Efficient Reasoning about Infeasible One Machine Sequencing

Raúl Mencía[1], Carlos Mencía[1], Joao Marques-Silva[2]

[1] University of Oviedo, Gijón, Spain
[2] IRIT, CNRS, Toulouse, France
{menciaraul, menciacarlos}@uniovi.es, joao.marques-silva@irit.fr

Abstract

This paper addresses the tasks of explaining and correcting infeasible one machine sequencing problems with a limit on the makespan. Concretely, the paper studies the computation of high-level explanations and corrections, which are given in terms of irreducible subsets of the set of jobs. To achieve these goals, the paper shows that both tasks can be reduced to the general framework of computing a minimal set over a monotone predicate (MSMP). The reductions enable the use of any general-purpose algorithm for solving MSMP, and three well-known approaches are instantiated for the two tasks. Furthermore, the paper details efficient scheduling techniques aimed at enhancing the performance of the proposed algorithms. The experimental results confirm that the proposed approaches are efficient in practice, and that the scheduling optimizations enable critical performance gains.

Introduction

One machine problems play an essential role in the area of scheduling. Besides their many applications, they often act as building blocks of other, more complex, problems. As a consequence, progress in one machine scheduling has enabled the computation of accurate lower bounds, approximations or efficient filtering techniques (Adams, Balas, and Zawack 1988; Brucker, Jurisch, and Sievers 1994; Laborie 2003) in a variety of scheduling domains.

This paper focuses on the classical problem of scheduling a set of jobs on a unary resource (Carlier 1982), in which we impose a limit on the makespan. Such constraint may appear naturally in practice (e.g., a global deadline that must be met in a project). However, if the limit is too tight, the problem may become infeasible, i.e., no possible schedule may exist for it. In this paper, we consider such infeasible problems.

The analysis of infeasibility has been widely studied in the literature, especially in the fields of model-based diagnosis, constraint reasoning or Boolean satisfiability, e.g., see the recent surveys (Marques-Silva and Mencía 2020; Gupta, Genc, and O'Sullivan 2021). In this context, two central reasoning tasks arise: *i*) explaining the causes of infeasibility and *ii*) correcting it by relaxing the problem to some extent. The former has been typically addressed by computing so-called *minimal unsatisfiable subsets* of constraints (MUSes), whereas the latter is usually tackled by computing *minimal correction subsets* (MCSes), i.e., irreducible sets of constraints whose removal renders feasibility, or their complements *maximal satisfiable subsets* (MSSes).

To analyze infeasible one machine scheduling problems, we adopt a *job-based* view, in which jobs can be removed to achieve feasibility (Mencía, Mencía, and Varela 2021; Rodler, Teppan, and Jannach 2021). This enables the definition of high-level explanations and corrections, given in terms of irreducible subsets of the set of jobs. From a user perspective, jobs may be considered as the basic elements the problem is made of, so job-based notions may be useful in this respect. Besides, these may serve as a first step in the computation of finer-grained notions in the analysis of infeasible scheduling problems (Lauffer and Topku 2019).

In this setting, we focus on the tasks of computing a single arbitrary explanation and a single correction. The importance of efficiently solving these tasks should not be understated. Methods for extracting a single set are the core of enumeration algorithms (Liffiton et al. 2016; Narodytska et al. 2018; Bendík and Cerná 2020) which, in turn, are useful to approximate optimization versions of the considered tasks, i.e., finding the *smallest* explanations and corrections.

For this purpose, we first show that both tasks can be reduced to the general framework of computing a minimal set over a monotone predicate (MSMP) (Marques-Silva, Janota, and Mencía 2017). These reductions allow for stating a number of properties of the explanations and corrections studied in this paper, as well as using any general-purpose algorithm for solving MSMP to compute them. In particular, we instantiate three well-known approaches for the two tasks: Deletion (Chinneck and Dravnieks 1991; Bakker et al. 1993), QuickXplain (Junker 2004) and Progression (Marques-Silva, Janota, and Belov 2013). More importantly, building on the large body of research in the analysis of inconsistency in other domains, we develop efficient scheduling techniques aimed at improving performance.

The proposed approaches were implemented by interfacing the constraint programming solver IBM ILOG CP Optimizer (Laborie et al. 2018). The results from an extensive experimental study indicate that both tasks are efficiently solved. Furthermore, the results reveal that the optimizations bring dramatic performance gains, what enables the methods to cope with problem instances with thousands of jobs.

Preliminaries

This section provides the necessary background and definitions that will be used throughout.

One Machine Sequencing

We consider the classical One Machine Sequencing Problem (OMSP) formulated in (Carlier 1982). It consists in scheduling n jobs $\mathcal{J} = \{1, ..., n\}$ on a single machine. Each job i is available at its release date $r_i \geq 0$, and has a duration $p_i > 0$ and a tail $q_i \geq 0$, indicating the amount of time the job remains in the system after it has been processed.

A schedule S is an assignment of a starting time st_i to each job i satisfying the following constraints: *i*) Jobs cannot start before their release dates, i.e., $st_i \geq r_i$ for all $i \in \mathcal{J}$; *ii*) Only one job can be processed at a time: $(st_i + p_i \leq st_j) \vee (st_j + p_j \leq st_i)$ for all $i \neq j \in \mathcal{J}$; and *iii*) Jobs cannot be preempted.

Different metrics can be used to assess the quality of a schedule (Brucker and Knust 2006), and these are often used as an objective to optimize. Carlier (1982) studied the minimization of the makespan, defined for a schedule S as $C_{max}(S) = \max_{i \in \mathcal{J}}(st_i + p_i + q_i)$.

The decision version of the OMSP asks whether a schedule S exists with $C_{max}(S) \leq C$, where C is a limit on the makespan. If such schedule exists, the problem instance is said *feasible*. Otherwise, if C is too low and such schedule does not exist, the instance is said *infeasible*. This decision problem is NP-complete (Garey and Johnson 1979).

Throughout, we will consider infeasible problem instances. These will be denoted as a pair $(\mathcal{J}, C)$.

Example 1. *Figure 1 shows an optimal schedule for a set of jobs $\mathcal{J} = \{1, 2, 3, 4\}$ (r_i, p_i and q_i are shown in the chart for each job). The optimal makespan is 20 so, for any value $C < 20$, the instance $(\mathcal{J}, C)$ is infeasible.*

Job-based Explanations and Corrections

In this paper, we address the tasks of explaining and correcting the infeasibility of OMSP instances. For this purpose, we focus on the computation of *high-level* explanations and corrections, defined in terms of irreducible subsets of the set of jobs. Given an infeasible problem instance $(\mathcal{J}, C)$, the following definitions apply:

Definition 1. *(MISJ) $\mathcal{M} \subseteq \mathcal{J}$ is a* minimal infeasible subset of jobs *(MISJ) if and only if $(\mathcal{M}, C)$ is infeasible and for all $\mathcal{M}' \subsetneq \mathcal{M}$, $(\mathcal{M}', C)$ is feasible.*

Definition 2. *(MCSJ) $\mathcal{C} \subseteq \mathcal{J}$ is a* minimal correction subset of jobs *(MCSJ) if and only if $(\mathcal{J} \setminus \mathcal{C}, C)$ is feasible and for all $\mathcal{C}' \subsetneq \mathcal{C}$, $(\mathcal{J} \setminus \mathcal{C}', C)$ is infeasible.*

Definition 3. *(MFSJ) $\mathcal{F} \subsetneq \mathcal{J}$ is a* maximal feasible subset of jobs *(MFSJ) if and only if $(\mathcal{F}, C)$ is feasible and for all $\mathcal{F} \subsetneq \mathcal{F}' \subseteq \mathcal{J}$, $(\mathcal{F}', C)$ is infeasible.*

These definitions are analogous to MUSes, MCSes and MSSes respectively. An MISJ is a subset-minimal set of jobs that cannot be scheduled within the makespan limit. On the other hand, an MCSJ is an irreducible subset of jobs whose removal enables to schedule the remaining jobs without exceeding the limit. The complement of an MCSJ is an MFSJ.

