Argumentation for Explainable Scheduling

Kristijonas Čyras and **Dimitrios Letsios** and **Ruth Misener** and **Francesca Toni**
Imperial College London, London, UK

Abstract

Mathematical optimization offers highly-effective tools for finding solutions for problems with well-defined goals, notably scheduling. However, optimization solvers are often unexplainable black boxes whose solutions are inaccessible to users and which users cannot interact with. We define a novel paradigm using argumentation to empower the interaction between optimization solvers and users, supported by tractable explanations which certify or refute solutions. A solution can be from a solver or of interest to a user (in the context of 'what-if' scenarios). Specifically, we define argumentative and natural language explanations for why a schedule is (not) feasible, (not) efficient or (not) satisfying fixed user decisions, based on models of the fundamental makespan scheduling problem in terms of abstract argumentation frameworks (AFs). We define three types of AFs, whose stable extensions are in one-to-one correspondence with schedules that are feasible, efficient and satisfying fixed decisions, respectively. We extract the argumentative explanations from these AFs and the natural language explanations from the argumentative ones.

1 Introduction

Computational optimization empowers effective decision making. Given a mathematical optimization model with well-defined numerical variables, objective function(s), and constraints, a solver generates an efficient and ideally optimal solution. If the model and solver are correct, then implementing the optimal solution can have major benefits. But how can we explain the optimal solutions to a user? Currently, solvers express necessary and sufficient optimality conditions with formal mathematics, so users often consider the optimization solver as an unexplainable black box.

Explainable scheduling is a critical application (Sacchi et al. 2015) and our test bed for *explainable optimization*. Consider the fundamental makespan scheduling problem, a discrete optimization problem for effective resource allocation (Graham 1969). This problem arises in for example nurse rostering where staff of different skill qualification categories, e.g. Registered Nurse, Nurse's Aide, need to be assigned to shifts (Warner and Prawda 1972). In the planning period, staff are scheduled, e.g. for the next 4 weeks (Burke et al. 2004). But nursing personnel, hospital managers, or patients may inquire about the fairness or optimality of the schedule and possible changes. Further, when unexpected events occur, e.g. staff illness or an unusually high influx of patients, feasible schedules must be recovered (Moz and Vaz Pato 2007). We take the first steps towards enabling users to interact with, and obtain explanations from, optimal scheduling in general and nurse rostering in particular.

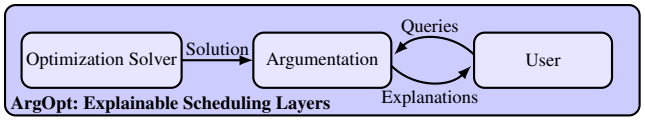


Figure 1: ArgOpt produces explanations about solutions generated by an optimization solver to the fundamental makespan scheduling problem. *Argumentation* is an intermediate layer between the *optimization solver* and the *user*. The optimization solver passes the computed solution to the argumentation layer. The user interacts with the argumentation layer to obtain argumentative and natural language explanations.

This paper proposes a novel paradigm (that we call *ArgOpt*) for explaining why a solution is (not) good by leveraging abstract argumentation (AA) as an intermediate layer between the user and optimization software. Argumentation is highly suitable for explaining reasoning and decisions (Moulin et al. 2002; Atkinson et al. 2017) with argumentative explanations proposed in various settings, see e.g. (García et al. 2013; Fan and Toni 2015; Čyras et al. 2018; Zeng et al. 2018). We show how AA offers computational optimization an accessible knowledge representation tool, namely AA frameworks (AFs), for modeling optimization problems. These AFs are constructed automatically given a scheduling problem instance and possibly fixed user decisions, allow formal explanation definitions and enable efficient generation of natural language explanations. Figure 1 illustrates ArgOpt.

What makes an optimization solution good? A good solution should (i) be feasible, (ii) be efficient (ideally optimal), and (iii) satisfy fixed (user) decisions. ArgOpt introduces a toolkit realizing these needs and dealing with a number of the relevant challenges. A good explanation should be efficiently attainable, combine few causal relationships and admit simple natural language interpretations. To build trust, explanations should be associated with a formal representation providing interpretable certificates on why the explanation is valid and how it is generated. For tractability, we implement polynomial explainability and thereby achieve both

computational tractability, i.e. quick generation of results, and cognitive tractability, i.e. clear user explanations. Our explanation-generating engine has a modular structure for generating different types of explanations.

Given a problem instance, we construct AFs to explain problem instance solutions. We map decisions (schedule assignments) to arguments and capture feasibility and optimality conditions via attacks. We then extract from AFs argumentative explanations pertaining to the decisions and the related conditions, which can in turn be rendered in natural language. ArgOpt comprising an optimization solver and an argumentation layer can thereby justify its solutions against human-proposed solutions and effectively perform human-AI interaction for efficient decision making. We may derive explanations on potential infeasibilities and weak solutions. Overall, explainability enables the decision maker to (i) check feasibility of possible solutions, (ii) perform what-if analysis for scenarios, and (iii) recover feasible solutions after various disturbances.

2 Background

2.1 Makespan Scheduling

An instance I of the *makespan scheduling problem*, e.g. (Graham 1969; Leung 2004; Brucker 2007), is a pair $(\mathcal{M},\mathcal{J})$, where $\mathcal{J}=\{1,\ldots,n\}$ is a set of *n independent* jobs with a vector $\vec{p}=\{p_1,\ldots,p_n\}$ of processing times which have to be executed by a set $\mathcal{M}=\{1,\ldots,m\}$ of m parallel identical machines. Job $j\in\mathcal{J}$ must be processed by exactly one machine $i\in\mathcal{M}$ for p_j units of time non-preemptively, i.e. in a single continuous interval without interruptions. Each machine may process at most one job per time. The objective is to find a minimum makespan schedule, i.e. to minimize the last machine completion time. In the nurse rostering example, each task (or job) needs to be assigned a specific nurse (or machine). In this simple setting, each task is assigned to just one nurse and we aim for all staff to complete as soon as possible.

In a standard mixed integer linear programming formulation, binary decision variable $x_{i,j}$ is 1 if job $j\in\mathcal{J}$ is executed by machine $i\in\mathcal{M}$ and 0 otherwise. Thus, a schedule of $(\mathcal{M},\mathcal{J})$ can be seen as an $m\times n$ matrix $S\in\{0,1\}^{m\times n}$ with entries $x_{i,j}\in\{0,1\}$ representing job assignments to machines, for $i\in\mathcal{M}$ and $j\in\mathcal{J}$.

