in \mathcal{A}, w \in \mathcal{W}, \neg comp_binary_{a,w} \rightarrow \neg alloc_{a,w},$
 182 4. Same allocation constraint :
 183 $\forall (a_1, a_2) \in \mathcal{S}, \forall w \in \mathcal{W}, alloc_{a_1,w} = alloc_{a_2,w}$
 184 5. Used team constraint : $\forall a \in \mathcal{A}, w \in \mathcal{W}, alloc_{a,w} \rightarrow used_w$
 185 6. Aiming at speeding up solver we introduce two main additional kinds of constraint, one
 186 redundant for the overlapping constraint, and one adding symmetry breaking :
 187 a. Clique constraints :
 188 $\forall a \in \mathcal{A}$, let $overlapstart(a) = \{a' \in \mathcal{A}, start_{a'} \leq start_a < end_{a'}\}$ the set of task also
 189 executed at time $start_a$ (including a), then this set constitutes a clique of overlapping
 190 tasks. We add the following constraint :
 191 $\forall a \in \mathcal{A}, \forall w \in \mathcal{W}, \sum_{a' \in overlapstart(a)} alloc_{a',w} \leq 1$
 192 b. Symmetry breaking: Some teams $\in \mathcal{W}$ can execute the same set of tasks for the given
 193 time horizon. Hence, they are equivalent and tasks can be assigned to any of those
 194 teams without changing the validity of the allocation. Clearly, this means equivalent
 195 teams are *symmetric* and we add lexleader symmetry breaking constraints imposing
 196 an ordering of the teams [8, 39]. Several formulations are possible, but from limited
 197 testing, we found adding the ordering on the *used* variables seemed most promising.
 198 It's worth noticing that this constraint will not impact solution quality, only when the
 199 objective itself treats the teams as equivalent.

200 Objective functions

201 The main objective of interest here will be the number of different teams used, therefore we
 202 aim at minimizing $\sum_{w \in \mathcal{W}} used_w$. Several other objectives are under study, notably adding
 203 fairness objectives, and ensuring a balanced workload among the used teams. The inclusion
 204 of those objective functions has currently only been studied in the pure optimisation and
 205 performance side and not on the explainable, therefore they will not be considered in the
 206 remaining of the paper.

207 Example of solution

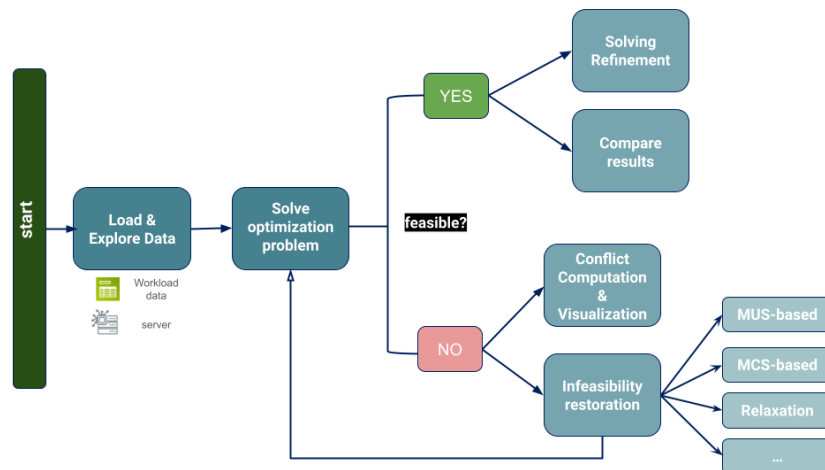
208 We can plot a Gantt chart to visualise the solution, as shown in Figure 1. Each row of
 209 the chart represents the schedule for a specific team of workers $w \in \mathcal{W}$. Due to the non-
 210 overlapping constraint (defined in constraint nb. 2), a feasible solution ensures that there
 211 are no overlapping activities within each row of the Gantt chart.



■ **Figure 1** Example of Gantt chart built to visualise a solution to the workforce allocation problem

3 Explainable Decision-making tool for workforce allocation

The development of a decision-making tool for workforce allocation is driven by the need to enhance operational efficiency, but such a tool introduces new trustworthiness requirements in order to get user acceptance. The following figure 2 outlines the primary workflow of the tool.



■ **Figure 2** Workflow of the Decision-Making Tool

Our tool integrates explainability components addressing two major needs: conflict computation and visualisation, and interactive infeasibility restoration. The explainability features of our tool are tailored to scenarios where the workforce allocation problem is infeasible, where the constraints cannot be satisfied simultaneously (e.g., when there are insufficient resources available to allocate all tasks). By addressing these infeasibility cases, the tool aims to provide insights into its decision-making process.