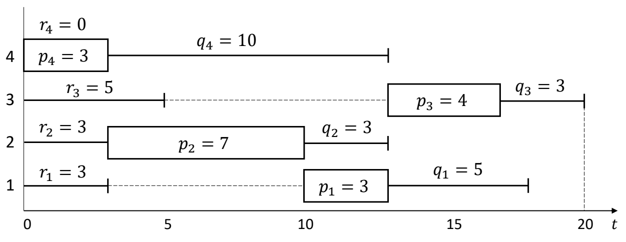

Figure 1: Optimal schedule for $\mathcal{J} = \{1, 2, 3, 4\}$

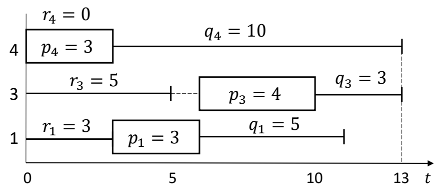

Figure 2: Schedule for the MFSJ $\{1, 3, 4\}$

Example 2. *Consider the infeasible instance $(\mathcal{J}, C)$, with $\mathcal{J}$ as in Example 1 and $C = 15$. There are two MISJs: $\{1, 2\}$ and $\{2, 3\}$ (these sets cannot be scheduled with a makespan not exceeding 15). The MCSJs are $\{2\}$ and $\{1, 3\}$, with the MFSJs $\{1, 3, 4\}$ and $\{2, 4\}$ (for each of these sets there exists a schedule with makespan not greater than 15). Figure 2 shows a schedule for the MFSJ $\{1, 3, 4\}$, with makespan 13.*

The notions above can be generalized to a setting in which the set of jobs is partitioned as $\mathcal{J} = \{\mathcal{B}, \mathcal{S}\}$, where $\mathcal{B}$ are *background* jobs, that always have to be scheduled, and $\mathcal{S}$ are jobs that can be removed. In this context, MISJs and MCSJs would be defined as irreducible subsets of $\mathcal{S}$.

Minimal Sets over a Monotone Predicate (MSMP)

Several problems are instances of the unifying framework of computing a minimal set over a monotone predicate (MSMP), including problems related to the analysis of inconsistency (Marques-Silva, Janota, and Belov 2013; Marques-Silva, Janota, and Mencía 2017).

A predicate $P : 2^{|\mathcal{R}|} \to \{0, 1\}$, defined over a reference set $\mathcal{R}$, is *monotone* iff whenever $P(\mathcal{R}_0)$ holds, with $\mathcal{R}_0 \subseteq \mathcal{R}$, then $P(\mathcal{R}_1)$ holds as well, for all $\mathcal{R}_0 \subseteq \mathcal{R}_1 \subseteq \mathcal{R}$.

Minimal sets over a monotone predicate exhibit a useful property: $\mathcal{M} \subseteq \mathcal{R}$ is a minimal set over a monotone predicate P iff $P(\mathcal{M})$ holds and, for all $u \in \mathcal{M}$, $P(\mathcal{M} \backslash \{u\})$ does not hold. Thus, checking the minimality of a set requires a worst-case linear number of predicate tests, in contrast to testing all (exponentially many) subsets of $\mathcal{M}$.

Throughout, w.l.o.g. we will consider that the elements in a set follow a fixed arbitrary order. We might use subscripts to indicate subsets, expressing ranges indexed from 1 on: for a set $\mathcal{T} = \{u_1, ...u_{|\mathcal{T}|}\}$, $\mathcal{T}_{i..j} = \{u_i, ..., u_j\}$, with $i \leq j$.

A variety of algorithms can be used to solve the MSMP problem. Given a monotone predicate P and a reference set $\mathcal{R}$, these methods assume that $P(\mathcal{R})$ holds, and they differ in the way they search for so-called *transition* or *necessary* elements, defined as follows:

Algorithm 1: Deletion for MSMP

```
    Input: $P$: Monotone predicate, $\mathcal{R}$: Reference set
    Output: $\mathcal{M}$: Minimal set over $P$
1   $\mathcal{M} \leftarrow \mathcal{R}$
2   foreach $u \in \mathcal{M}$ do
3       if $P(\mathcal{M} \setminus \{u\})$ then
4           $\mathcal{M} \leftarrow \mathcal{M} \setminus \{u\}$
5   return $\mathcal{M}$
```

Algorithm 2: QuickXplain for MSMP

```
    Input: $P$: Monotone predicate, $\mathcal{R}$: Reference set
    Output: $\mathcal{M}$: Minimal set over $P$
1   $\mathcal{M} \leftarrow$ QXRec$(P, \emptyset, \emptyset, \mathcal{R})$
2   return $\mathcal{M}$

3   Function QXRec ($P$, $\mathcal{B}$, $\mathcal{D}$, $\mathcal{T}$)
        Output: Minimal set over $P$ w.r.t. $\mathcal{B}$
4       if $\mathcal{D} \neq \emptyset \wedge P(\mathcal{B})$ then return $\emptyset$
5       if $|\mathcal{T}| = 1$ then return $\mathcal{T}$
6       $m \leftarrow \lfloor \frac{|\mathcal{T}|}{2} \rfloor$
7       $(\mathcal{T}_1, \mathcal{T}_2) \leftarrow (\mathcal{T}_{1..m}, \mathcal{T}_{m+1..|\mathcal{T}|})$
8       $\mathcal{M}_2 \leftarrow$ QXRec$(P, \mathcal{B} \cup \mathcal{T}_1, \mathcal{T}_1, \mathcal{T}_2)$
9       $\mathcal{M}_1 \leftarrow$ QXRec$(P, \mathcal{B} \cup \mathcal{M}_2, \mathcal{M}_2, \mathcal{T}_1)$
10      return $\mathcal{M}_1 \cup \mathcal{M}_2$
```

Definition 4. *Given $\mathcal{U} \subseteq \mathcal{R}$, $t \in \mathcal{U}$ is a transition element for P iff $P(\mathcal{U})$ holds and $P(\mathcal{U} \setminus \{t\})$ does not hold.*

Whenever a transition element $t \in \mathcal{U}$ is found, it holds that t must belong to all minimal sets over P contained in $\mathcal{U}$, so it is deemed to belong to the minimal set being computed.

In this paper, we instantiate three of such algorithms for computing MISJs and MCSJs of infeasible OMSP instances.

Deletion Arguably, the simplest approach is Deletion (Chinneck and Dravnieks 1991; Bakker et al. 1993), shown in Algorithm 1. First, the set $\mathcal{M}$ is initialized with all the elements in $\mathcal{R}$. Then, for each $u \in \mathcal{M}$, it tests whether the predicate still holds after dropping u. If it does, u is discarded (notice that $\mathcal{M} \setminus \{u\}$ contains a minimal set over P). Otherwise, u is a transition element, so it is kept in $\mathcal{M}$. Upon termination, $\mathcal{M}$ is a minimal set over P. This algorithm requires $\mathcal{O}(m)$ predicate tests, with $m = |\mathcal{R}|$.

QuickXplain QuickXplain (Junker 2004) is a divide-and-conquer approach. It is based on the following principle: Let $\{\mathcal{T}_1, \mathcal{T}_2\}$ be a partition of $\mathcal{R}$, $\mathcal{M}_2 \subseteq \mathcal{T}_2$ a minimal set such that $P(\mathcal{T}_1 \cup \mathcal{M}_2)$ holds (i.e., a minimal subset of $\mathcal{T}_2$ that together with $\mathcal{T}_1$ makes P hold), and $\mathcal{M}_1 \subseteq \mathcal{T}_1$ a minimal set such that $P(\mathcal{M}_2 \cup \mathcal{M}_1)$ holds (i.e., a minimal subset of $\mathcal{T}_1$ that together with $\mathcal{M}_2$ makes P hold). Then, $\mathcal{M}_1 \cup \mathcal{M}_2$ is a minimal set over P.

