Imperial College London

MEng. Individual Project

Imperial College of Science, Technology and Medicine

Department of Computing

Explaining Makespan Scheduling

Draft Final Report

Author:
Myles Lee

Supervisors:
Dr. Kristijonas Čyras
Dr. Dimitrios Letsios

Second Marker:
Dr. Ruth Misener

June 2019

Abstract

Scheduling arises in many decision processes and has a wide range of practical applications, such as in healthcare. Scheduling problems are modelled using mathematics, where optimisers find solutions to large scheduling problems quickly. Problems and solutions are often complex with inaccessible formulations, resulting in users interpreting solvers and solutions as black boxes. This hinders the users ability to understand why a schedule is reasonable, that is critical for decision-making.

We build an argumentation-supported tool, namely, *Schedule Explainer* that explains any makespan schedule with clarity. We will explore theoretical and practical considerations of applying argumentation to makespan scheduling.

Acknowledgements

I would like to give my gratitude to Kristijonas Čyras for his support and continued patience in shaping this project, and Dimitrios Letsios for his technical guidance towards the project.

Contents

This report, source-code of the tool and related resources can be found at `github.com/mylestunglee/aes`.

Chapter 1

Introduction

1.1 Motivation

Scheduling arises in countless decision processes and its abstract nature results in a wide range of practical applications, such as in healthcare [1, 2]. Scheduling problems are accurately modelled in mathematics, hence scheduling is often interpreted as a class of mathematical optimisation problems. With many mature and developed optimisation solvers [3], they can find solutions to large scheduling problems quickly, which are impractical using manual optimisation techniques. However, solver scalability and performance considerations often lead to complex algorithms. This combined with mathematical formulations of scheduling, result in users interpreting solvers and solutions as black boxes. Therefore, decision-making is often not transparent.

Transparency of decision making is important. Users require to understand why a particular schedule is reasonable. For example, hospital managers may seek to understand the efficiency of their staff and resources. Schedules may need to be robust to unpredictable changes in staff and resources, such as a nurse being unavailable to attend patients. In other scenarios, schedules may need to minimise staff to maximise profits while fulfilling all staff and patient requirements.

Explanations are vital in communication for understanding. With many possible schedules and many variations of scheduling problems [4], it is impractical to manually-craft explanations for every schedule of every variation. This motivates a tool to generate clear explanations.

1.2 Objectives

The first objective is to devise and implement an interactive tool that explains solvers' solutions so that schedules are human-understandable. We implement the explanation methodology outlined in the paper [5]. With argumentation, the user can easily understand scheduling in a practical environments such as nurse rostering and dialysis scheduling. The analysis of such methodology is important to understand the applicability of argumentation in practical settings.

The second objective is to extend the theoretical and practical capabilities of argumentation for scheduling. Extensions may include interval scheduling, shop scheduling and job-dependent scheduling.

1.3 Challenges

A tool satisfying such objectives are subjected solving the following challenges:

1. **Trust:** Users of the tool need to be confident that the explanations generated are true. That is, the algorithms proposed and implemented are correct.
2. **Accessibility:** Explanations generated from this tool are required to be accessible to computer novices without domain-specific knowledge of optimisation or argumentation.
3. **Applicability:** Explanations should relate to makespan schedules. The tool should generate clear explanations promptly to users.
4. **Knowledge transfer:** Explanations are constructed using knowledge structures, commonly represented using natural languages or diagrams. A challenge would be to effectively explore the usefulness of using natural languages compared to diagrams.
5. **Background:** Explainable planning is relatively new research area compared to optimisation [6]. Challenges arise from finding related literature on the project area.

1.4 Contributions

The main contributions of this project are listed as follows:

- A new tool *Schedule Explainer* that implements the concepts behind using argumentation for makespan scheduling. (Appendix C)
- Algorithms and their optimisations, alongside with proofs as necessary. (Chapter 3)
- Theoretical applications of argumentation with theorems. (Chapter 4)
- A discussion on applicability of argumentation. (Chapter 6)

Chapter 2

Background

We will first introduce optimisation and argumentation, where our project lies at their intersection. Afterwards, we will lightly explore on the notion of explanations. Finally, we will conduct two case studies that explore existing tools.