3.1 Conflicts Computation and Visualisation

In complex allocation or scheduling scenarios, conflicts are often inevitable due to various reasons: overlapping tasks, resource constraints, and varying team availabilities. Our tool computes and visualises these conflicts, allowing users to see where and why the allocation process encounters issues. Visual representations of conflicts enable users to quickly grasp problematic areas and understand the constraints causing these issues. This transparency

229 builds trust in the system, as users can see the logical reasoning behind the solver’s decisions.
 230 Finding the best way to visualise the conflicts depends on user preferences, and this is the
 231 subject of ongoing work.

232 3.2 Interactive Infeasibility Restoration

233 Confronting infeasible problems is a common challenge in real-world applications [5]. Tradi-
 234 tional CP solvers may report infeasibility without providing guidance on resolution. However,
 235 our tool offers the users an interactive method to solve conflicts in the problem; upon detecting
 236 an infeasible problem, users are presented with several methods to restore feasibility:

- 237 ■ **Resolving MUS conflicts interactively (local conflict resolution):** This method
 238 involves resolving each MUS conflict one by one in an interactive manner (by selecting
 239 a constraint in the MUS to relax). Local conflict restoration refers to the process of
 240 addressing each conflict individually within its localized context, rather than attempting
 241 to solve all conflicts simultaneously. Users are guided through the process of addressing
 242 each local conflict sequentially, enabling a step-by-step restoration of feasibility.
- 243 ■ **Using MCS interactively (global conflict resolution):** Instead of addressing conflicts
 244 individually, this approach computes one of the minimal correction subsets (MCS) to
 245 resolve all conflicts simultaneously on a problem-wide scale. Global conflict restoration
 246 refers to the process of identifying and correcting a minimal set of constraints that, when
 247 adjusted, will restore feasibility to the entire system. In our tool, we consider the scenario
 248 where the user can choose only a subset of the relaxations provided by a single MCS, and
 249 users may want to mix-and-match constraints relaxations from different MCSes. Hence,
 250 our tool re-computes a new MCS after a user has relaxed some constraints, making the
 251 process iterative and interactive.
- 252 ■ **Fine-tuning task priorities (prioritized conflict resolution):** This method involves
 253 solving and optimising a relaxed version of the problem where task allocations become
 254 optional. Each task is given a priority/weight value which is taken into account in the
 255 optimisation criteria. Users can interactively change the priority level of tasks, allowing a
 256 lot of flexibility in the way the problem feasibility is restored, e.g. which tasks are more
 257 likely to remain or be removed.

258 By involving users in the resolution process, our tool ensures a more transparent, inter-
 259 active, and trustworthy decision-making experience.

260 3.3 Implementation

261 The workforce allocation model was implemented using the `CPMpy` library [11], a flexible
 262 and user-friendly tool for modelling constraint programming (CP) problems. `CPMpy` offers
 263 an intuitive API that closely mirrors the functionality of `numpy`, making it accessible and
 264 easy to use for those familiar with numerical computing in Python. Using this modelling
 265 library allows us to test different solver backends, including `ortools-cpsat` [32], `gurobi` [13],
 266 `pysat` [14], or `exact` [7, 9]. It also includes some native utilities to compute MUSes or MCSes,
 267 which we use extensively in this research for conflict analysis and feasibility restoration.

268 In practice, several customization options regarding optimisation and explainability
 269 aspects are available through our configuration parameters tab within the tool, as illustrated
 270 in Figure 3.

The screenshot shows the Tuplex web interface. On the left is a sidebar with a 'Mode' toggle, the Tuplex logo, and two file upload sections: 'Uploading Files' (for resource description files) and 'Path to activity description file' (for task/activity files). The main area is titled 'ALLOCATION = Team allocation problem including calendar availability and with objective the # of resource used'. It contains two tabs: 'Params constraints' and 'Params model encoding'. The 'Params constraints' tab is active, showing a grid of dropdown menus for various constraint settings. The 'Params model encoding' tab is also visible, showing settings for model encoding. Below these is a 'MUS settings' section with more dropdowns.

Params constraints			
Constraint settings task_allocated: soft	Constraint settings pair_overlap: hard	Constraint settings allowed_team: soft	Constraint settings clique_overlap: soft
Constraint settings team_is_used: hard	Constraint settings symmetries: hard	Constraint settings same_allocation: soft	Constraint settings allowed_alloc: soft
Constraint settings nb_max_teams: soft			

Params model encoding			
Param include_all_binary_vars: True	Param include_pair_overlap: False	Param overlapping_advanced: True	Param symmbreak_on_used: True

MUS settings			
Infeasibility Method: ADD_CONSTRAINT	Solver inside mus: exact	Method Mus: mus	Visual Conflict: MAX_SAT

■ Figure 3 Configure the methods parameters tab

271 4 Case study/Application example

272 The problems to be solved in the industrial use case range from scheduling tasks over a
 273 six-hour period to creating a full day (24-hour) schedule, involving the allocation of a few
 274 dozen activities to possibly up to one thousand. The number of available resources (i.e. our
 275 $\mathcal{W}$ teams) varies over time, but typically there are about 20 ($|\mathcal{W}| \approx 20$).