Given a schedule S, let C_i be the completion time of machine $i\in\mathcal{M}$ in S and let $C_{\max}=\max_{1\leqslant i\leqslant m}\{C_i\}$ be the makespan. The problem is formally described by Equations (1a)–(1e) (next column). Expression (1a) minimizes makespan. Constraints (1b) are the makespan definition. Constraints (1c) allow a machine to execute at most one job per time. Constraints (1d) assign each job to exactly one machine. An optimal schedule satisfies all Constraints (1b)–(1d) and minimizes makespan objective Expression (1a).

$$\min_{C_{\max},C_i,x_{i,j}} \quad C_{\max} \tag{1a}$$

$$C_{\max}\geqslant C_i \qquad i\in\mathcal{M} \tag{1b}$$

$$C_i=\sum_{j=1}^{n}x_{i,j}\cdot p_j \qquad i\in\mathcal{M} \tag{1c}$$

$$\sum_{i=1}^{m}x_{i,j}=1 \qquad j\in\mathcal{J} \tag{1d}$$

$$x_{i,j}\in\{0,1\} \qquad j\in\mathcal{J},i\in\mathcal{M} \tag{1e}$$

In the nurse rostering example, this formulation allows to deal with skill qualifications, e.g. to limit tasks assigned to nurses. More elaborate nurse rostering incorporates contractual obligations, e.g. assigning a nurse the correct number of shifts per week, and allows holiday, e.g. avoiding jobs for a nurse in a given week.

This paper assumes an instance $I=(\mathcal{M},\mathcal{J})$ of a makespan scheduling problem with $\mathcal{M}=\{1,\ldots,m\}$ and $\mathcal{J}=\{1,\ldots,n\}$, for $m,n\geqslant 1$, unless stated otherwise.

2.2 Abstract Argumentation (AA)

We give essential background on Abstract Argumentation (AA) following its original definition in (Dung 1995).

An *AA framework* (*AF*) is a directed graph $(Args,\rightsquigarrow)$ with
- a set *Args* of *arguments*, and
- a binary *attack* relation $\rightsquigarrow$ over *Args*.

For $\mathsf{a},\mathsf{b}\in Args$, $\mathsf{a}\rightsquigarrow\mathsf{b}$ means that a attacks b, and $\mathsf{a}\not\rightsquigarrow\mathsf{b}$ means that a does not attack b. With an abuse of notation, we extend the attack notation to sets of arguments as follows. For $A\subseteq Args$ and $\mathsf{b}\in Args$:
- $A\rightsquigarrow\mathsf{b}$ iff $\exists\mathsf{a}\in A$ with $\mathsf{a}\rightsquigarrow\mathsf{b}$;
- $\mathsf{b}\rightsquigarrow A$ iff $\exists\mathsf{a}\in A$ with $\mathsf{b}\rightsquigarrow\mathsf{a}$.

A set $E\subseteq Args$ of arguments, also called an *extension*, is
- *conflict-free* iff $E\not\rightsquigarrow E$;
- *stable* iff E is conflict-free and $\forall\mathsf{b}\in Args\setminus E$, $E\rightsquigarrow\mathsf{b}$.

While finding stable extensions of a given AF is NP-hard, verifying if a given extension is stable is polynomial (in the number of arguments) (Dunne and Wooldridge 2009).

3 Setting the Ground for ArgOpt

Within the makespan scheduling problem, we identify three dimensions, namely schedule **feasibility**, schedule **efficiency** and accommodating **fixed user decisions** within schedules, formally defined below. Our novel paradigm ArgOpt focuses on explaining why a given schedule:
- is **feasible** or not,
- is **efficient** or not,
- **satisfies fixed user decisions** or not.

Feasibility simply amounts to dropping the makespan minimization objective:

Definition 3.1. A schedule is *feasible* iff it meets constraints (1b) – (1d).

It can be shown that this notion of feasibility amounts to assigning each job to exactly one machine:

Lemma 3.1. *A schedule is feasible iff it meets constraint (1d), i.e.* $\sum_{i=1}^{m}x_{i,j}=1\ \forall j\in\mathcal{J}$.

(Full version of the paper with proofs can be found at http://arxiv.org/abs/1811.05437.)

Feasibility is polynomial, whereas finding optimal solutions for the makespan scheduling problem is strongly NP-hard (Garey and Johnson 1979). A standard, less drastic approach in optimization to deal with intractability is approximation algorithms, e.g. the common longest processing time first heuristic (Graham 1969) produces a 4/3-approximate schedule, namely attaining makespan a constant factor far from the optimal. In this vein, we define efficiency as feasibility and satisfaction of the single and pairwise exchange properties that guarantee approximately optimal schedules.

Definition 3.2. Schedule S satisfies the *single* and *pairwise exchange properties* iff for every *critical* job $j \in \mathcal{J}$ such that $x_{i,j} = 1$ and $C_i = C_{\max}$, it respectively holds that, for $i' \neq i$,

- *Single Exchange Property* (*SEP*): $C_i - C_{i'} \leqslant p_j$;
- *Pairwise Exchange Property* (*PEP*): for any job $j' \neq j$ with $x_{i',j'} = 1$, if $p_j > p_{j'}$, then $C_i + p_{j'} \leqslant C_{i'} + p_j$.

S is *efficient* iff S is feasible and satisfies both SEP and PEP.

SEP concerns improving a schedule by a single exchange of any critical job between machines. PEP concerns pairwise exchanges of critical jobs with other jobs on other machines. SEP and PEP are necessary optimality conditions (but not sufficient; see e.g. the list-scheduling algorithm tightness analysis in (Williamson and Shmoys 2011)).

Lemma 3.2. *Every optimal schedule satisfies SEP and PEP.*

The overall setting for explanation is as follows. An optimization solver recommends an optimal schedule S^* for the given instance I of the makespan scheduling problem. The user inquires whether another schedule S could be used instead. S expresses changes to S^* within 'what-if' scenarios (e.g. 'what if this job were assigned to that machine instead'?) If S is also optimal, then the answer is positive and a certifiable explanation as to why this is so may be provided. Else, if S is (provably) not optimal, then the answer is negative and an explanation is generated as to why.

In addition, we envisage that a user may fix some decisions, i.e. (non-)assignments of jobs to machines originally unbeknownst to the scheduler, e.g. that a nurse is absent or lacks necessary skills for a task, or that a nurse volunteers for a task, and may want to find out whether and why S satisfies or violates these decisions.

Definition 3.3. Let

- *negative fixed decisions* be $D^- = \mathcal{M}^- \times \mathcal{J}^- \subseteq \mathcal{M} \times \mathcal{J}$;
- *positive fixed decisions* be $D^+ = \mathcal{M}^+ \times \mathcal{J}^+ \subseteq \mathcal{M} \times \mathcal{J}$;
- *fixed decisions* be $D = (D^-, D^+)$ such that $D^- \cap D^+ = \emptyset$ and $\nexists (i,j),(k,j) \in D^+$ with $i \neq k$.

We say that schedule S satisfies

- D^- iff $(i,j) \in D^-$ implies $x_{i,j} = 0$;
- D^+ iff $(i,j) \in D^+$ implies $x_{i,j} = 1$;
- $D = (D^-, D^+)$ iff S satisfies both D^- and D^+.

S violates D^-, D^+, D iff it does not satisfy D^-, D^+, D, resp.

Negative (resp. positive) fixed decisions insist on which jobs cannot (resp. must) be done on which machines. Fixed decisions consist of compatible negative and positive fixed decisions, where the positive fixed decisions, if any, are feasible in that no two machines are asked to do the same job.