2.1 Optimisation

Optimisation is the process of finding an optimal decision with respect to constraints, given a set of possible decisions. The optimal decision is measured by a cost function, which typically determines a numerical value representing how good a decision is. The objective is to find the optimal decision, either as a minima or maxima using systematic methods. Applications of optimisation occur in many practical situations such as in engineering, manufacturing and science. In engineering applications, accurate modelling of a problem may require discrete decisions and non-linear relationships, resulting in difficulty in finding optimal decisions. There are a wide range of techniques for optimisation. In this project, we focus on mixed-integer linear programming, in particular makespan scheduling.

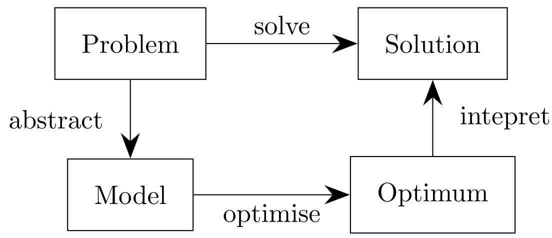


Figure 2.1: Abstraction of problem solving

Real-life problems are formulated mathematically to be optimised. Consider the following linear programming example where person wants to sell some drinks.

- Each unit of hot chocolate requires 1 litre of milk and 3 bars of chocolate.
- Each unit of milkshake requires 1 litre of milk and 2 bars of chocolate.

- The person only has 5 units of milk and 12 bars of chocolate.
- The person sells a unit of hot chocolate for 6 and a unit of milkshake for 5 monetary units.

What is the best strategy for maximising profit given that all units produced are sold? The problem is abstracted as follows. Let x and y be the number of hot chocolates and milkshakes produced respectively.

$$
\begin{aligned}
&\max_{x,y}\ 6x + 5y && \text{subject to:} \\
&\quad x + y \leq 5 && \text{milk resource constraint} \\
&\quad 3x + 2y \leq 12 && \text{chocolate resource constraint} \\
&\quad x, y \geq 0 && \text{non-negative units} \\
&\quad x, y \in \mathbb{N} && \text{whole units only}
\end{aligned}
$$

The problem is sufficiently small to be solved by inspection, but may also be solved using graphical methods or a simplex algorithm. The optimum is $x^* = 2$ and $y^* = 3$, which is interpreted by the best strategy is to produce 2 units of hot chocolate and 3 units of milkshake.

In non-linear or large optimisation problems, finding a global optima is not trivial. For gradient-based and local-search algorithms, estimated solutions can return a local optima. This is typically undesirable, however, finding a global optima result in non-polynomial complexity for arbitrary dimensional problems. In context of the project, it is intractable to compute a complete explanation of optimality in polynomial time.

2.1.1 Makespan Scheduling

The simplistic definition of makespan scheduling gives a good foundation for experimenting with argumentation. Makespan schedules are defined by $m \in \mathbb{N}$ independent machines and $n \in \mathbb{N}$ independent jobs [7]. Let $\mathcal{M} = \{1, ..., m\}$ be the set of machines and $\mathcal{J} = \{1, ..., n\}$ be the set of jobs. Each job $j \in \mathcal{J}$, has an associated processing time $p_j \in \mathbb{R}_{\geq 0}$. All processing times are collectively denoted by a vector $\mathbf{p}$. A machine can only execute at most one job at any time. For a schedule to be feasible, each job is assigned to a machine non-preemptively. For some machine $i \in \mathcal{M}$, let C_i be the completion time of the i^{th} machine. Let $C_{\max}$ be the total completion time. Let $\mathbf{x} \in \{0, 1\}^{m \times n}$ denote the assignment matrix that represents the allocation jobs to machines. Formally, makespan schedules are modelled as an optimisation problem:

$$
\begin{aligned}
&\min_{C_{\max}, \mathbf{C}, \mathbf{x}}\ C_{\max} && \text{subject to:} \\
&\forall i \in \mathcal{M}.\ C_{\max} \geq C_i \\
&\forall i \in \mathcal{M}.\ C_i = \sum_{j \in \mathcal{J}} x_{i,j} \cdot p_j \\
&\forall j \in \mathcal{J}.\ \sum_{j \in \mathcal{J}} x_{i,j} = 1 \\
&\forall i \in \mathcal{M},\ \forall j \in \mathcal{J}.\ x_{i,j} \in \{0, 1\}
\end{aligned}
$$

Definition 1. A schedule S is defined by its assignment matrix $\mathbf{x}$. S will be used to reference a high-level representation of $\mathbf{x}$ but does not specify its formal representation unlike $\mathbf{x}$, which will be used in linear programming and algorithms with its precise definition.