276 4.1 Preliminary results

277 In this section, we present our initial findings on the computational performance of the
 278 optimisation method and on the explainability components across various scenarios. For both
 279 the optimisation and explainability experiments, we generated 20 instances of the allocation
 280 problem with different lengths: 6, 8 and 24 hours. These instances were generated to reflect
 281 a real-world scenario with specific constraints and conditions derived from historical data.
 282 This analysis serves as a foundation for further refinement and optimisation of our approach.

283 4.1.1 Optimisation results

284 Despite the workforce allocation problem being NP-Hard (akin to a list colouring graph
 285 problem), preliminary empirical runs and benchmarks on historical data have demonstrated
 286 good performance. Our preliminary results (Table 1) consider the mean computation time to
 287 optimality (or cut to timeout) for different lengths of the instances, different CP formulations
 288 of the CP model, and different solver settings. The column **clique** refers to the redundant
 289 clique constraint 6a and **symmetry** to the symmetry breaking constraint on used team 6b.
 290 The solver backend used is Ortools' CP-Sat solver, a state-of-the-art solver for CP problems
 291 [30, 31]. CP-Sat heavily relies on a portfolio approach to accomplish its search and using this
 292 feature usually will improve a lot the solving performance. To check this on our use case, we
 293 tested 2 different settings: using 1 or 6 search worker (column **#w**). As we expected, CP-Sat
 294 is more efficient in its multi-worker settings and found optimal solution on all instances in
 295 less than 1 second in average. From the multi-worker settings instances, we also observed
 296 that symmetry and redundant constraints have a clear negative effect on both initialisation
 297 time of the model and on solving time. On the contrary, in the mono-worker mode, the use of
 298 the symmetry constraints helps prove optimality for more instances and offers a computation
 299 time advantage (but still those have worse performance than their multi-worker equivalents).

300 These results show the efficiency of using CP-Sat with its full features activated, but it is

#w	len	clique	symmetry	t_init(s)	t_solve(s)	t_total(s)	optimal
1	6	False	False	0.03	12.21	12.24	0.43
1	6	False	True	0.03	2.01	2.04	0.95
1	6	True	False	0.06	10.98	11.04	0.48
1	6	True	True	0.06	2.03	2.09	0.95
1	8	False	False	0.04	13.13	13.17	0.45
1	8	False	True	0.04	2.93	2.97	0.95
1	8	True	False	0.08	12.99	13.06	0.40
1	8	True	True	0.08	2.44	2.52	0.95
1	24	False	False	0.12	14.64	14.76	0.30
1	24	False	True	0.12	5.54	5.66	0.85
1	24	True	False	0.33	14.61	14.94	0.30
1	24	True	True	0.33	5.06	5.39	0.85
6	6	False	False	0.03	0.06	0.09	1.00
6	6	False	True	0.03	0.18	0.21	1.00
6	6	True	False	0.06	0.06	0.12	1.00
6	6	True	True	0.06	0.17	0.24	1.00
6	8	False	False	0.04	0.08	0.12	1.00
6	8	False	True	0.04	0.19	0.23	1.00
6	8	True	False	0.08	0.09	0.17	1.00
6	8	True	True	0.08	0.22	0.30	1.00
6	24	False	False	0.12	0.29	0.41	1.00
6	24	False	True	0.12	0.57	0.69	1.00
6	24	True	False	0.34	0.32	0.66	1.00
6	24	True	True	0.33	0.72	1.06	1.00

■ **Table 1** Mean computation time for different lengths of problem instances (**len** column), modeling parameters (**clique**, **symmetry**), and solver config on number of search worker (**#w**) We split the table in 2 to distinguish the multi and mono-worker settings of CPSat solver. *Remark* : Each solve call has a timeout of 30 seconds, so when optimality is not proven (like in many tests in mono-mode setting), the solve time is equal to the timeout.

important to note that these results do not constitute a comprehensive benchmark. Moreover, upon closer inspection of the results, we noticed mainly the LP-subsolver implemented in OR-tools contributed to finding a better bound during the search. Hence, it may be interesting to evaluate the performance of LP-specific solvers too such as Gurobi. Overall, further analysis and more extensive testing is required to validate and generalize these findings. Nonetheless, these results are promising and indicate the potential efficiency of CP solvers in handling complex workforce allocation problems.