Note that capturing fixed decisions allows us to capture various phenomena, such as (in the running example):

- nurse i falling ill, with $D_i^- = \{(i,j) \,:\, j \in \mathcal{J}\}$;
- cancelled job j, with $D_j^- = \{(i,j) \,:\, i \in \mathcal{M}\}$.

These may be particularly useful in a dynamic setting, where fixed decisions need to be taken into account after having executed some part of the schedule.

In summary, feasibility is a basic constraint; efficiency concerns schedule-specific optimality conditions; fixed user decisions pertain to schedule-specific feasibility while disregarding optimality.

Our paradigm ArgOpt is driven by the following desiderata (where 'good' stands for any amongst feasible, efficient or satisfying fixed decisions):

- **soundness and completeness** of explanations, in the sense that, given schedule S, there exists an explanation as to why S is (not) 'good' iff S is (not) 'good';
- **computational tractability**, in the sense that explaining whether and why schedule S is (not) 'good' can be performed in polynomial time in the size of the makespan scheduling problem instance I;
- **cognitive tractability**, in the sense that each explanation pertaining to schedule S and presented to the user should be polynomial in the size of S.

Tractability is crucial to ensure that explaining results in a low construction overhead, an essential property of explanations (Sørmo, Cassens, and Aamodt 2005). We have restricted attention to feasible and efficient (rather than optimal) solutions because answering and explaining why an arbitrary schedule is optimal in polynomial time is ruled out due to NP-hardness of makespan scheduling, unless P=NP.

We choose *argumentation* as the underlying technology for ArgOpt as it serves us well to fulfil the above desiderata:

- argumentation affords knowledge representation tools, such as AFs, for providing sound and complete counterparts for a diverse range of problems, e.g. games (Dung 1995), and it has long been identified as suitable for explaining, see e.g. (Moulin et al. 2002; Atkinson et al. 2017); we define counterparts for determining 'good' schedules (see Section 4) and build upon these for defining sound and complete explanations (see Section 5);
- argumentation enables tractable specifications of the optimization problems we consider and tractable generation of explanations (see Section 5);
- cognitively tractable explanations can be extracted from AFs, acting as certificates as to why a schedule is (not) 'good' (see Section 5).

4 Argumentation for Optimization

We approach the issue of explaining, using AA, whether and why the solutions to the makespan scheduling problem are 'good' in three steps, as illustrated in Figure 2.

First, we capture schedule feasibility in the sense of mapping an instance of the makespan scheduling problem into an AF by identifying assignments with arguments, so that the feasible schedules are in one-to-one correspondence with the stable extensions. We then capture optimality conditions in the sense of mapping the properties of a given schedule

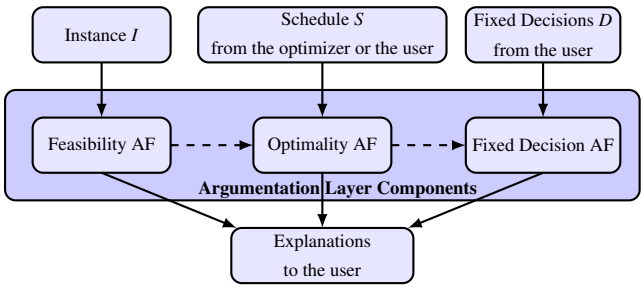


Figure 2: Argumentation layer components in ArgOpt: (i) the feasibility AF $(Args_F, \rightsquigarrow_F)$ takes an instance I and explains whether a given schedule for I is feasible; (ii) the optimality AF $(Args_S, \rightsquigarrow_S)$ takes the instance I represented via $(Args_F, \rightsquigarrow_F)$ and a schedule S for I, and explains whether S is efficient; (iii) the fixed decision AF $(Args_D, \rightsquigarrow_D)$ takes either $(Args_F, \rightsquigarrow_F)$ with a schedule or $(Args_S, \rightsquigarrow_S)$, some fixed decisions D and explains whether the schedule satisfies these decisions.

into a schedule-specific AF by modifying the attack relation, so that the schedule satisfying optimality conditions equates to the corresponding extension being stable. Lastly, we capture fixed decisions for a specific schedule in the absence of optimality considerations, in the sense of mapping the fixed decisions into an AF by modifying the attacks, so that the schedule satisfying fixed decisions equates to the corresponding extension being stable.

The design of the proposed AFs incorporates tractability. First, the mappings from the makespan scheduling problem to AA are polynomial. Second, explaining whether and why a given schedule is feasible and/or efficient (optimality) and/or satisfies fixed user decisions amounts to *verifying* if the corresponding stable extension is stable in the relevant AF; this problem is also polynomial.

We chose stable extensions as the underlying semantics for the following reasons. 1. The makespan scheduling problem requires all jobs to be assigned, so we need a "global" semantics (Dung 1995) and stable semantics is one such. 2. Verification of stable extensions is polynomial, allowing us to meet the computational tractability desideratum. 3. Other semantics are either non-global (e.g. the grounded extension) or have non-polynomial verification (e.g. *coNP*-complete for preferred extensions).

4.1 Feasibility

We model the feasibility space of makespan scheduling in AA to be able to explain why a user's proposed schedule is or not feasible. We do this by mapping assignment variables (binary decisions) to arguments and capturing feasibility constraints via attacks. Specifically, argument $\mathsf{a}_{i,j}$ stands for an assignment of job $j \in \mathcal{J}$ to machine $i \in \mathcal{M}$. Attacks among arguments model pairwise incompatible decisions: $\mathsf{a}_{i,j}$ and $\mathsf{a}_{k,l}$ attacking each other models the incompatibility of assignments $\mathsf{a}_{i,j}$ and $\mathsf{a}_{k,l}$. Intuitively, the attack relation encodes different machines competing for the same job. Formally:

Definition 4.1. The *feasibility AF* $(Args_F, \rightsquigarrow_F)$ is given by
- $Args_F = \{\mathsf{a}_{i,j} \,:\, i \in \mathcal{M}, j \in \mathcal{J}\}$,
- $\mathsf{a}_{i,j} \rightsquigarrow_F \mathsf{a}_{k,l}$ iff $i \neq k$ and $j = l$.

The following definition formalizes a natural bijective mapping between schedules and extensions.

Definition 4.2. Let $(Args_F, \rightsquigarrow_F)$ be the feasibility AF. A schedule S and an extension $E \subseteq Args_F$ are *corresponding*, denoted $S \approx E$, when the following invariant holds:

$$x_{i,j} = 1 \text{ iff } \mathsf{a}_{i,j} \in E.$$

With this correspondence, the feasibility AF encodes exactly the feasibility of the makespan scheduling problem, in that feasible schedules coincide with stable extensions:

Theorem 4.1. *Let $(Args_F, \rightsquigarrow_F)$ be the feasibility AF. For any $S \approx E$, S is feasible iff E is stable.*

Proof. Let E be a stable extension of $(Args_F, \rightsquigarrow_F)$. To prove that the corresponding schedule S is feasible, we need to show that Equation 1d holds: $\sum_{i=1}^{m} x_{i,j} = 1$ for any $j \in \mathcal{J}$.