Following this principle, the reference set is partitioned into two sets $\mathcal{T}_1$ and $\mathcal{T}_2$, which are then recursively simplified, using $\mathcal{T}_1$ as a reference for simplifying $\mathcal{T}_2$ and then using the resulting $\mathcal{M}_2$ as a reference for simplifying $\mathcal{T}_1$. Algorithm 2 shows its pseudocode. It relies on an invocation to the procedure QXRec$(P, \mathcal{B}, \mathcal{D}, \mathcal{T})$, which recursively computes a minimal set $\mathcal{T}' \subseteq \mathcal{T}$ such that $P(\mathcal{B} \cup \mathcal{T}')$ holds. In each invocation, $P(\mathcal{B} \cup \mathcal{T})$ holds, and the set $\mathcal{D}$ contains the latest elements added to $\mathcal{B}$. If $\mathcal{D} \neq \emptyset$ and $P(\mathcal{B})$ holds, the empty set is returned, meaning that $\mathcal{B} = \mathcal{B} \cup \mathcal{D}$ contains a minimal set over P. Otherwise, if $\mathcal{T} = \{t\}$ then t is a transition element, which is returned; if $|\mathcal{T}| > 1$, the set $\mathcal{T}$ is split and two recursive calls are made.

QuickXplain requires worst-case $\mathcal{O}(k + k\log(\frac{m}{k}))$ predicate tests, where $m = |\mathcal{R}|$ and k is the size of the largest minimal set over P. So it is more efficient when k is small.

Algorithm 3: Progression for MSMP

```
    Input: $P$: Monotone predicate, $\mathcal{R}$: Reference set
    Output: $\mathcal{M}$: Minimal set over $P$
1   $(\mathcal{M}, \mathcal{T}, \nu) \leftarrow (\emptyset, \mathcal{R}, 1)$
2   while $\mathcal{T} \neq \emptyset$ do
3       $\nu \leftarrow \min(\nu, |\mathcal{T}|)$
4       if $P(\mathcal{M} \cup (\mathcal{T} \setminus \mathcal{T}_{1..\nu}))$ then
5           $(\mathcal{T}, \nu) \leftarrow (\mathcal{T} \setminus \mathcal{T}_{1..\nu}, 2 \times \nu)$
6       else
7           $j \leftarrow$ BinSearch$(P, \mathcal{M}, \mathcal{T}, \nu)$
8           $(\mathcal{M}, \mathcal{T}, \nu) \leftarrow (\mathcal{M} \cup \mathcal{T}_{j..j}, \mathcal{T} \setminus \mathcal{T}_{j..j}, 1)$
9   return $\mathcal{M}$

10  Function BinSearch ($P$, $\mathcal{M}$, $\mathcal{T}$, $\nu$)
        Output: $r$: Index of transition element
11      $(l, r) \leftarrow (0, \nu)$
12      while $l < r - 1$ do
13          $m \leftarrow \lfloor \frac{l+r}{2} \rfloor$
14          if $P(\mathcal{M} \cup (\mathcal{T} \setminus \mathcal{T}_{1..m}))$ then
15              $l \leftarrow m$
16          else
17              $r \leftarrow m$
18      return $r$
```

Progression Alternatively, Progression (Marques-Silva, Janota, and Belov 2013) looks for each transition element following an *exponential search* approach.

It is shown in Algorithm 3. It manages two sets $\mathcal{M}$ and $\mathcal{T}$, as well as an integer ν initialized to value 1. Initially $\mathcal{M} = \emptyset$, and this set will grow until eventually representing a minimal set over P. This set is included in all the predicate tests. The set $\mathcal{T}$, initialized with all the elements in $\mathcal{R}$, contains the elements unknown to belong or not to the minimal set being computed. Iteratively, until $\mathcal{T} = \emptyset$, the algorithm tests whether the predicate still holds after dropping the first ν elements of $\mathcal{T}$ (i.e., $\mathcal{T}_{1..\nu}$). If it does, $\mathcal{M} \cup \mathcal{T}_{\nu+1..|\mathcal{T}|}$ contains a minimal set over P, so the elements in $\mathcal{T}_{1..\nu}$ are discarded and the value of ν is doubled. Otherwise, $\mathcal{T}_{1..\nu}$ contains a transition element, which is identified by means of a binary search procedure. After it is found, it is moved to $\mathcal{M}$ and the value of ν is reset to 1.

Progression requires a worst-case $\mathcal{O}(k\ \log(1 + \frac{m}{k}))$ number of predicate tests, with m and k defined as before. As QuickXplain, it is expected to be more efficient when k is small. A similar algorithm was proposed in (Laborie 2014).

Reasoning about Infeasible OMS Problems

This section reduces the computation of MISJs and MCSJs to the MSMP framework. The reductions enable to state a number of properties and to use a variety of algorithms.

Reductions to MSMP

The following monotonicity results will be used throughout:

Proposition 1. *Let $(\mathcal{F}, C)$ be a feasible problem instance. Then, $(\mathcal{F}', C)$ is feasible for all $\mathcal{F}' \subseteq \mathcal{F}$.*

Proof. There is a schedule S for the jobs in $\mathcal{F}$ with $C_{max}(S) \leq C$. Given the jobs $\mathcal{F}' \subseteq \mathcal{F}$, build a schedule S' setting their starting times as in S. Then, $C_{max}(S') \leq C_{max}(S) \leq C$, and thus $(\mathcal{F}', C)$ is feasible. $\square$

Proposition 2. *Let $(\mathcal{I}, C)$ be an infeasible problem instance. Then, $(\mathcal{I}', C)$ is infeasible for all $\mathcal{I} \subseteq \mathcal{I}'$.*

Proof. Suppose there exists a set $\mathcal{I}'$, with $\mathcal{I} \subseteq \mathcal{I}'$, such that $(\mathcal{I}', C)$ is feasible. By Proposition 1, for all $\mathcal{I}'' \subseteq \mathcal{I}'$, $(\mathcal{I}'', C)$ is feasible, including $(\mathcal{I}, C)$. A contradiction. $\square$

Remark 1. *The proofs above focus on the makespan, but could be easily adapted to show that the monotonicity results hold for many other regular metrics (non-decreasing with the completion times). Examples are total completion time or maximum/total lateness/tardiness (if due dates are considered). However, these may not hold for other regular, e.g., average completion time, or non-regular metrics.*

What follows holds for any setting in which the monotonicity properties proven above hold. We now reduce the computation of MISJs and MCSJs to MSMP. In both cases we define a predicate, show it is monotone and that any minimal set over it is an MISJ or an MCSJ respectively.

Proposition 3. *Given an infeasible problem $(\mathcal{J}, C)$, computing an MISJ of $\mathcal{J}$ is an instance of MSMP.*

Proof. Define $P_{\text{ISJ}}(\mathcal{W}, C) \triangleq \neg\texttt{Feasible}(\mathcal{W}, C)$, with $\mathcal{W} \subseteq \mathcal{R}$ and $\mathcal{R} \triangleq \mathcal{J}$.
Monotonicity: Let $\mathcal{I} \subseteq \mathcal{R}$ be such that $P_{\text{ISJ}}(\mathcal{I}, C)$ holds, i.e., $(\mathcal{I}, C)$ is infeasible. By Proposition 2, for all $\mathcal{I} \subseteq \mathcal{I}' \subseteq \mathcal{R}$, $(\mathcal{I}', C)$ is infeasible, so $P_{\text{ISJ}}(\mathcal{I}', C)$ holds.
Correctness: Let $\mathcal{I}$ be a minimal set for which $P_{\text{ISJ}}(\mathcal{I}, C)$ holds, i.e., $(\mathcal{I}, C)$ is infeasible. Since $\mathcal{I}$ is minimal, for all $\mathcal{I}' \subsetneq \mathcal{I}$, $P_{\text{ISJ}}(\mathcal{I}', C)$ does not hold, i.e., $(\mathcal{I}', C)$ is feasible. Thus, by Definition 1, $\mathcal{I}$ is an MISJ. $\square$