4.1.2 Explainability results

We run a benchmark study to compute minimal unsatisfiable subset (MUS) conflicts across various scenarios by categorizing the problem constraints into soft and hard constraints directly from our tool interface (see Figure 3). Hard constraints were necessary conditions that must be met, while soft constraints were desirable but not mandatory. This process involved determining which constraints could not be satisfied simultaneously by extracting an MUS. The algorithm used for finding such a MUS is based on the well-known deletion-based

method [25], which extracts any MUS from the problem. As this algorithm greatly benefits incremental solving [2], we used the Exact solver [7], a pseudo-Boolean solver which supports solving under assumptions.

The study was conducted on the same instances introduced in previous sections. The evaluation focuses on the time taken to compute one explanation of infeasibility for each instance and the size of the explanation, measured in terms of the number of constraints involved in the MUS.

Length	Average Time (s)	Average Explanation Length
6	0.60	10
8	0.86	10
24	1.13	10

■ **Table 2** Average Calculation Time and Explanation Length by Instance Length

The results (Table 2) indicate that as the instance length increases from 6 to 24 hours, the average calculation time for generating single explanations of infeasibility also increases (from 0.60s to 1.13s), while the average length of the explanations remains relatively consistent even for bigger instances. These results suggest that longer instances require more computation time, but the complexity of the explanations does not significantly increase. However, further experiments are necessary to draw definitive conclusions.

4.2 Visualising Conflicts and Restoring Feasibility: A Practical Demonstration of our tool

To showcase the capabilities of our explainability techniques and enable their evaluation by end users, we developed a demonstrator application using **Streamlit**¹. This section offers an overview of its features and functionalities.

4.2.1 Solving the Problem

The first step in our application consists of encoding and solving the allocation problem using the **CPMpy** library. Our tool allows to load data and configure various parameters, such as choosing the optimisation solver (see Figure 3). Once the problem is encoded, the solver is called to find an optimal solution. The results, including the allocation of teams to tasks, are then displayed to the user (see Figure 4).

4.2.2 Solution Refinement

After the solver generates a solution, users have the opportunity to review it and modify it. The application allows users to propose alternative allocations overriding the solver's decisions. This interactive review process ensures that users can make adjustments based on their expertise and knowledge of the specific context (see Figure 5).

¹ an open-source app framework for Machine Learning and Data Science projects (<https://streamlit.io/>)

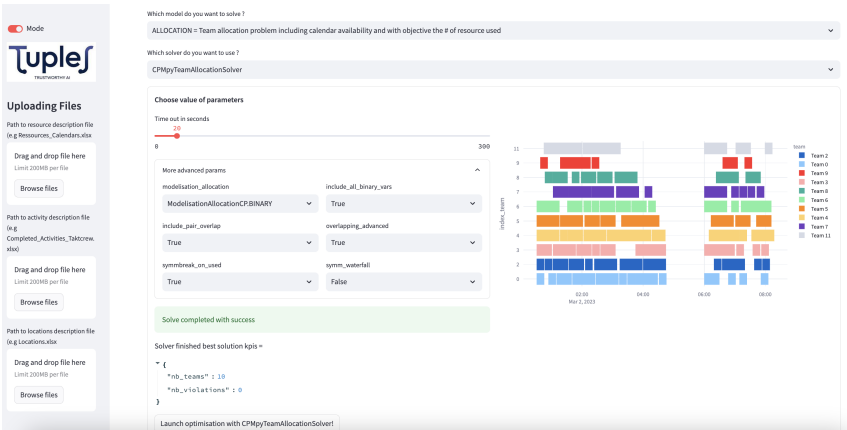


Figure 4 Solving tab of the app, showing the results after calling the solver.