As $x_{i,j} \in \{0,1\}\ \forall i,j$, we have $\sum_{i=1}^{m} x_{i,j} \in \mathbb{N} \cup \{0\}$. Suppose for a contradiction that for some $j \in \mathcal{J}$ we have $\sum_{i=1}^{m} x_{i,j} \neq 1$.

a) First assume $\sum_{i=1}^{m} x_{i,j} > 1$. Then $\mathsf{a}_{i,j}, \mathsf{a}_{k,j} \in E$ for some $i \neq k$. But then $\mathsf{a}_{i,j} \rightsquigarrow_F \mathsf{a}_{k,j}$, whence E is not conflict-free. This contradicts E being stable.

b) Now assume $\sum_{i=1}^{m} x_{i,j} = 0$. Then $\mathsf{a}_{i,j} \notin E\ \forall i \in \mathcal{M}$. By definition of $\rightsquigarrow_F$, we then have $E \not\rightsquigarrow_F \mathsf{a}_{i,j}$ for any $i \in \mathcal{M}$. In particular, $E \not\rightsquigarrow_F \mathsf{a}_{1,j}$. This contradicts E being stable.

We next prove that if S is a feasible schedule, then the corresponding extension E is stable in $(Args_F, \rightsquigarrow_F)$.

We have $\sum_{i=1}^{m} x_{i,j} = 1$ for any $j \in \mathcal{J}$ because S is feasible. This means that for every $j \in \mathcal{J}$, E contains one and only one $\mathsf{a}_{i,j}$ for some $i \in \mathcal{M}$. Thus, by definition of $\rightsquigarrow_F$, E is conflict-free. Moreover, for any $j \in \mathcal{J}$, $\mathsf{a}_{i,j} \in E$ attacks every other $\mathsf{a}_{k,j}$ with $k \neq i$. Hence, E is stable in $(Args_F, \rightsquigarrow_F)$. □

Example 4.3. Let $\mathcal{M} = \{1,2\}$, $\mathcal{J} = \{1,2\}$, e.g. 2 nurses for 2 tasks. Figure 3a) depicts the feasibility AF $(Args_F, \rightsquigarrow_F)$. It has 4 stable extensions: $\{\mathsf{a}_{1,1}, \mathsf{a}_{1,2}\}$, $\{\mathsf{a}_{1,1}, \mathsf{a}_{2,2}\}$, $\{\mathsf{a}_{2,1}, \mathsf{a}_{1,2}\}$, $\{\mathsf{a}_{2,1}, \mathsf{a}_{2,2}\}$. They correspond to the 4 feasible $(\mathcal{M}, \mathcal{J})$ rostering schedules where each task is completed by 1 nurse.

Finally, note that constructing the feasibility AF, as well as verifying if a given schedule is feasible, is polynomial:

Lemma 4.2. *The feasibility AF $(Args_F, \rightsquigarrow_F)$ can be constructed in $O(nm^2)$ time. Verifying whether an extension $E \subseteq Args_F$ is stable can be done in $O(n^2m^2)$ time.*

4.2 Optimality Conditions

We model optimality conditions in AA to be able to explain why a user's proposed schedule is or not efficient. Lemma 3.2 implies that if a feasible schedule S can be improved by making a single exchange, i.e. S violates SEP, then S is not optimal. Likewise, Lemma 3.2 implies that if a feasible schedule S can be improved by making a pairwise exchange, i.e. S violates PEP, then S is not optimal.

We model both SEP and PEP in a single schedule-specific AF by modifying the feasibility AF as follows. Given S, we know $C_{\max}$, and so all the *critical machines* i (such that $C_i = C_{\max}$) and all the associated *critical jobs* j (such that $x_{i,j} = 1$). Then, for any (critical) pair (i,j) and any other machine $i' \neq i$, if $C_i > C_{i'} + p_j$, then S violates SEP and can

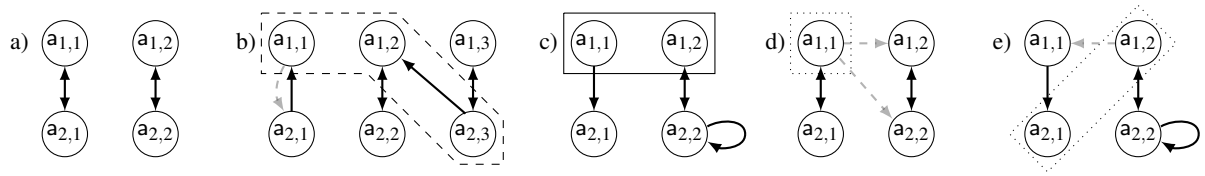


Figure 3: In all graphs depicting AFs, nodes hold arguments and edges hold attacks and (dashed) non-attacks. a) Feasibility AF of Example 4.3. b) Optimality AF of Example 4.5; (dashed) box highlights the extension (and the corresponding schedule) in question; in Example 5.7, the non-attack (dashed) explains why the schedule is not near-optimal, particularly violates SEP. c) Fixed decision AF of Example 4.7; the box indicates the unique stable extension. d) Feasibility AF with non-attacks (dashed) explaining why the schedule of Example 5.6 (the corresponding extension of which is highlighted in the (dotted) box) is not feasible. e) Fixed decision AF with the non-attack (dashed) explaining why the schedule of Example 5.8 (the corresponding extension of which is highlighted in the (dotted) box) violates fixed decisions.

be improved by making the single exchange of moving job j from machine i to machine i'. We model this by *removing* the attack $\mathsf{a}_{i,j} \rightsquigarrow_F \mathsf{a}_{i',j}$ from $(Args_F, \rightsquigarrow_F)$, and this represents that machine i should not be competing for job j with machine i'. Similarly, for any (critical) pair (i,j) and any other machine $i' \neq i$ assigned some other job $j' \neq j$ with $p_j > p_{j'}$, if $C_i + p_{j'} > C_{i'} + p_j$, then S violates PEP and can be improved by a pairwise exchange of j and j' from i to i'. We model this by *adding* an attack from $\mathsf{a}_{i',j'}$ to $\mathsf{a}_{i,j}$ in $(Args_F, \rightsquigarrow_F)$, and this represents that assigning j' to i' conflicts with assigning j to i because the latter is less efficient.

Definition 4.4. Let $(Args_F, \rightsquigarrow_F)$ be the feasibility AF and S a schedule. The *optimality AF* $(Args_S, \rightsquigarrow_S)$ is given by

- $Args_S = Args_F$,
- $\rightsquigarrow_S = \big(\rightsquigarrow_F \setminus \{(\mathsf{a}_{i,j}, \mathsf{a}_{i',j}) \,:\, C_i = C_{\max}, x_{i,j} = 1, C_i > C_{i'} + p_j\}\big) \cup \{(\mathsf{a}_{i',j'}, \mathsf{a}_{i,j}) \,:\, C_i = C_{\max}, x_{i,j} = 1, x_{i',j'} = 1, i' \neq i, j' \neq j, p_j > p_{j'}, C_i + p_{j'} > C_{i'} + p_j\}$.