Proposition 4. *Given an infeasible problem $(\mathcal{J}, C)$, computing an MCSJ of $\mathcal{J}$ is an instance of MSMP.*

Proof. Define $P_{\text{CSJ}}(\mathcal{W}, C) \triangleq \texttt{Feasible}(\mathcal{R} \setminus \mathcal{W}, C)$, with $\mathcal{W} \subseteq \mathcal{R}$ and $\mathcal{R} \triangleq \mathcal{J}$.
Monotonicity: Let $\mathcal{C} \subseteq \mathcal{R}$ be such that $P_{\text{CSJ}}(\mathcal{C}, C)$ holds, i.e., $(\mathcal{J} \setminus \mathcal{C}, C)$ is feasible. By Proposition 1, for all $\mathcal{C} \subseteq \mathcal{C}' \subseteq \mathcal{R}$, $(\mathcal{J} \setminus \mathcal{C}', C)$ is feasible (note that $\mathcal{J} \setminus \mathcal{C}' \subseteq \mathcal{J} \setminus \mathcal{C}$), so $P_{\text{CSJ}}(\mathcal{C}', C)$ holds.
Correctness: Let $\mathcal{C}$ be a minimal set for which $P_{\text{CSJ}}(\mathcal{C}, C)$ holds, i.e., $(\mathcal{J} \setminus \mathcal{C}, C)$ is feasible. Since $\mathcal{C}$ is minimal, for all $\mathcal{C}' \subsetneq \mathcal{C}$, $P_{\text{CSJ}}(\mathcal{J} \setminus \mathcal{C}', C)$ does not hold, i.e., $(\mathcal{J} \setminus \mathcal{C}', C)$ is infeasible. Thus, by Definition 2, $\mathcal{C}$ is an MCSJ. $\square$

Algorithm 4: Deletion for MISJ extraction

```
   Input: $\mathcal{J}$: Set of jobs, $C$: Makespan limit
   Output: $\mathcal{I}$: MISJ of $\mathcal{J}$
1  $\mathcal{I} \leftarrow \mathcal{J}$
2  foreach $u \in \mathcal{I}$ do
3      if not isFeasible($\mathcal{I} \setminus \{u\}, C$) then
4          $\mathcal{I} \leftarrow \mathcal{I} \setminus \{u\}$
5  return $\mathcal{I}$
```

Algorithm 5: Linear Search for MCSJ extraction

```
   Input: $\mathcal{J}$: Set of jobs, $C$: Makespan limit
   Output: MCSJ of $\mathcal{J}$
1  $(\mathcal{F}, \mathcal{U}) \leftarrow (\emptyset, \mathcal{J})$
2  while $\mathcal{U} \neq \emptyset$ do
3      Pick a job $j \in \mathcal{U}$
4      $\mathcal{U} \leftarrow \mathcal{U} \setminus \{j\}$
5      if isFeasible($\mathcal{F} \cup \{j\}, C$) then
6          $\mathcal{F} \leftarrow \mathcal{F} \cup \{j\}$
7  return $\mathcal{J} \setminus \mathcal{F}$
```

By these reductions, both kinds of sets exhibit a property previously mentioned. $\mathcal{I} \subseteq \mathcal{J}$ is an MISJ iff $(\mathcal{I}, C)$ is infeasible and for all $u \in \mathcal{I}$, $(\mathcal{I} \setminus \{u\}, C)$ is feasible. Analogously, $\mathcal{C} \subseteq \mathcal{J}$ is an MCSJ iff $(\mathcal{J} \setminus \mathcal{C}, C)$ is feasible and for all $u \in \mathcal{C}$, $(\mathcal{J} \setminus (\mathcal{C} \setminus \{u\}), C)$ is infeasible.

Besides, P_{ISJ} and P_{CSJ} (defined in the proofs) are dual predicates: the former tests the infeasibility of a set, whereas the latter tests the feasibility of the complement of the set. As a result, a relationship can be established – see Theorem 1 in (Marques-Silva, Janota, and Mencía 2017) – which is well-known to hold between MUSes and MCSes of inconsistent systems (Reiter 1987; Birnbaum and Lozinskii 2003): Every MISJ is a *minimal hitting set* of all MCSJs, and vice versa. This enables to use minimal hitting set duality algorithms to enumerate both kinds of sets, e.g., (Liffiton and Sakallah 2008; Liffiton et al. 2016).

Computing one MISJ / MCSJ

Any general-purpose MSMP algorithm can be used to compute an MISJ or an MCSJ of a given infeasible problem $(\mathcal{J}, C)$, by just using the corresponding predicate and the set of jobs $\mathcal{J}$ as reference set. To test the predicates we need a decision procedure, that we name $\texttt{isFeasible}(\mathcal{J}, C)$, which determines if the instance $(\mathcal{J}, C)$ is feasible or not.

As an example, Algorithm 4 shows the Deletion approach for computing an MISJ. To compute an MCSJ, it suffices to replace the condition in line 3 to test the feasibility of the complementary set of jobs. However, when applied to repairing inconsistency, Deletion is usually presented differently, and named Linear Search (LS) (Bailey and Stuckey 2005). It is shown in Algorithm 5. LS keeps the sets $\mathcal{F}$, an under-approximation of an MFSJ, and $\mathcal{U}$ with jobs to be tested. Iteratively, until $\mathcal{U} = \emptyset$, it removes a job $j \in \mathcal{U}$ and tests if $(\mathcal{F} \cup \{j\}, C)$ is feasible. If it is, $\mathcal{F}$ is extended with j. On termination, $\mathcal{F}$ is an MFSJ and so $\mathcal{J} \setminus \mathcal{F}$ is an MCSJ.

Optimizations

To improve performance, we build on a number of techniques originally proposed for computing MUSes and MCSes of unsatisfiable propositional formulas. These are aimed at reducing the number of feasibility tests by using information given by the underlying solver. This section shows how to efficiently lift these methods to the scheduling domain and proposes new ones to avoid expensive feasibility tests.

Building Blocks

The following procedures will be used as building blocks to enhance the extraction of both MISJs and MCSJs.

Fitting jobs into schedules Throughout, we assume that the decision procedure returns a witness (i.e., a schedule), whenever it is invoked on a feasible instance. An invocation will be represented as $(res, S) \leftarrow \texttt{isFeasible}(\mathcal{F}, C)$, with res a Boolean indicating the feasibility of $(\mathcal{F}, C)$ and S a schedule for the jobs in $\mathcal{F}$ when the instance is feasible.

Given such a schedule S, we describe an incomplete procedure to determine whether a job $j \notin \mathcal{F}$ can be scheduled, so that efficiently proving $(\mathcal{F} \cup \{j\}, C)$ feasible. The proposed method follows an *insertion-based* approach, that looks for a position (w.r.t. the order of the jobs in S) in which j could be placed while satisfying all the constraints.

We assume that S will be given as an assignment of a starting time to each job in $\mathcal{F}$. This assignment induces a sequence (total order) of the jobs $\sigma = (\sigma_1, \sigma_2, ..., \sigma_{|\mathcal{F}|})$, where $st_{\sigma_i} < st_{\sigma_j}$ for all $i < j$. Obtaining σ from S can be done in $\mathcal{O}(|\mathcal{F}| \times \log |\mathcal{F}|)$, since only sorting is necessary.