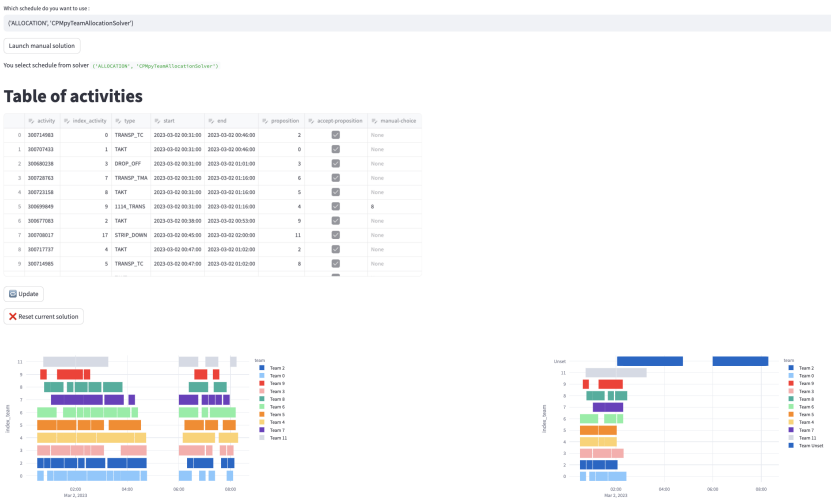


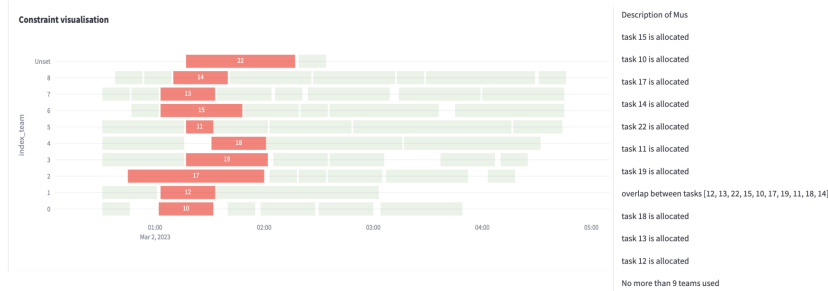
Figure 5 Interactive solving tab (Manual/Automatic)

4.2.3 Conflict Computation and Visualisation

When the problem resolution is infeasible, the application computes and provides a visualisation of the conflicting constraints causing infeasibility. Conflicts are described with a basic text description of each of the constraints, along with a Gantt representation of the problem highlighting the activities involved in the conflict (see Figure 6). The displayed Gantt is built by solving an optimisation problem: it is the result of optimising the number of allocated tasks, e.g. it computes a size-maximal satisfiable subset. These tasks are then visualised, and non-allocated tasks are added to a virtual team we call "Unset", the top line of the plot. This method allows to have a visual representation even when dealing with infeasible problems where no solution (nor visualisation thereof) exists as is.

4.2.4 Feasibility Restoration

If the solver encounters an infeasible problem, our application offers several methods for restoring feasibility. These methods are designed to be interactive by involving the user in



■ **Figure 6** Conflicts visualisation solving tab, with "Unset" line at the top.

357 the resolution process.

358 Local Conflict Resolution

One approach to restoring feasibility is by resolving conflicts one by one interactively. The application identifies a minimum unsatisfiable subset (MUS) of constraints and guides users through the process of addressing each conflict individually. This local resolution method allows users to make targeted adjustments. The process is illustrated in Figure 7. In our preliminary experiment, similarly to the scenario depicted, few iterations were required to restore feasibility, and we surmise that this observation remains true for real scenarios.

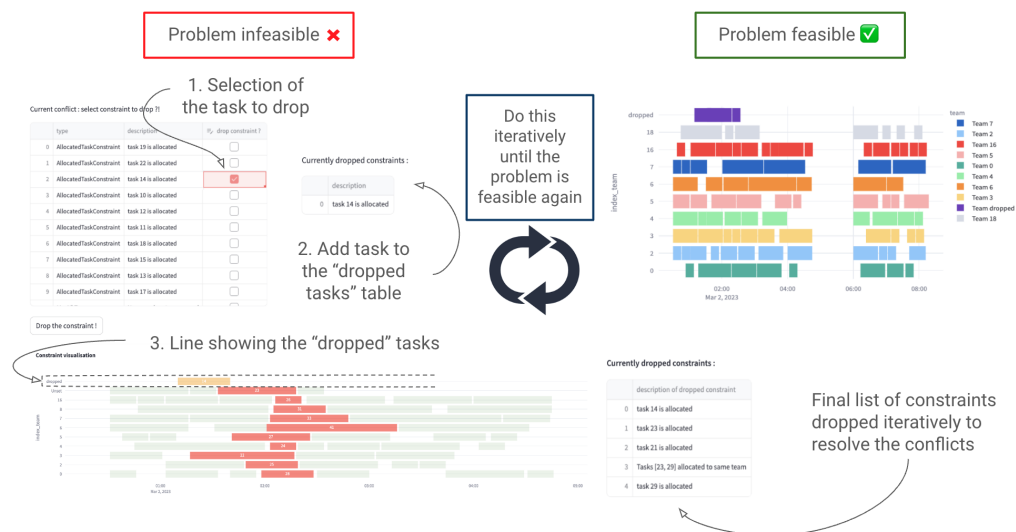


Figure 7 Process of conflict resolution with MUS

365 Use of Minimum Correction Subset

Alternatively, users can employ a minimal correction subset (MCS) to resolve conflicts globally as shown in Figure 8. The application identifies a minimal set of constraints that need to be corrected to restore full feasibility. In the interactive setup, we consider that users can accept to remove only a subset of the constraints proposed by the tool (which would not

completely restore the feasibility). We envisage providing multiple MCSs in the future if none of the corrective actions fits user preference.