We say that $(Args_F, \rightsquigarrow_F)$ is *underlying* $(Args_S, \rightsquigarrow_S)$.

To determine whether the user's proposed schedule is efficient we just need to check if the corresponding extension is stable in the optimality AF:

Theorem 4.3. *Let $(Args_F, \rightsquigarrow_F)$ be the feasibility AF, S a schedule and $S \approx E$. Let $(Args_S, \rightsquigarrow_S)$ be the optimality AF. Then E is stable in $(Args_S, \rightsquigarrow_S)$ iff S is feasible and satisfies both SEP and PEP.*

Proof. Let E be stable in $(Args_S, \rightsquigarrow_S)$. Then E is conflict-free in $(Args_F, \rightsquigarrow_F)$, because the attacks removed to capture SEP only make asymmetric attacks symmetric, and the attacks added to capture PEP are among arguments not attacking each other in $(Args_F, \rightsquigarrow_F)$. For the same reason E is also stable in $(Args_F, \rightsquigarrow_F)$. Hence, S is feasible, by Theorem 4.1. Moreover, as E is stable, $\forall j \in \mathcal{J}$ we find $\mathsf{a}_{i,j} \in E$ with $\mathsf{a}_{i,j} \rightsquigarrow_S \mathsf{a}_{i',j}$ $\forall i' \neq i$. This means no attacks were removed from $\rightsquigarrow_F$ to obtain $\rightsquigarrow_S$, so, in particular, $C_i = C_{\max}$, $x_{i,j} = 1$, $C_i > C_{i'} + p_j$ cannot hold for any (i, i', j). Hence, S satisfies SEP. Similarly, as E is conflict-free, we have that $C_i = C_{\max}, x_{i,j} = 1, x_{i',j'} = 1, i' \neq i, j' \neq j, p_j > p_{j'}$, $C_i + p_{j'} > C_{i'} + p_j$ cannot all hold for any (i, i', j, j'). Hence, S satisfies PEP.

If S is feasible, then E is stable in $(Args_F, \rightsquigarrow_F)$. If S satisfies SEP and PEP, then $\rightsquigarrow_S = \rightsquigarrow_F$. So if S is feasible and satisfies SEP and PEP, then E is stable in $(Args_S, \rightsquigarrow_S)$ too. □

Example 4.5. Let $\mathcal{M} = \{1,2\}$, $\mathcal{J} = \{1,2,3\}$ and $p_1 = p_3 = 1$, $p_2 = 2$. Let schedule S be given by $x_{1,1} = x_{1,2} = x_{2,3} = 1$ and $x_{2,1} = x_{2,2} = x_{1,3} = 0$, e.g. nurse 1 completes jobs 1 and 2, and nurse 2 does job 3. S is feasible with $C_1 = x_{1,1}p_1 + x_{1,2}p_2 + x_{1,3}p_3 = 3$ and $C_2 = 1$. So nurse 1 is critical, i.e. serves the longest shift, and both jobs 1 and 2 are critical. Since $C_1 = 3 > 2 = 1 + 1 = C_2 + p_1$ and $C_1 + p_3 = 4 > 3 = C_2 + p_2$, schedule S violates SEP and PEP.

So, given the feasibility AF $(Args_F, \rightsquigarrow_F)$, construct the optimality AF $(Args_S, \rightsquigarrow_S)$ specific to S with:

- $Args_S = Args_F = \{\mathsf{a}_{i,j} \,:\, i \in \{1,2\},\ j \in \{1,2,3\}\}$,
- $\rightsquigarrow_S = \{(\mathsf{a}_{i,j}, \mathsf{a}_{i,l}) \,:\, j \neq l\} \setminus \{(\mathsf{a}_{1,1}, \mathsf{a}_{2,1})\} \cup \{(\mathsf{a}_{2,3}, \mathsf{a}_{1,2})\}$.

Figure 3b) depicts the resulting $(Args_S, \rightsquigarrow_S)$. The extension $\{\mathsf{a}_{1,1}, \mathsf{a}_{1,2}, \mathsf{a}_{2,3}\} \approx S$ is not stable in $(Args_S, \rightsquigarrow_S)$ because (i) it has conflict $\mathsf{a}_{2,3} \rightsquigarrow_S \mathsf{a}_{1,2}$ and (ii) it does not attack $\mathsf{a}_{2,1}$.

As for feasibility, constructing the optimality AF, as well as verifying if a given schedule is efficient, is polynomial:

Lemma 4.4. *Given a schedule S, the optimality AF $(Args_S, \rightsquigarrow_S)$ can be constructed in $O(nm^2)$ time. Verifying whether an extension $E \subseteq Args_S$, such that $E \approx S$, is stable can be done in $O(n^2m^2)$ time.*

4.3 Fixed User Decisions

We model fixed user decisions to be able to explain why a given schedule satisfies or violates the given fixed decisions. We do this, given either a feasibility or an optimality AF, by modifying the attacks in the (underlying) feasibility AF. Specifically, a negative decision induces a self-attack on the respective argument, whereas a positive decision results in removal of all the attacks on the respective argument.

Definition 4.6. Let $(Args_F, \rightsquigarrow_F)$ be the feasibility AF, possibly underlying a given optimality AF $(Args_S, \rightsquigarrow_S)$, and let $D = (D^-, D^+)$ be fixed decisions. The *fixed decision AF* $(Args_D, \rightsquigarrow_D)$ is given by

- $Args_D = Args_F$,
- $\rightsquigarrow_D = (\rightsquigarrow_F \cup \{(\mathsf{a}_{i,j}, \mathsf{a}_{i,j}) \,:\, (i,j) \in D^-\}) \setminus \{(\mathsf{a}_{k,l}, \mathsf{a}_{i,j}) \,:\, (i,j) \in D^+, (k,l) \in \mathcal{M} \times \mathcal{J}\}$.

Fixed decisions thus result in the respective arguments being self-attacked or unattacked. This yields the following.

Theorem 4.5. *Let S be a schedule, $(Args_F, \rightsquigarrow_F)$ the feasibility AF, possibly underlying a given optimality AF $(Args_S, \rightsquigarrow_S)$, $E \approx S$, D a fixed decision, $(Args_D, \rightsquigarrow_D)$ the fixed decision AF. Then S is feasible and satisfies D iff E is stable in $(Args_D, \rightsquigarrow_D)$.*

Proof. Similar to the proof of Theorem 4.1, using Definition 3.3. □

Example 4.7. In Example 4.3, let $D^- = \{(2,2)\}$ and $D^+ = \{(1,1)\}$, e.g. nurse 2 cannot do job 2 and nurse 1 must do job 1. Then $D = (D^-, D^+)$. The fixed decision AF $(Args_D, \rightsquigarrow_D)$ is depicted in Figure 3c). It has a unique stable extension $\{\mathsf{a}_{1,1}, \mathsf{a}_{1,2}\}$, which corresponds to the unique feasible schedule satisfying D, in which nurse 1 is assigned both jobs.