Then, a *left-shifted* schedule L_σ can be obtained in $\mathcal{O}(|\sigma|)$ by traversing σ and scheduling the jobs as early as possible. As a result, $C_{max}(L_\sigma) \leq C$. To this aim, we use a procedure $\texttt{ComputeSchedule}(\sigma)$: the first job σ_1 is scheduled at its release date, i.e., $st_{\sigma_1} = r_{\sigma_1}$, whereas $st_{\sigma_i} = \max(r_{\sigma_i}, st_{\sigma_{(i-1)}} + p_{\sigma_{(i-1)}})$ for $i > 1$. In addition, this procedure computes the latest possible starting time lst_{σ_i} to which each job could be delayed (possibly delaying the next jobs in σ too) with makespan not exceeding C. This is done by traversing σ in reverse: for the last job $\sigma_{|\sigma|}$, $lst_{\sigma_{|\sigma|}} = C - q_{\sigma_{|\sigma|}} - p_{\sigma_{|\sigma|}}$ and, for $i < |\sigma|$, $lst_{\sigma_i} = \min(C - q_{\sigma_i} - p_{\sigma_i}, lst_{\sigma_{(i+1)}} - p_{\sigma_i})$.

Now, we can efficiently check whether a job $j \notin \mathcal{F}$ can be *inserted* into σ, resulting in a sequence $\sigma' = (\sigma_1, ..., \sigma_{(i-1)}, j, \sigma_i, ..., \sigma_{|\sigma|})$ such that $L_{\sigma'} \leq C$. By using the st_i and lst_i values, this operation can be performed without building $L_{\sigma'}$. We define a procedure $\texttt{Fits}(j, \sigma, C)$, that returns the first position in which j could be inserted (or 0 if no such position is found). After computing a schedule from σ, this procedure iterates from position 1 to $|\sigma| + 1$. At the i-th iteration, it checks whether j can be inserted right before σ_i. To this aim, it computes a tentative starting time for j as $st'_j = \max(r_j, st_{\sigma_{(i-1)}} + p_{\sigma_{(i-1)}})$ (when $i = 1$, $st'_j = r_j$). If $st'_j + p_j + q_j \leq C$ and $st'_j + p_j \leq lst_{\sigma_i}$ (the second condition is not tested for $i = |\sigma| + 1$), it returns the position i. Otherwise, the next position is tested. If eventually no position is found, the value 0 is returned, indicating that the job j cannot be inserted into σ. This process runs in $\mathcal{O}(|\sigma|)$.

Lower bounds Given a set of jobs $\mathcal{J}$, a simple lower bound on the optimal makespan is given by $\texttt{SLB}(\mathcal{J}) = \min_{j \in \mathcal{J}} r_j + \sum_{j \in \mathcal{J}} p_j + \min_{j \in \mathcal{J}} q_j$. A more accurate lower bound can be computed as $\texttt{LB}(\mathcal{J}) = \max_{\mathcal{J}' \subseteq \mathcal{J}} \texttt{SLB}(\mathcal{J}')$, by considering all subsets $\mathcal{J}' \subseteq \mathcal{J}$. It is well-known that $\texttt{LB}(\mathcal{J})$ can be computed in $\mathcal{O}(n \log n)$, with $n = |\mathcal{J}|$, by relaxing the non-preemption constraints and computing a so-called Jackson's Preemptive Schedule (JPS) (Carlier 1982). In the JPS, jobs are preemptively scheduled by non-increasing tails, breaking ties arbitrarily. The makespan of such schedule equals $\texttt{LB}(\mathcal{J})$.

Lower bounds can be used to improve the efficiency of both MISJ and MCSJ extraction. Given an instance $(\mathcal{W}, C)$, if $\texttt{LB}(\mathcal{W}) > C$, then it is declared infeasible, avoiding an invocation to the (expensive) decision procedure.

Boosting MISJ Extraction

During the computation of an MISJ for an infeasible instance $(\mathcal{J}, C)$, the set of jobs is implicitly partitioned as $\mathcal{J} = \{\mathcal{I}, \mathcal{D}, \mathcal{T}\}$, where $\mathcal{I}$ contains jobs included into the MISJ under construction, $\mathcal{D}$ are the jobs that have been discarded, and $\mathcal{T}$ contains jobs still unknown to belong or not to the MISJ. The invariant that $(\mathcal{I} \cup \mathcal{T})$ is infeasible holds.

If at a given step, for a job $t \in \mathcal{T}$, $(\mathcal{I} \cup \mathcal{T} \setminus \{t\}, C)$ is feasible, then it holds that t must belong to all MISJs contained in $\mathcal{I} \cup \mathcal{T}$. In this case, t is a *transition job*.

Consider a schedule S for $(\mathcal{I} \cup \mathcal{T} \setminus \{t\}, C)$ and a job $j \in \mathcal{T}$ with $j \neq t$. If from S we can obtain a schedule S', with $C_{max}(S') \leq C$, for the instance $(\mathcal{I} \cup \mathcal{T} \setminus \{j\}, C)$, that would be a proof that j is also a transition job. Such a schedule S' can be efficiently searched for. First, from S we get the induced sequence of jobs σ, and define the new sequence $\sigma' = \sigma \setminus \{j\}$. Then, if $\texttt{Fits}(t, \sigma', C)$ returns a position greater than 0, the job t can be inserted in the schedule $L_{\sigma'}$, thus proving that j is a transition job as well, i.e., $(\mathcal{I} \cup \mathcal{T} \setminus \{j\}, C)$ is feasible. The process can be done for all $j \in \mathcal{T}$.

This method is inspired in the *model rotation* technique (Belov, Lynce, and Marques-Silva 2012; Wieringa 2012) proposed for speeding up the computation of MUSes of unsatisfiable propositional formulas. We call it *job rotation*. It is shown in Algorithm 6. Each transition job discovered is added to the set $\mathcal{N}$, which is returned. This procedure runs in $\mathcal{O}(n^2)$, with $n = |\mathcal{I} \cup \mathcal{T}|$.

An enhanced version of the Deletion algorithm, exploiting both lower bounds and job rotation, is shown in Algorithm 7. If at an iteration $\texttt{LB}(\mathcal{I} \cup \mathcal{T} \setminus \{t\}) > C$, the job t is discarded, saving a call to the decision procedure. Besides, after identifying a transition job, job rotation is issued to discover new transition jobs $\mathcal{N} \subseteq \mathcal{T}$, which are moved from $\mathcal{T}$ to $\mathcal{I}$.

Regarding QuickXplain, lower bounds can be easily integrated as well. However, due to its recursive nature (see Algorithm 2), the application of job rotation is restricted to the elements in the set $\mathcal{T}$ (which, in this case, is *local* to the invocation) after having identified a transition job in the previous call to the recursive algorithm. Progression does not suffer from this limitation. In this case, a transition job is identified after the binary search (see Algorithm 3), and job rotation can be applied over the whole (*global*) set $\mathcal{T}$.

Algorithm 6: Procedure JobRotation

```
    Input: $\mathcal{I}, \mathcal{T}$: Sets of jobs, $t$: Transition job, $S$:
           Schedule for $\mathcal{I} \cup \mathcal{T} \setminus \{t\}$, $C$: Limit
    Output: $\mathcal{N}$: Discovered transition jobs
1   $(\sigma, \mathcal{N}) \leftarrow$ (GetSequence$(S), \emptyset)$
2   foreach $j \in \mathcal{T}$ do
3       $\sigma' \leftarrow \sigma \setminus \{j\}$
4       if Fits$(t, \sigma', C)$ then $\mathcal{N} \leftarrow \mathcal{N} \cup \{j\}$
5   return $\mathcal{N}$
```

Algorithm 7: Enhanced Deletion (MISJ extraction)

```
    Input: $\mathcal{J}$: Set of jobs, $C$: Makespan limit
    Output: $\mathcal{I}$: MISJ of $\mathcal{J}$
1   $(\mathcal{I}, \mathcal{T}) \leftarrow (\emptyset, \mathcal{J})$
2   while $\mathcal{T} \neq \emptyset$ do
3       Pick a job $t \in \mathcal{T}$
4       $\mathcal{T} \leftarrow \mathcal{T} \setminus \{t\}$
5       if LB$(\mathcal{I} \cup \mathcal{T}) > C$ then continue
6       $(res, S) \leftarrow$ isFeasible$(\mathcal{I} \cup \mathcal{T}, C)$
7       if $res$ then        // $t$ transition job
8           $\mathcal{N} \leftarrow$ JobRotation$(\mathcal{I}, \mathcal{T}, t, S, C)$
9           $(\mathcal{I}, \mathcal{T}) \leftarrow (\mathcal{I} \cup \mathcal{N} \cup \{t\}, \mathcal{T} \setminus \mathcal{N})$
10  return $\mathcal{I}$
```

Boosting MCSJ Extraction

The solutions computed by the decision procedure can be used to speed up the computation of minimal corrections of inconsistent systems (Nöhrer, Biere, and Egyed 2012).