Definition 2. A schedule S is optimal iff S feasibly achieves the minimal total completion time.

Definition 3. A machine $i \in \mathcal{M}$ is critical iff $C_i = C_{max}$.

Definition 4. A job $j \in \mathcal{J}$ is critical iff $i \in \mathcal{M}$ is critical and $x_{i,j} = 1$.

Definition 5. A schedule satisfies the single exchange property (SEP) iff for any critical machine and any machine $i, i' \in \mathcal{M}$ and for all critical jobs $j \in \mathcal{J}$, $C_i - C_{i'} \leq p_j$

Definition 6. A schedule satisfies the pairwise exchange property (PEP) iff for any critical job and any job $j, j' \in \mathcal{J}$, if $p_j > p_{j'}$, then $C_i + p_{j'} \leq C_{i'} + p_j$.

Definition 7. A schedule S is exchange iff S satisfies SEP and PEP. Note this definition of efficiency succinctly captures necessary optimally conditions, which differs from the property efficiency defined in the paper [5].

Proposition 1. Schedule efficiency is a necessary condition for optimality [5].

Makespan schedules are often represented using cascade charts. The charts shows graphically the difference in total completion time of schedules where the problem has the parameters $m = 4$ and $n = 13$.

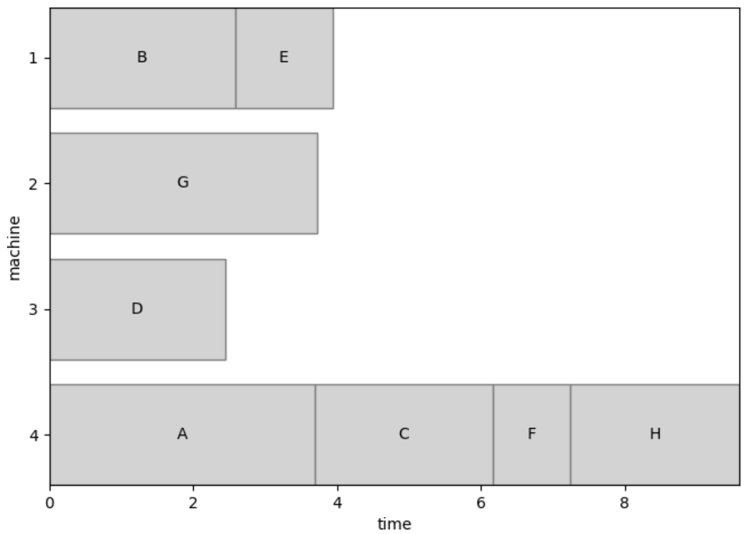


Figure 2.2: An inefficient schedule

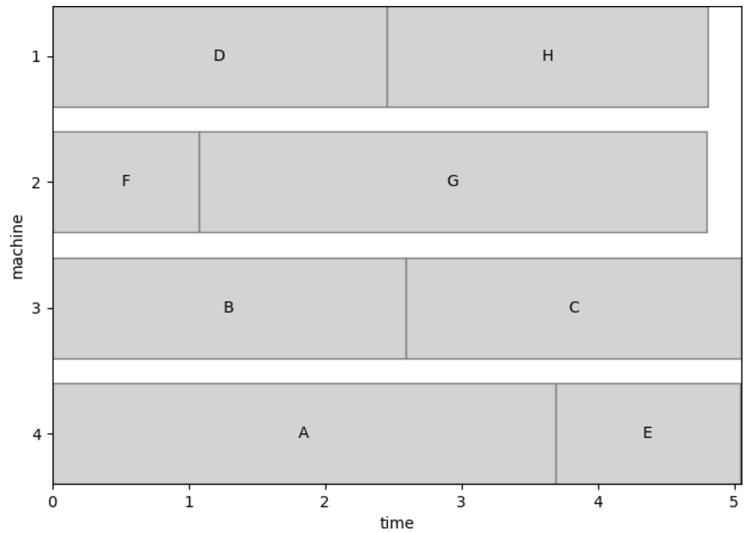


Figure 2.3: An efficient schedule

2.1.2 User Fixed Decisions

To accommodate practical applications of makespan schedules, we introduce user positive and negative fixed decisions as extensions to makespan scheduling. In a hospital scenario, positive fixed decisions capture patients exclusively allocated to a nurse, while negative fixed decisions capture unavailable or incompatible nurses and patients [5]. Let $D^-, D^+ \subseteq \mathcal{M} \times \mathcal{J}$ be the negative and positive fixed decisions respectively. Let D be the fixed decisions such that $D = (D^-, D^+)$.

Definition 8. A schedule S satisfies D iff $\forall (i, j) \in D^-.\ x_{i,j} = 0$ and $\forall \langle i, j \rangle \in D^+.\ x_{i,j} = 1$.

Definition 9. A fixed decision D is satisfiable iff there exists a schedule S such that S satisfies D.

Definition 10. A fixed decision D is satisfiable iff if the following necessary and sufficient conditions hold:

- D^+ and D^- are disjoint.
- $\forall \langle i, j \rangle, \langle i', j' \rangle \in D^+.\ i = i' \vee j \neq j'$
- $\forall j \in \mathcal{J}.\ \exists i \in \mathcal{M}.\ \langle i, j \rangle \notin D^-$

The relaxed definition of D allows it to be not satisfiable, which is not permitted in previous work [5]. This relaxation accommodates for poorly-formulated user problems, allowing explanations for validation of user input, which is practically useful.

2.1.3 Interval Scheduling

Interval scheduling is a natural extension to makespan scheduling with arrangement of its jobs. In practice, rearranging critical jobs may effect the feasibility of a schedule. Interval scheduling is a well researched area, with many literature proposing algorithms for variants of interval scheduling [8].