Clearly, modeling fixed decisions is polynomial:

Lemma 4.6. *Given a schedule S, the fixed decision AF $(Args_D, \rightsquigarrow_D)$ can be constructed in $O(n^2m^2)$ time. Verifying whether the extension $E \subseteq Args_D$, such that $E \approx S$, is stable can be done in $O(n^2m^2)$ time.*

Efficiently capturing feasibility constraints, optimality conditions and fixed decisions in AA allows us to provide tractable explanations refuting or certifying 'goodness' of the schedules generated by the optimization solver or else proposed by the user. We do this next.

5 Explanations

We here formally define argumentative explanations as to why a given schedule is not 'good'. We define two types of explanations: in terms of *attacks* and *non-attacks* in AFs.

At a high-level, if a given schedule S is not feasible, efficient or violates fixed decisions, the formal argumentative explanations allow to identify which assignments, represented by arguments, are responsible. In addition, existence or non-existence of attacks with respect to the identified arguments determines the reasons as represented by the attack relationships of different AFs: feasibility, optimality and fixed decisions. Thus identified assignments and reasons allow to instantiate argumentative explanations with templated natural language generated (NLG) explanations to be given to the user, e.g. as in (Zhong et al. 2019). Further, if S is 'good' as far as AFs can model, an NLG explanation can be given relating to the properties satisfied by S.

5.1 Explanations via Attacks

Explanations via attacks concern schedule feasibility, pairwise exchanges and negative fixed decisions. We focus on attacks among arguments in the extension E corresponding to a given schedule S. These attacks make E non-conflict-free and hence not stable, and arise whenever S is not feasible due to some job assigned to more than one machine, or violates either PEP or negative fixed decisions. We exploit this to define argumentative explanations for why S is not feasible, not efficient or violates fixed decisions.

Definition 5.1. Let S be a schedule, $E \approx S$ and $(Args, \rightsquigarrow) \in \{(Args_F, \rightsquigarrow_F), (Args_S, \rightsquigarrow_S), (Args_D, \rightsquigarrow_D)\}$. We say that *an attack* $\mathsf{a} \rightsquigarrow \mathsf{b}$ *with* $\mathsf{a}, \mathsf{b} \in E$ *explains why* S:
- *is not feasible*, when $(\mathsf{a}, \mathsf{b}) \in \rightsquigarrow_F$;
- *is not efficient*, when $(\mathsf{a}, \mathsf{b}) \in \rightsquigarrow_S \setminus \rightsquigarrow_F$;
- *violates fixed decisions*, when $(\mathsf{a}, \mathsf{b}) \in \rightsquigarrow_D \setminus \rightsquigarrow_F$.

So, if a given schedule S (i) is either not feasible due to some job assigned to more than one machine ($\rightsquigarrow_F$), (ii) or is not efficient due to some improving pairwise exchange ($\rightsquigarrow_S \setminus \rightsquigarrow_F$), (iii) or assigns some job contrary to a negative fixed decision ($\rightsquigarrow_D \setminus \rightsquigarrow_F$), then the particular reason together with the relevant assignments is indicated. This allows to give an NLG explanation via template

> S { • is not feasible; • is not efficient; • violates fixed decisions} because attack $\mathsf{a}_{i,j} \rightsquigarrow \mathsf{a}_{k,l}$ shows that { • two machines i and k are assigned the same job $j = l$; • S can be improved by swapping jobs j and l on machines i and k; • job $i = k$ is assigned to machine $j = l$ contrary to the negative fixed decision (i, j)}

with cases chosen and indices i, j, k, l instantiated accordingly. We exemplify this in the three settings next.

Example 5.2. In Example 4.3, let schedule S be given by $x_{1,1} = x_{2,1} = 1$ and $x_{1,2} = x_{2,2} = 0$. S is not feasible, because job 1 is assigned to 2 machines (e.g. nurses). We have $S \approx E = \{\mathsf{a}_{1,1}, \mathsf{a}_{2,1}\}$. Any of the attacks $\mathsf{a}_{1,1} \rightsquigarrow \mathsf{a}_{2,1}$ and $\mathsf{a}_{2,1} \rightsquigarrow \mathsf{a}_{1,1}$ in the feasibility AF $(Args_F, \rightsquigarrow_F) = (Args, \rightsquigarrow)$ explains why S is not feasible: see Figure 3a). One NLG explanation is: S is not feasible because attack $\mathsf{a}_{1,1} \rightsquigarrow \mathsf{a}_{2,1}$ shows that two machines (e.g. nurses) 1 and 2 are assigned the same job 1.

Example 5.3. In Example 4.5, the attack $\mathsf{a}_{2,3} \rightsquigarrow \mathsf{a}_{1,2}$ in the optimality AF $(Args_S, \rightsquigarrow_S) = (Args, \rightsquigarrow)$ explains why $S \approx \{\mathsf{a}_{1,1}, \mathsf{a}_{1,2}, \mathsf{a}_{2,3}\}$ is not efficient, particularly as it violates PEP: see Figure 3b). The NLG explanation is: S is not efficient because attack $\mathsf{a}_{2,3} \rightsquigarrow \mathsf{a}_{1,1}$ shows that S can be improved by swapping jobs 3 and 1 between machines (e.g. nurses) 2 and 1.

Example 5.4. In Example 4.7, the self-attack $\mathsf{a}_{2,2} \rightsquigarrow \mathsf{a}_{2,2}$ in the fixed decision AF $(Args_D, \rightsquigarrow_D) = (Args, \rightsquigarrow)$ explains why $S \approx \{\mathsf{a}_{1,1}, \mathsf{a}_{2,2}\}$ violates the negative fixed decision D^- represented by the self-attack: see Figure 3c). The NLG explanation is: S violates fixed decisions because attack $\mathsf{a}_{2,2} \rightsquigarrow \mathsf{a}_{2,2}$ shows that job 2 is assigned to machine (e.g. nurse) 2 contrary to the negative fixed decision $(2,2)$.

5.2 Explanations via Non-Attacks

Explanations via non-attacks concern schedule feasibility, single exchanges and positive fixed decisions. We here focus on arguments outside the extension which are not attacked by the extension E corresponding to a given schedule S. Such non-attacks result in E being not stable, and arise whenever S is not feasible due to some unassigned job, or violates either SEP or positive fixed decisions. As in the case of explanations via attacks, we exploit this to define argumentative explanations for why S is not feasible, not efficient or violates fixed decisions.

Definition 5.5. Let S be a schedule, $E \approx S$ and $(Args, \rightsquigarrow) \in \{(Args_F, \rightsquigarrow_F), (Args_S, \rightsquigarrow_S), (Args_D, \rightsquigarrow_D)\}$. We say that *a non-attack* $E \not\rightsquigarrow \mathsf{b}$ *with* $\mathsf{b} \notin E$ *explains why* S:
- *is not feasible*, when $\rightsquigarrow = \rightsquigarrow_F$;
- *is not efficient*, when $\rightsquigarrow = \rightsquigarrow_S$ and $\mathsf{b} \rightsquigarrow_S E$;
- *violates fixed decisions*, when $\rightsquigarrow = \rightsquigarrow_D$ and b is unattacked.