Similarly as before, in the computation of an MCSJ the set of jobs $\mathcal{J}$ is implicitly partitioned as $\mathcal{J} = \{\mathcal{F}, \mathcal{C}, \mathcal{U}\}$, where $\mathcal{F}$ is a feasible subset, $\mathcal{C}$ contains jobs in the MCSJ, and $\mathcal{U}$ are jobs unknown to belong to the MCSJ or not.

After testing a feasible instance $(\mathcal{F} \cup \mathcal{U}', C)$, with $\mathcal{U}' \subseteq \mathcal{U}$, the jobs in $\mathcal{U}'$ are moved to $\mathcal{F}$. Given a schedule S to such instance, if it can be *extended* to a schedule S' that includes a job $j \in \mathcal{U} \setminus \mathcal{U}'$, then $(\mathcal{F} \cup \{j\}, C)$ would be proved feasible.

We can look for such schedule efficiently: we first obtain the sequence σ induced by S, and invoke the procedure Fits(j, σ, C), which returns a position i. If $i > 0$, we build a new sequence σ' by inserting the job j at the i-th position in σ. The left-shifted schedule $L_{\sigma'}$ proves $(\mathcal{F} \cup \{j\}, C)$ feasible without an invocation to the decision procedure. This process can be performed for all the jobs in a set $\mathcal{U}$ by iteratively extending the schedule, as shown in Algorithm 8. The algorithm returns a set of jobs that can be added to the feasible set $\mathcal{F}$. It runs in $\mathcal{O}(n^2)$, with $n = |\mathcal{F} \cup \mathcal{U}|$.

Algorithm 9 shows an enhanced version of Linear Search for computing an MCSJ. Before invoking the decision procedure on the instance $(\mathcal{F} \cup \{j\}, C)$, it discards the job j if the lower bound exceeds C (eventually, j will be included in the MCSJ). In addition, after a feasible invocation it tries to extend the computed schedule with jobs in $\mathcal{U}$.

Both optimizations can be also integrated into QuickXplain and Progression. In these cases, it suffices to keep a global set $\mathcal{F}$ with the under-approximation of the MFSJ. After a feasible call to the decision procedure, the schedule is used to extend $\mathcal{F}$ with jobs contained in the target set $\mathcal{T}$ (see Algorithms 2 and 3), which are then removed from this set.

Algorithm 8: Procedure ExtendSchedule

```
    Input: $\mathcal{F}, \mathcal{U}$: Sets, $S$: Schedule for $\mathcal{F}$, $C$: Limit
    Output: $\mathcal{N}$: Jobs to add to $\mathcal{F}$
1   $(\sigma, \mathcal{N}) \leftarrow$ (GetSequence$(S), \emptyset)$
2   foreach $j \in \mathcal{U}$ do
3       $i \leftarrow$ Fits$(j, \sigma, C)$
4       if $i > 0$ then
5           $\sigma \leftarrow$ InsertJobAtPos$(j, i, \sigma)$
6           $\mathcal{N} \leftarrow \mathcal{N} \cup \{j\}$
7   return $\mathcal{N}$
```

Algorithm 9: Enhanced LS (MCSJ extraction)

```
    Input: $\mathcal{J}$: Set of jobs, $C$: Makespan limit
    Output: MCSJ of $\mathcal{J}$
1   $(\mathcal{F}, \mathcal{U}) \leftarrow (\emptyset, \mathcal{J})$
2   while $\mathcal{U} \neq \emptyset$ do
3       Pick a job $j \in \mathcal{U}$
4       $\mathcal{U} \leftarrow \mathcal{U} \setminus \{j\}$
5       if LB$(\mathcal{F} \cup \{j\}) > C$ then continue
6       $(res, S) \leftarrow$ isFeasible$(\mathcal{F} \cup \{j\}, C)$
7       if $res$ then
8           $\mathcal{F} \leftarrow \mathcal{F} \cup \{j\}$
9           $\mathcal{N} \leftarrow$ ExtendSchedule$(\mathcal{F}, \mathcal{U}, S, C)$
10          $(\mathcal{F}, \mathcal{U}) \leftarrow (\mathcal{F} \cup \mathcal{N}, \mathcal{U} \setminus \mathcal{N})$
11  return $\mathcal{J} \setminus \mathcal{F}$
```

Experimental Results

A series of experiments was conducted to assess the performance of the proposed algorithms.

We implemented a prototype[1] in C++, interfacing the constraint programming solver IBM ILOG CP Optimizer (Laborie et al. 2018), used by the algorithms as the decision procedure. A feasibility test on $(\mathcal{J}, C)$ is modeled by using an interval variable (IloIntervalVar) for each job and enforcing a global IloNoOverlap constraint between them. In addition, the makespan is represented by an IloMax expression, enforced not to exceed C. The invocations were run using one worker and default parameters.

The experiments were performed over 600 infeasible instances, generated as follows: First, different sets of jobs $\mathcal{J}$ were built using a process described in (Carlier 1982). Given n and a value k, each job $i \in \{1, ..., n\}$ is assigned the integers $p_i \in U(1, 50)$, $r_i \in U(0, n \times k)$ and $q_i \in U(0, n \times k)$, where U denotes a uniform distribution. Five such sets of jobs were built for each pair (n, k), with $n \in \{100, 250, 500, 1000, 2500, 5000, 7500, 10000, 12500, 15000\}$ and $k \in \{1, 2, 5\}$. Then, the optimal makespan C^* was computed for each of them, and we built 5 infeasible instances $(\mathcal{J}, C)$ by setting C to a fraction

[1]Available at https://github.com/carlosmencia/erioms.git

		Avg. Time (s)		Avg. #Calls	
Method	#Sol.	All	*Easy*	Solved	*Easy*
`CR1`	219	2361.0	159.5	-	-
`CR2`	244	2222.3	131.3	-	-
`Del`	297	1955.2	65.8	1352.9	462.8
`QX`	297	2026.3	118.0	810.5	557.8
`Prog`	302	1922.2	66.2	440.4	293.7
`EDel`	**560**	**540.8**	**0.7**	**5.5**	**4.5**
`EQX`	342	1764.4	28.4	257.3	145.6
`EProg`	532	726.2	1.0	10.8	8.7

Table 1: Summary of results for MISJ extraction

$r \in \{0.1, 0.25, 0.5, 0.75, 0.9\}$ of C^*. Some small values of r led to instances containing jobs j with $st_j + p_j + q_j > C$. These were removed as, in these cases, the MISJ $\{j\}$ can be extracted in linear time. In all, there are 600 instances, with 60 of each size. Regarding r, there are 50 instances with $r = 0.1$, 100 with $r = 0.25$ and 150 with the other values.

The experiments were run on a Linux cluster (Intel Xeon 2.26 GHz), with a time limit of 1 hour for each instance.

MISJ Extraction

We analyze the results obtained by Deletion, QuickXplain and Progression in both their basic (`Del`, `QX`, `Prog`) and enhanced (`EDel`, `EQX`, `EProg`) versions. We also consider CP Optimizer's *Conflict Refiner* functionality, that identifies minimal infeasible subsets of constraints. To compute an MISJ using the model described above, we activated the option `ConflictRefinerOnVariables`. This approach is named `CR1`. An alternative, referred to as `CR2`, models the jobs as *optional activities*, enforcing their presence with an `IloPresenceOf` constraint. Such constraints included in a minimal conflict represent an MISJ.