Interval scheduling is defined over $m \in \mathbb{N}$ and $n \in \mathbb{N}$ jobs where they are collectively denoted by the sets $\mathcal{M}$ and $\mathcal{J}$ such that $\mathcal{M} = \{1, ..., m\}$ and $\mathcal{J} = \{1, ..., n\}$ respectively. Each job $j \in \mathcal{J}$ must be allocated a machine $i \in \mathcal{M}$ preemptively. In addition, each job must be allocated between the interval $[s_j, f_j)$ such that $s_j < f_j$ where $s_j, f_j \in \mathbb{R}_{\geq 0}$ without loss of generality. Finally, each job is associated with a processing time $p_j \in \mathbb{R}_{\geq 0}$. No jobs are allowed to overlap. If $p_j > f_j - s_j$ is possible, then the scheduling problem has no feasible solution.

2.2 Argumentation

Argumentation is a method to understand and evaluate reasons for and against conclusions. Argumentation is useful in resolving conflicts, clarifying incomplete information and generating explanations. The precise definition of an argument varies between literature, however it is commonly agreed that arguments can attack or support other arguments. For an argument α to attack an argument β, α may critically challenge β such that acceptability of β is doubted [9]. This may be to question one of β's premises, by proposing a counter-example. For example, consider an scenario whether to sleep. Let α be "I want to sleep", β be "I have work" and γ be "I can work tomorrow". Using human intuition, we can derive that γ attacks β and β attacks α. This is represented graphically as follows.

$$\gamma \longrightarrow \beta \longrightarrow \alpha$$

If we conclude α to be acceptable, then we must not accept β. Accepting two conflicting arguments can be interpreted as a contradiction or being hypocritical. Hence, argumentation theory have notions of acceptable extensions, to decide whether some set of arguments are acceptable, with respect to different intuitions. This motivates to use abstract argumentation frameworks.

Note that this example uses implicit background knowledge, also known as enthymemes. In this scenario, one cannot sleep and work at the same time. To our advantage, scheduling problems are mathematically well-defined, so we do not encounter enthymemes in this project.

2.2.1 Abstract Argumentation Frameworks

An abstract argumentation framework (AAF) models the relation of attacks between arguments [10]. Formally, an AAF is a directed graph $\langle Args, \rightsquigarrow \rangle$ where $Args$ is the set of arguments and $\rightsquigarrow$ is a binary relation over $Args$. For $a, b \in Args$, a attacks b iff $a \rightsquigarrow b$. Attacks are extended over sets of arguments, where $A \subseteq Args \rightsquigarrow b \in Args$ iff $\exists a \in A \; a \rightsquigarrow b$. An extension E is a subset of $Args$.