As with explanations via attacks, if a given schedule is not 'good', then the particular reason together with the relevant assignments is indicated. This allows to give an NLG explanation via template

S { • is not feasible; • is not efficient; • violates fixed decisions} because non-attack $E \not\rightsquigarrow \mathsf{a}_{k,l}$ shows that { • job l is not scheduled; • S can be improved by moving job l to machine k; • job l is not assigned to machine k contrary to the positive fixed decision (k,l)}

with cases chosen and indices i, j, k, l instantiated accordingly. We exemplify this in the three settings next.

Example 5.6. In Example 4.3, let schedule S be given by $x_{1,1} = 1$ and $x_{1,2} = x_{2,1} = x_{2,2} = 0$. We have $S \approx E = \{\mathsf{a}_{1,1}\}$. Both non-attacks $E \not\rightsquigarrow \mathsf{a}_{1,2}$ and $E \not\rightsquigarrow \mathsf{a}_{2,2}$ in the feasibility AF $(Args_F, \rightsquigarrow_F) = (Args, \rightsquigarrow)$ explain why S is not feasible: see Figure 3d). One NLG explanation is: S is not feasible because non-attack $E \not\rightsquigarrow \mathsf{a}_{1,2}$ shows that job 2 is not scheduled.

Example 5.7. In Example 4.5, the non-attack $E \not\rightsquigarrow \mathsf{a}_{2,1}$ in the optimality AF $(Args_S, \rightsquigarrow_S) = (Args, \rightsquigarrow)$ explains why S is not efficient, as it violates SEP: see Figure 3b). The NLG explanation is: S is not efficient because non-attack $E \not\rightsquigarrow \mathsf{a}_{2,1}$ shows that the longest completion (e.g. shift) time can be reduced by moving job 1 to machine (e.g. nurse) 2.

Example 5.8. In Example 4.7, let schedule S be given by $x_{1,2} = x_{2,1} = 1$, $x_{1,1} = x_{2,2} = 0$. Then $\{\mathsf{a}_{1,2}, \mathsf{a}_{2,1}\} = E \approx S$. In the fixed decision AF $(Args_D, \rightsquigarrow_D) = (Args, \rightsquigarrow)$, the non-attack $E \not\rightsquigarrow \mathsf{a}_{1,1}$ explains why S violates the positive fixed decision D^+ represented by the unattacked argument $\mathsf{a}_{1,1}$: see Figure 3e). The NLG explanation is: S violates fixed decisions because non-attack $E \not\rightsquigarrow \mathsf{a}_{1,1}$ shows that job 1 is not assigned to machine (e.g. nurse) 1 contrary to the positive fixed decision $(1,1)$.

5.3 Desiderata for ArgOpt

We now show that our argumentative explanations meet the desiderata stated in Section 3.

Theorem 5.1. *Let S be a schedule, $E \approx S$ and $(Args, \rightsquigarrow) \in \{(Args_F, \rightsquigarrow_F), (Args_S, \rightsquigarrow_S), (Args_D, \rightsquigarrow_D)\}$.*

- *S is not feasible / is not efficient / violates fixed decisions, respectively, iff: either there is an attack $\mathsf{a} \rightsquigarrow \mathsf{b}$ with $\mathsf{a}, \mathsf{b} \in E$ or there is a non-attack $E \not\rightsquigarrow \mathsf{b}$ with $\mathsf{b} \notin E$, explaining why S is not feasible / is not efficient / violates fixed decisions, respectively.*
- *Explaining why S is not feasible / is not efficient / violates fixed decisions can be done in $O(n^2m^2)$ time.*
- *Each explanation is polynomial in the size of S.*

Proof. The first claim follows from Definitions 4.1, 4.4, 5.1 and 5.5, Theorems 4.1, 4.3 and 4.5. The second claim follows from Lemmas 4.2, 4.4 and 4.6. The third claim follows from the fact that mapping schedules and extensions one to another (Definition 4.2) is linear in $O(nm)$. □

This result shows that ArgOpt meets the desiderata of soundness and completeness, computational and cognitive tractability. Theorem 5.1 also implies that we can provide explanations if and when the given schedule S is 'good'. Indeed, if S is feasible / efficient / satisfies fixed decisions, then the corresponding extension E is a *certificate* in the feasibility / optimality / fixed decision AF to the 'goodness' of S. This certificate can help the user understand the accompanying NLG explanations as to why the schedule is 'good'.

For instance, consider the fixed decision AF and its unique stable extension $E = \{\mathsf{a}_{1,1}, \mathsf{a}_{1,2}\}$ as in Figure 3c). E certifies that schedule $S \approx E$ where nurse 1 does both jobs 1 and 2 is feasible and meets the fixed decisions: nurse 1 is assigned job 1 because of e.g. a manager request, as per the positive fixed decision $(1,1)$ represented by the unattacked argument $\mathsf{a}_{1,1}$; similarly, nurse 1 is assigned job 2 because e.g. nurse 2 is unqualified, as per the negative fixed decision $(2,2)$ represented by the self-attacking argument $\mathsf{a}_{2,2}$.

6 Related Work

To the best of our knowledge, there are no works concerning either explainable scheduling or integrating argumentation and optimization to explain the latter. Some preliminary works consider explainable planning, e.g. (Fox, Long, and Magazzeni 2017), which is generally different from scheduling. Argumentation can also be used for making and explaining decisions, e.g. (Amgoud and Prade 2009; Zeng et al. 2018), but mainly in multicriteria decision making which is a different setting from ours. Integration-wise, abduction can be used for scheduling, as in e.g. (Kakas and Michael 2001; Bernard, Borillo, and Gaume 1991), but not for the purpose of explaining. Optimization can be used to implement argumentation solvers, e.g. by mapping AFs to constraint satisfaction problems (Bistarelli and Santini 2010), which is opposite to using argumentation to supplement optimization.

Argumentation-based explanations in the literature are by and large formalized as (sub-)graphs/trees within AFs, see e.g. (García et al. 2013; Fan and Toni 2015; Čyras et al. 2018; Rago, Cocarascu, and Toni 2018; Zeng et al. 2018). There the user needs to follow the reasoning chains represented by the graphs to deduce the reasons for why a particular argument (representing e.g. a statement, a decision, a recommendation) is acceptable. In contrast, our argumentative explanations consist of at most two decision points (arguments) and the associated relationship ((non-)attack). They can thus be seen as paths of length 1 that pinpoint exactly which decisions violate which properties for a given schedule and optimization considerations, without the need to follow possibly lengthy chains of arguments. Our explanations are thus cognitively tractable. They can also be efficiently generated and afford natural language interpretations.

Other graph-based models could be used to explain decisions of, in particular, machine learning classifiers, e.g. ordered decision diagrams as in (Shih, Choi, and Darwiche 2018) and decision trees as in (Frosst and Hinton 2017). Moreover, natural language explanations could also be used for explanations, e.g. via counterfactual statements for machine learning predictions (Sokol and Flach 2018). We leave the study of relationships and formal comparison to such approaches for future work.