The results are summarized in Table 1. For each method, it reports the number of instances solved by the time limit (i.e., an MISJ was computed), the average running time over all instances (taking 3600s for the unsolved ones) and over 212 *easy* instances solved by all methods. It also shows the average number of calls to the decision procedure for the instances solved by the method and for the easy ones. Moreover, Figure 3 depicts the running times of the algorithms. In this plot, for a given method, the point (x, y) indicates that x instances are solved in no more than y seconds.

The basic versions of the algorithms clearly outperform `CR1` and `CR2`. Surprisingly, these basic algorithms obtain similar results. `QX` and `Prog` reduce the number of calls (w.r.t. `Del`), but this does not translate into significant performance gains. We have observed that infeasible instances are usually solved by the decision procedure faster than feasible ones (often more than 10 times faster). In the computation of an MISJ, `QX` and `Prog` aim at reducing the number of infeasible calls, what explains these results. For example, for the easy instances, on average `Del` performs 275.2 feasible and 187.6 infeasible calls, whereas `Prog` makes 280.4 and 13.3, but infeasible calls take much shorter time. The

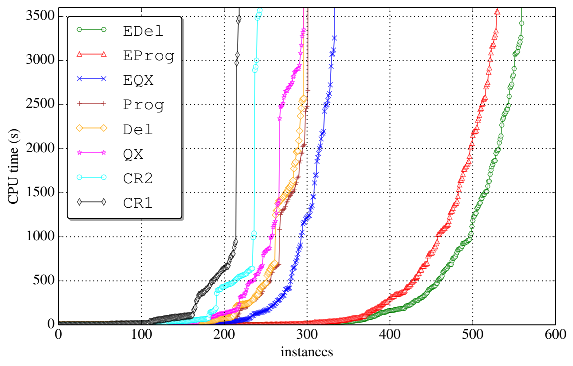

Figure 3: MISJ extraction: Running times

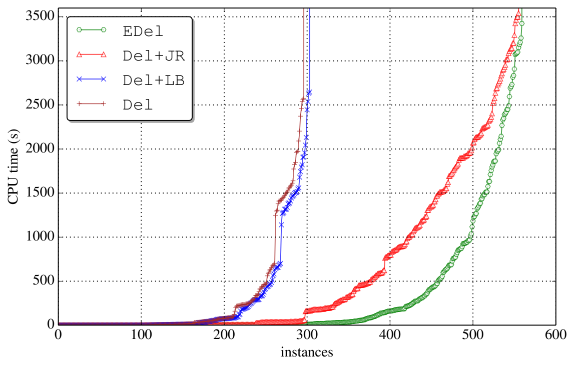

Figure 4: MISJ extraction: Versions of Deletion

enhanced algorithms perform much better. For `EProg` and `EDel` the improvement is dramatic, what allows them to solve several instances with 15000 jobs (25 and 36 respectively, while `Prog` and `Del` solve 5 and 0). `EQX` lays behind (due to the limitations of job rotation in this algorithm), but it clearly outperforms `QX`.

Figure 4 shows the effect of each optimization in Deletion. `Del+LB` only exploits lower bounds, whereas `Del+JR` only uses job rotation. The main responsible for the efficiency gains is job rotation, but combining both techniques (`EDel`) results in a remarkable improvement.

MCSJ Extraction

We evaluate the basic and enhanced versions of Linear Search (`LS` and `ELS` respectively) as well as those of QuickXplain and Progression for computing an MCSJ.

Table 2 reports a summary of the results (in this case there are 363 *easy* instances solved by all the methods). Running times are shown in Figure 5. The basic methods perform better than they did for MISJ extraction, and there are substantial differences among them. In this case, `QX` and `Prog` successfully avoid (expensive) feasible calls, what results in a clear advantage over `LS`. The proposed optimizations are very effective, enabling the algorithms to solve more instances with very few calls to the decision procedure. Remarkably, `EQX` solves *all* the instances in the benchmark set.

Figure 6 assesses the effect of each optimization in Quick-

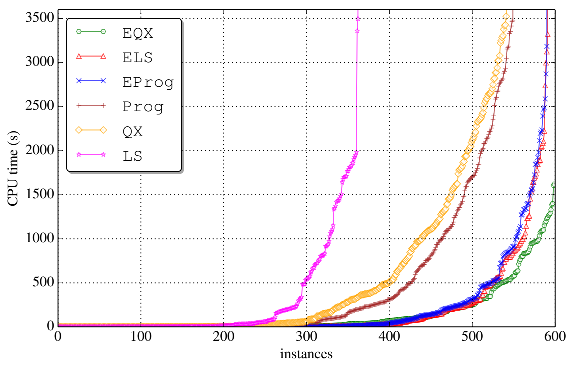

Figure 5: MCSJ extraction: Running times

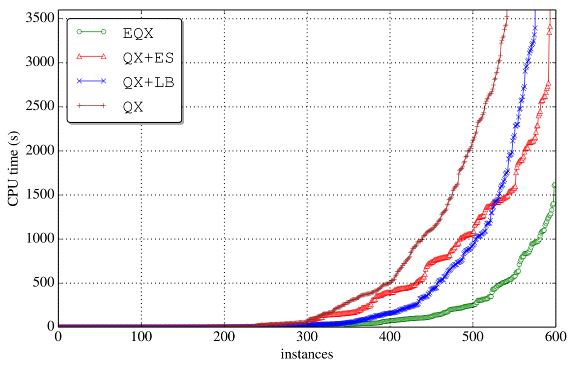

Figure 6: MCSJ extraction: Versions of QuickXplain

		Avg. Time (s)		Avg. #Calls	
Method	#Sol.	All	*Easy*	Solved	*Easy*
LS	363	1575.8	254.2	2125.0	2125.0
QX	542	785.1	49.4	4249.4	2791.7
Prog	550	678.3	24.2	2176.5	1413.9
ELS	592	225.6	**2.2**	3.1	2.6
EQX	**600**	**145.9**	6.7	**2.5**	**2.2**
EProg	591	243.7	2.4	3.3	2.7

Table 2: Summary of results for MCSJ extraction

Xplain. QX+LB only exploits lower bounds and QX+ES tries to extend schedules obtained after feasible calls. Each of the techniques alone brings significant improvements, and their combination results in the best-performing approach.

It is worth mentioning that, for both tasks, the instances tend to get harder with both size and increasing values of r. This is not surprising, as more (and potentially harder) feasibility tests may be necessary as size grows. Besides, higher values of r may result in the existence of larger MISJs and MFSJs (smaller MCSJs), what affects performance.

Related Work

In the scheduling literature, infeasible problems have attracted a considerable interest. In this respect, most works focused on scheduling as many jobs as possible under the given hard constraints, and that is related to computing an MFSJ (resp. MCSJ) of the largest (resp. smallest) size.

This task was studied in different settings. For instance, Della Croce, Gupta, and Tadei (2000) investigate the problem in the context of flow shop scheduling with a common due date. Barbulescu et al. (2004) address oversubscribed satellite scheduling problems with multiple resources by means of genetic algorithms and local search. Liao et al. (2019) use MaxSAT solvers to tackle over-loaded real-time systems with a unary resource. Recently, Mencía, Mencía, and Varela (2021) studied the problem in the context of job shop scheduling with a constraint on the makespan, and proposed a genetic algorithm that approximates MFSJs by using an incomplete version of the Linear Search algorithm. The same problem was considered in (Rodler, Teppan, and Jannach 2021), where it was solved by computing random MFSJs with a variant of QuickXplain. This work identified the relationship between MFSJs and MSMP, and proposed a reduction similar to the one we showed for one machine scheduling. This approach invokes a solver as a black-box, without using optimizations.