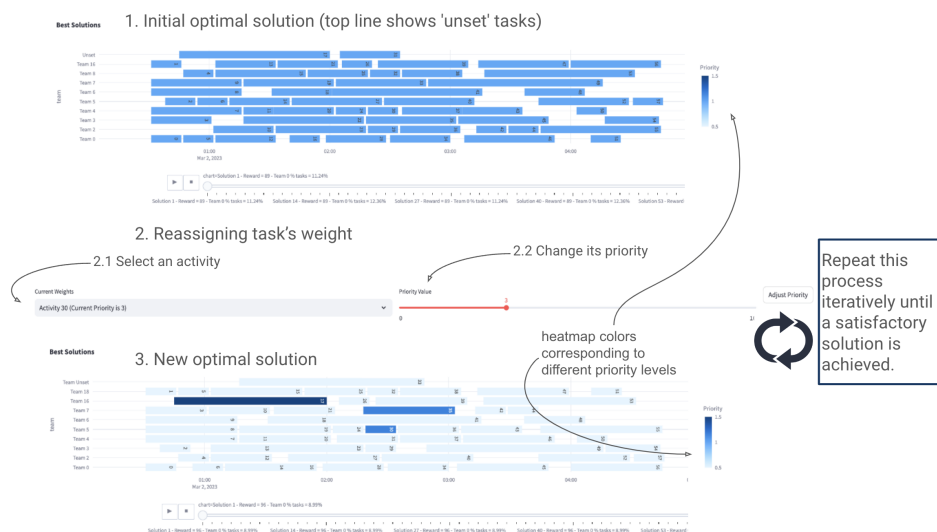
Figure 8 Conflict resolution with MCS

Fine-tuning task priorities

Finally, the application provides the option to solve relaxed versions of the problem using a weighted Max-CSP formulation (optimisation variant of the satisfiability problem where each constraint is assigned a weight, and the goal is to maximize the sum of the weights of the satisfied constraints). We relax the constraint requiring each task to be allocated (constraint 1 of our model) and maximize the sum of allocated activities, where each activity a is weighted by a weight w_a . This method can easily generate several alternative solutions, maximizing the weighted objective. If the user is unhappy with the relaxed solutions, it is possible to interact with the solver by setting different w_a weights on some chosen activities. This method should lead to feasible solutions obtained using domain expert constraint relaxations (see Figure 9).

5 Conclusion & Discussion on Future work

Our decision-making tool for workforce allocation combines the power of constraint programming with interactive and explainable features. By involving users in the decision-making process and providing clear explanations of conflicts and resolutions, we aim to enhance the trust and adoption of CP solvers in industrial settings. The prototype application



■ **Figure 9** Conflict resolution with the fine-tuning task priorities method. The second image shows an example of changed priority/weight for some chosen task, leading to new solver propositions.

demonstrates the practical implementation of these concepts and serves as a foundation for further development and evaluation within the TUPLES project. Our next step in the research is to evaluate the relevance of the generated explanations from a user perspective. These XAI methods should be assessed by expert users who can judge the usability and applicability of XAI/CP technology components in realistic scenarios. Hence, we plan to conduct scientifically rigorous user studies to determine preferred methods for infeasibility restoration. We also plan another user study focused more on a visual interface that will gather user feedback on conflict visualisations and description methods. We are currently implementing various visualisation approaches and textual description techniques to enhance user acceptability.

In this paper, we focused on a pure allocation problem where the activities are already scheduled and can't be shifted in time. In a more realistic model, the possibility of shifting tasks (e.g. changing the start time) in the feasibility restoration step should be considered. However, this would require to transform the model into a scheduling problem, and we are currently working in this direction. This raises interesting scalability challenges for the XAI technology bricks such as MUS computation. To address the interpretability of large conflict explanations, we could consider using step-wise explanations [1]. By breaking down complex explanations into simpler steps, we can create short, interpretable sequences that collectively clarify the issue. Also, in this more complex setup where we consider scheduling constraints, there might be implicit constraints that the planners keep in mind but are not articulated in the problem formulation. Hence we are looking at the techniques from the literature on constraint acquisition [38].

References

- 1 Ignace Bleux, Jo Devriendt, Emilio Gamba, Bart Bogaerts, and Tias Guns. Simplifying step-wise explanation sequences. In *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.