7 Conclusions and Future Work

This paper introduces a paradigm for clearly explaining to a user why a proposed schedule is 'good' or not. We propose abstract argumentation as an intermediate layer between the user and the optimization solver for defining and extracting explanations. In the makespan scheduling prob-

lem, we capture three essential dimensions—feasibility, efficiency, fixed decisions—and capture them with argumentation frameworks. These proposed argumentative explanations justify whether and why a given schedule is 'good' in those dimensions. We also establish the soundness and completeness of argumentative explanations, prove that they can be efficiently extracted and show how argumentative explanations can give rise to natural language explanations.

This work explicitly incorporates an example that assigns jobs to specific nurses. Each job is completed exactly once and the goal is that everyone gets to leave work as quickly as possible. Our *fixed decision* setting also recognizes that some nurses have (or lack) particular skills and therefore certain nurses cannot be assigned certain jobs. But we could incorporate more modeling requirements into this framework, e.g. introduce constraints incorporating contractual obligations such as a number of shifts per week or design the solution so that it is more robust to uncertainty (Letsios and Misener 2018). Moreover, we have shown how to incorporate explanations for some necessary optimality conditions, but we could also develop intuitive explanations for other optimality concepts such as fractional relaxations or cutting planes. We leave these extensions for future work.

Acknowledgements The authors were funded by the EPSRC project **EP/P029558/1** ROAD2H, except for Dimitrios Letsios who was funded by the EPSRC project **EP/M028240/1** Uncertainty-Aware Planning and Scheduling in the Process Industries.

Data access statement: No new data was collected in the course of this research.

References

Amgoud, L., and Prade, H. 2009. Using Arguments for Making and Explaining Decisions. *Artif. Intell.* (3-4):413–436.

Atkinson, K.; Baroni, P.; Giacomin, M.; Hunter, A.; Prakken, H.; Reed, C.; Simari, G. R.; Thimm, M.; and Villata, S. 2017. Towards Artificial Argumentation. *AI Mag.* 38(3):25–36.

Bernard, D.; Borillo, M.; and Gaume, B. 1991. From event calculus to the scheduling problem. semantics of action and temporal reasoning in aircraft maintenance. *Appl. Intell.* 1(3):195–221.

Bistarelli, S., and Santini, F. 2010. A Common Computational Framework for Semiring-based Argumentation System. In Coelho, H.; Studer, R.; and Wooldridge, M., eds., *19th European Conference on Artificial Intelligence*, 131–136.

Brucker, P. 2007. *Scheduling Algorithms (5th Ed.)*. Springer.

Burke, E. K.; De Causmaecker, P.; Berghe, G. V.; and Van Landeghem, H. 2004. The state of the art of nurse rostering. *J. Scheduling* 7(6):441–499.

Čyras, K.; Fan, X.; Schulz, C.; and Toni, F. 2018. Assumption-Based Argumentation: Disputes, Explanations, Preferences. In Baroni, P.; Gabbay, D. M.; Giacomin, M.; and van der Torre, L., eds., *Handbook Of Formal Argumentation*. College Publications.

Dung, P. M. 1995. On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-person Games. *Artif. Intell.* 77:321–357.

Dunne, P. E., and Wooldridge, M. 2009. Complexity of Abstract Argumentation. In Simari, G. R., and Rahwan, I., eds., *Argumentation in Artificial Intelligence*. Springer. 85–104.

Fan, X., and Toni, F. 2015. On Computing Explanations in Argumentation. In Bonet, B., and Koenig, S., eds., *29th AAAI Conference on Artificial Intelligence*, 1496–1502.

Fox, M.; Long, D.; and Magazzeni, D. 2017. Explainable Planning. In Aha, D. W.; Darrell, T.; Pazzani, M.; Reid, D.; Sammut, C.; and Stone, P., eds., *1st Workshop on Explainable Artificial Intelligence*.

Frosst, N., and Hinton, G. E. 2017. Distilling a neural network into a soft decision tree. In *1st International Workshop on Comprehensibility and Explanation in AI and ML*.

García, A. J.; Chesñevar, C.; Rotstein, N.; and Simari, G. R. 2013. Formalizing Dialectical Explanation Support for Argument-Based Reasoning in Knowledge-Based Systems. *J. Exp. Syst. Appl.* 40:3233–3247.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

Graham, R. L. 1969. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.* 17(2):416–429.

Kakas, A. C., and Michael, A. 2001. An abductive-based scheduler for air-crew assignment. *Appl. Artif. Intell.* 15(3):333–360.

Letsios, D., and Misener, R. 2018. Exact lexicographic scheduling and approximate rescheduling. *arXiv preprint arXiv:1805.03437*.

Leung, J. Y., ed. 2004. *Handbook of Scheduling - Algorithms, Models, and Performance Analysis*. Chapman and Hall - CRC.

Moulin, B.; Irandoust, H.; Bélanger, M.; and Desbordes, G. 2002. Explanation and Argumentation Capabilities: Towards the Creation of More Persuasive Agents. *Artif. Intell. Rev.* 17(3):169–222.

Moz, M., and Vaz Pato, M. 2007. A genetic algorithm approach to a nurse rerostering problem. *Comput. Oper. Res.* 34(3):667–691. Logistics of Health Care Management.

Rago, A.; Cocarascu, O.; and Toni, F. 2018. Argumentation-Based Recommendations: Fantastic Explanations and How to Find Them. In *27th International Joint Conference on Artificial Intelligence*, 1949–1955.

Sacchi, L.; Rubrichi, S.; Rognoni, C.; Panzarasa, S.; Parimbelli, E.; Mazzanti, A.; Napolitano, C.; Priori, S. G.; and Quaglini, S. 2015. From Decision to Shared-decision: Introducing Patients' Preferences Into Clinical Decision Analysis. *Artif. Intell. in Med.* 65(1):19–28.

Shih, A.; Choi, A.; and Darwiche, A. 2018. A Symbolic Approach to Explaining Bayesian Network Classifiers. In *27th International Joint Conference on Artificial Intelligence*, 5103–5111.

Sokol, K., and Flach, P. 2018. Conversational Explanations of Machine Learning Predictions Through Class-contrastive Counterfactual Statements. In *27th International Joint Conference on Artificial Intelligence*, 5785–5786.

Sørmo, F.; Cassens, J.; and Aamodt, A. 2005. Explanation in Case-Based Reasoning–Perspectives and Goals. *Artif. Intell. Rev.* 24(2):109–143.

Warner, D. M., and Prawda, J. 1972. A mathematical programming model for scheduling nursing personnel in a hospital. *Manag. Sci.* 19:411–422.

Williamson, D. P., and Shmoys, D. B. 2011. *The Design of Approximation Algorithms*. Cambridge University Press.

Zeng, Z.; Fan, X.; Miao, C.; Leung, C.; Jing Jih, C.; and Yew Soon, O. 2018. Context-based and Explainable Decision Making with Argumentation. In *17th International Conference on Autonomous Agents and MultiAgent Systems*, 1114–1122.

Zhong, Q.; Fan, X.; Luo, X.; and Toni, F. 2019. An Explainable Multi-Attribute Decision Model Based on Argumentation. *J. Exp. Syst. Appl.* 117:42–61.