Our work is also related to the growing area of explainability in scheduling. In this respect, the closest work we can mention is the framework proposed by Lauffer and Topku (2019) for explaining infeasible RCPSP-like problems. Explanations with different levels of detail are computed by enumerating MUSes of an SMT encoding. One of such explanations is defined in terms of the tasks in the problem, what is related to the notion of MISJs studied herein. Other works include the tool developed by Agrawal, Yelamanchili, and Chien (2020) for explaining why certain activities were not scheduled in the context of NASA's Mars Rover missions, or the use of abstract argumentation to explain why a schedule is feasible or efficient (Cyras, Lee, and Letsios 2021). Besides, Pozanco et al. (2022) use MILP to explain why a user preference was not met in an optimal schedule. Korikov and Beck (2021) proposed a framework based on inverse constraint programming for computing counterfactual explanations about optimal solutions, and used this approach in a one machine problem with due dates. Such explanations share some similarities with MCSJs. Investigating this relationship seems interesting for the future.

Conclusions

This paper studies the tasks of explaining and correcting infeasible one machine scheduling problems with a limit on the makespan. We have shown that the analysis of such infeasible problems can be reduced to problems of finding minimal subsets over monotone predicates. In turn, this enables a large number of well-known algorithms to be used for this purpose. Furthermore, we have identified a number of optimizations which can be exploited with any of these algorithms. The experimental results demonstrate that the proposed optimizations, integrated on several algorithms for reasoning about infeasibility, yield critical performance gains in practice, resulting in a large number of problem instances solved within the allowed running times.

Acknowledgments

This work is supported by the Spanish Government under grant PID2019-106263RB-I00 and by the AI Interdisciplinary Institute ANITI, funded by the French program "Investing for the Future – PIA3" under Grant agreement n^o ANR-19-PI3A-0004. This research started during a stay R.M. held in ANITI, funded by University of Oviedo's Campus de Excelencia Internacional / Banco de Santander.

References

Adams, J.; Balas, E.; and Zawack, D. 1988. The Shifting Bottleneck Procedure for Job Shop Scheduling. *Management Science*, 34(3): 391–401.

Agrawal, J.; Yelamanchili, A.; and Chien, S. A. 2020. Using Explainable Scheduling for the Mars 2020 Rover Mission. *CoRR*, abs/2011.08733.

Bailey, J.; and Stuckey, P. J. 2005. Discovery of Minimal Unsatisfiable Subsets of Constraints Using Hitting Set Dualization. In *PADL*, 174–186.

Bakker, R. R.; Dikker, F.; Tempelman, F.; and Wognum, P. M. 1993. Diagnosing and Solving Over-Determined Constraint Satisfaction Problems. In *IJCAI*, 276–281.

Barbulescu, L.; Howe, A. E.; Whitley, L. D.; and Roberts, M. 2004. Trading Places: How to Schedule More in a Multi-Resource Oversubscribed Scheduling Problem. In *ICAPS*, 227–234.

Belov, A.; Lynce, I.; and Marques-Silva, J. 2012. Towards efficient MUS extraction. *AI Commun.*, 25(2): 97–116.

Bendík, J.; and Cerná, I. 2020. Replication-Guided Enumeration of Minimal Unsatisfiable Subsets. In *CP*, 37–54.

Birnbaum, E.; and Lozinskii, E. L. 2003. Consistent subsets of inconsistent systems: structure and behaviour. *J. Exp. Theor. Artif. Intell.*, 15(1): 25–46.

Brucker, P.; Jurisch, B.; and Sievers, B. 1994. A Branch and Bound Algorithm for the Job-Shop Scheduling Problem. *Discret. Appl. Math.*, 49(1-3): 107–127.

Brucker, P.; and Knust, S. 2006. *Complex Scheduling*. Springer.

Carlier, J. 1982. The one-machine sequencing problem. *European Journal of Operational Research*, 11(1): 42–47.

Chinneck, J. W.; and Dravnieks, E. W. 1991. Locating Minimal Infeasible Constraint Sets in Linear Programs. *ORSA Journal on Computing*, 3(2): 157–168.

Cyras, K.; Lee, M.; and Letsios, D. 2021. Schedule Explainer: An Argumentation-Supported Tool for Interactive Explanations in Makespan Scheduling. In *EXTRAAMAS*, 243–259.

Della Croce, F.; Gupta, J. N.; and Tadei, R. 2000. Minimizing tardy jobs in a flowshop with common due date. *European Journal of Operational Research*, 120(2): 375 – 381.

Garey, M.; and Johnson, D. 1979. *Computers and Intractability*. Freeman.

Gupta, S. D.; Genc, B.; and O'Sullivan, B. 2021. Explanation in Constraint Satisfaction: A Survey. In *IJCAI*, 4400–4407.

Junker, U. 2004. QUICKXPLAIN: Preferred Explanations and Relaxations for Over-Constrained Problems. In *AAAI*, 167–172.

Korikov, A.; and Beck, J. C. 2021. Counterfactual Explanations via Inverse Constraint Programming. In Michel, L. D., ed., *CP*, volume 210 of *LIPIcs*, 35:1–35:16.

Laborie, P. 2003. Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artif. Intell.*, 143(2): 151–188.

Laborie, P. 2014. An Optimal Iterative Algorithm for Extracting MUCs in a Black-box Constraint Network. In *ECAI*, 1051–1052.

Laborie, P.; Rogerie, J.; Shaw, P.; and Vilím, P. 2018. IBM ILOG CP optimizer for scheduling - 20+ years of scheduling with constraints at IBM/ILOG. *Constraints*, 23(2): 210–250.

Lauffer, N.; and Topku, U. 2019. Human-understandable explanations of infeasibility for resource-constrained scheduling problems. In *Proc. 2nd Workshop on Explainable AI Planning*, 44–52.

Liao, X.; Zhang, H.; Koshimura, M.; Huang, R.; and Yu, W. 2019. Maximum Satisfiability Formulation for Optimal Scheduling in Overloaded Real-Time Systems. In *PRICAI*, 618–631.

Liffiton, M. H.; Previti, A.; Malik, A.; and Marques-Silva, J. 2016. Fast, flexible MUS enumeration. *Constraints*, 21(2): 223–250.

Liffiton, M. H.; and Sakallah, K. A. 2008. Algorithms for Computing Minimal Unsatisfiable Subsets of Constraints. *J. Autom. Reasoning*, 40(1): 1–33.

Marques-Silva, J.; Janota, M.; and Belov, A. 2013. Minimal Sets over Monotone Predicates in Boolean Formulae. In *CAV*, 592–607.

Marques-Silva, J.; Janota, M.; and Mencía, C. 2017. Minimal sets on propositional formulae. Problems and reductions. *Artif. Intell.*, 252: 22–50.

Marques-Silva, J.; and Mencía, C. 2020. Reasoning About Inconsistent Formulas. In *IJCAI*, 4899–4906.

Mencía, R.; Mencía, C.; and Varela, R. 2021. Efficient repairs of infeasible job shop problems by evolutionary algorithms. *Eng. Appl. Artif. Intell.*, 104: 104368.

Narodytska, N.; Bjørner, N.; Marinescu, M. V.; and Sagiv, M. 2018. Core-Guided Minimal Correction Set and Core Enumeration. In *IJCAI*, 1353–1361.

Nöhrer, A.; Biere, A.; and Egyed, A. 2012. Managing SAT inconsistencies with HUMUS. In *VaMoS*, 83–91.

Pozanco, A.; Mosca, F.; Zehtabi, P.; Magazzeni, D.; and Kraus, S. 2022. Explaining Preference-Driven Schedules: The EXPRES Framework. In *ICAPS*, 710–718.

Reiter, R. 1987. A Theory of Diagnosis from First Principles. *Artif. Intell.*, 32(1): 57–95.

Rodler, P.; Teppan, E.; and Jannach, D. 2021. Randomized Problem-Relaxation Solving for Over-Constrained Schedules. In *KR*, 696–701.

Wieringa, S. 2012. Understanding, Improving and Parallelizing MUS Finding Using Model Rotation. In *CP*, 672–687.