- 415 2 Ignace Bleukx, Tias Guns, and Dimos Tsouros. Explainable Constraint Solving: A hands-on
416 tutorial, February 2024. doi:10.5281/zenodo.10694140.
- 417 3 Ryma Boumazouza, Fahima Cheikh-Alili, Bertrand Mazure, and Karim Tabia. A symbolic
418 approach for counterfactual explanations. In *International Conference on Scalable Uncertainty*
419 *Management*, pages 270–277. Springer, 2020.
- 420 4 Ryma Boumazouza, Fahima Cheikh-Alili, Bertrand Mazure, and Karim Tabia. Asteryx: A
421 model-agnostic sat-based approach for symbolic and score-based explanations. In *Proceedings*
422 *of the 30th ACM International Conference on Information & Knowledge Management*, pages
423 120–129, 2021.
- 424 5 J.W. Chinneck. *Feasibility and Infeasibility in Optimization:: Algorithms and Computational*
425 *Methods*. International Series in Operations Research & Management Science. Springer US,
426 2007. URL: <https://books.google.fr/books?id=BiBgz6AIRpMC>.
- 427 6 Kristijonas Cyras, Ramamurthy Badrinath, Swarup Kumar Mohalik, Anusha Mujumdar,
428 Alexandros Nikou, Alessandro Previti, Vaishnavi Sundararajan, and Aneta Vulgarakis Feljan.
429 Machine reasoning explainability. *arXiv preprint arXiv:2009.00418*, 2020.
- 430 7 Jo Devriendt. Exact solver, 2023. URL: <https://gitlab.com/JoD/exact>.
- 431 8 Jo Devriendt, Bart Bogaerts, Broes De Cat, Marc Denecker, and Christopher Mears. Symmetry
432 propagation: Improved dynamic symmetry breaking in SAT. In *IEEE 24th International*
433 *Conference on Tools with Artificial Intelligence, ICTAI 2012, Athens, Greece, November 7-9,*
434 *2012*, pages 49–56. IEEE Computer Society, 2012. doi:10.1109/ICTAI.2012.16.
- 435 9 Jan Elffers and Jakob Nordström. Divide and conquer: Towards faster pseudo-boolean solving.
436 In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelli-*
437 *gence, IJCAI-18*, pages 1291–1299. International Joint Conferences on Artificial Intelligence
438 Organization, 7 2018. doi:10.24963/ijcai.2018/180.
- 439 10 M Sinan Gönül, Dilek Önköl, and Michael Lawrence. The effects of structural characteristics
440 of explanations on use of a dss. *Decision support systems*, 42(3):1481–1493, 2006.
- 441 11 Tias Guns. Increasing modeling language convenience with a universal n-dimensional array,
442 cppy as python-embedded example. In *Proceedings of the 18th workshop on Constraint*
443 *Modelling and Reformulation at CP (Modref 2019)*, volume 19, 2019.
- 444 12 Sharmi Dev Gupta, Begum Genc, and Barry O’Sullivan. Finding counterfactual explanations
445 through constraint relaxations. *arXiv preprint arXiv:2204.03429*, 2022.
- 446 13 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL: <https://www.gurobi.com>.
- 447 [gurobi.com](https://www.gurobi.com).
- 448 14 Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. PySAT: A Python toolkit for pro-
449 totyping with SAT oracles. In *SAT*, pages 428–437, 2018. doi:10.1007/978-3-319-94144-8_26.
- 450 26.
- 451 15 Alexey Ignatiev, Alessandro Previti, Mark Liffiton, and Joao Marques-Silva. Smallest mus
452 extraction with minimal hitting set dualization. In *International Conference on Principles*
453 *and Practice of Constraint Programming*, pages 173–182. Springer, 2015.
- 454 16 Hilary Johnson and Peter Johnson. Explanation facilities and interactive systems. In *Pro-*
455 *ceedings of the 1st international conference on Intelligent user interfaces*, pages 159–166,
456 1993.
- 457 17 Ulrich Junker. Quickxplain: Conflict detection for arbitrary constraint propagation algorithms.
458 In *IJCAI’01 Workshop on Modelling and Solving problems with constraints*, volume 4, 2001.
- 459 18 Todd Kulesza, Margaret Burnett, Weng-Keen Wong, and Simone Stumpf. Principles of
460 explanatory debugging to personalize interactive machine learning. In *Proceedings of the 20th*
461 *international conference on intelligent user interfaces*, pages 126–137, 2015.
- 462 19 Carmen Lacave and Francisco J Díez. A review of explanation methods for bayesian networks.
463 *The Knowledge Engineering Review*, 17(2):107–127, 2002.
- 464 20 Niklas Lauffer and Ufuk Topcu. Human-understandable explanations of infeasibility for
465 resource-constrained scheduling problems. In *ICAPS 2019 Workshop XAIP*, 2019.

- 466 21 Kevin Leo and Guido Tack. Debugging unsatisfiable constraint models. In *Integration of AI*
467 *and OR Techniques in Constraint Programming: 14th International Conference, CPAIOR*
468 *2017, Padua, Italy, June 5-8, 2017, Proceedings 14*, pages 77–93. Springer, 2017.
- 469 22 Mark H Liffiton, Alessandro Previti, Ammar Malik, and Joao Marques-Silva. Fast, flexible
470 mus enumeration. *Constraints*, 21:223–250, 2016.
- 471 23 Mark H Liffiton and Karem A Sakallah. Algorithms for computing minimal unsatisfiable
472 subsets of constraints. *Journal of Automated Reasoning*, 40:1–33, 2008.
- 473 24 Brian Y Lim and Anind K Dey. Toolkit to support intelligibility in context-aware applications.
474 In *Proceedings of the 12th ACM international conference on Ubiquitous computing*, pages
475 13–22, 2010.
- 476 25 Joao Marques-Silva. Minimal unsatisfiability: Models, algorithms and applications. In *2010*
477 *40th IEEE International Symposium on Multiple-Valued Logic*, pages 9–14. IEEE, 2010.
- 478 26 Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial*
479 *intelligence*, 267:1–38, 2019.
- 480 27 Sina Mohseni, Niloofar Zarei, and Eric D Ragan. A multidisciplinary survey and framework for
481 design and evaluation of explainable ai systems. *ACM Transactions on Interactive Intelligent*
482 *Systems (TüS)*, 11(3-4):1–45, 2021.
- 483 28 Christoph Molnar. *Interpretable machine learning*. Lulu. com, 2020.
- 484 29 Bernard Moulin, Hengameh Irandoust, Micheline Bélanger, and Gaëlle Desbordes. Explanation
485 and argumentation capabilities: Towards the creation of more persuasive agents. *Artificial*
486 *Intelligence Review*, 17(3):169–222, 2002.
- 487 30 Laurent Perron. Or-tools.
- 488 31 Laurent Perron, Frédéric Didier, and Steven Gay. The CP-SAT-LP Solver. In Roland
489 H. C. Yap, editor, *29th International Conference on Principles and Practice of Constraint*
490 *Programming (CP 2023)*, volume 280 of *Leibniz International Proceedings in Informatics*
491 *(LIPIcs)*, pages 3:1–3:2, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum
492 für Informatik. URL: <https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CP.2023.3>, doi:10.4230/LIPIcs.CP.2023.3.
- 493 32 Laurent Perron and Vincent Furnon. Or-tools, 11 2022. URL: <https://developers.google.com/optimization/>.
- 494 33 Alun Preece. Asking ‘why’ in ai: Explainability of intelligent systems—perspectives and
495 challenges. *Intelligent Systems in Accounting, Finance and Management*, 25(2):63–72, 2018.
- 496 34 Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint*
497 *Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006. URL:
498 <https://www.sciencedirect.com/science/bookseries/15746526/2>.
- 499 35 Wojciech Samek, Grégoire Montavon, Andrea Vedaldi, Lars Kai Hansen, and Klaus Robert
500 Müller. Explainable ai—preface. In *Explainable AI: Interpreting, Explaining and Visualizing*
501 *Deep Learning*, pages v–vii. Springer, 2019.
- 502 36 Ilankaikone Senthoooran, Gleb Belov, Kevin Leo, Michael Wybrow, Matthias Klapperstueck,
503 Tobias Czauderna, Mark Wallace, and Maria Garcia De La Banda. Human-centred feasibility
504 restoration. In *International Conference on Principles and Practice of Constraint Programming*
505 *2021*, page 49. Schloss Dagstuhl, 2021.
- 506 37 Kacper Sokol and Peter Flach. Explainability fact sheets: A framework for systematic
507 assessment of explainable approaches. In *Proceedings of the 2020 conference on fairness,*
508 *accountability, and transparency*, pages 56–67, 2020.
- 509 38 Dimosthenis Tsouros, Senne Berden, and Tias Guns. Learning to learn in interactive constraint
510 acquisition. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(8):8154–8162,
511 Mar. 2024. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/28655>, doi:10.
512 1609/aaai.v38i8.28655.
- 513 39 Toby Walsh. General symmetry breaking constraints. In Frédéric Benhamou, editor, *Principles*
514 *and Practice of Constraint Programming - CP 2006, 12th International Conference, CP 2006,*
515

- 517 *Nantes, France, September 25-29, 2006, Proceedings*, volume 4204 of *Lecture Notes in Computer*
518 *Science*, pages 650–664. Springer, 2006. doi:10.1007/11889205_46.