Definition 11. An extension E is conflict-free iff $\forall a, b \in E.\ a \not\rightsquigarrow b$.

Definition 12. An extension E is stable iff E is conflict-free and $\forall a \in Args \backslash E.\ E \rightsquigarrow a$

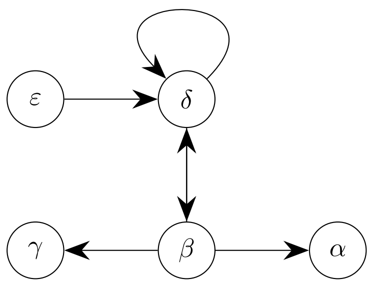


Figure 2.4: An AAF represented graphically

Consider the following example where $Args = \{\alpha, \beta, \gamma, \delta, \varepsilon\}$ and $\rightsquigarrow = \{\langle\beta, \alpha\rangle, \langle\beta, \gamma\rangle, \langle\beta, \delta\rangle, \langle\delta, \beta\rangle, \langle\delta, \delta\rangle, \langle\varepsilon, \delta\rangle\}$ as illustrated above. Then the following statements hold:

- $\varepsilon \rightsquigarrow \delta$.
- $\{\delta, \varepsilon\} \rightsquigarrow \beta$.
- $\{\alpha, \gamma\}$ is conflict-free but not stable.
- $\{\delta\}$ is not conflict-free and not stable.
- $\{\beta, \varepsilon\}$ is conflict-free and stable.

We will introduce common definitions in abstract argumentation, but these are not directly relevant in this project.

Definition 13. An extension E is admissible iff E is conflict-free and E attacks every argument attacking E.

An extension E defends an argument α iff for all every argument attacking α, E attacks such attacking argument

An extension E is complete iff E is admissible and E contains all arguments E defends.

An extension E is preferred iff E is maximally admissible with respect to $\subseteq$.

2.3 Explanations

Many explanation generation tasks appeal to either minimality or simplicity [6]. Explanations may occur over observations. For example, a sequence of states may lead to an error, an explanation can guide an user to avoid such error.

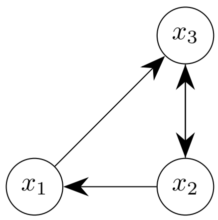
However, trustworthy and theoretical well-understood algorithms are difficult to explain to non-technical users [11]. The paper highlights concepts to explain given an optimisation context:

- Why did the optimiser do that?
- Why did the optimiser not do this?
- Why does this proposal result to more optimal result?
- Why can't this decision be taken?
- Why do I need to re-plan at this point?
- Will a better result be produced if given n more hours?

Arguably, understanding cannot be captured with a few questions. Future question include the negation, where their answers are not natively their negation.

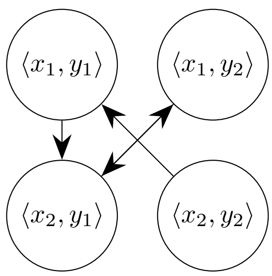
2.4 Tensors

Linear algebra is a mature area in literature. We will be using Boolean tensors as multi-dimensional arrays as data structures to manipulate AAFs. For directed graphs, we can obtain a matrix representation of its edges with an adjacency matrix [12]. If we generalise edges to multiple dimensions, then the graph's corresponding adjacency matrix becomes higher-dimensional.



$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Figure 2.5: Equivalent representations of a directed graph. Left: graphical; right: adjacency matrix.



$$\begin{bmatrix} \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \end{bmatrix}$$

Figure 2.6: A more complicated case with 2-dimensional nodes, resulting in a 4-dimensional adjacency tensor.

We will use the 2-dimension case for makespan schedules, with machines and jobs as the two dimensions, and the 3-dimensional case for interval scheduling.

2.5 Existing Tools

We will look at two scheduling software, Setmore and LEKIN, in terms of explainable planning. We will not look existing tools using argumentation because of existing tools do not use argumentation to explain scheduling.

2.5.1 Setmore

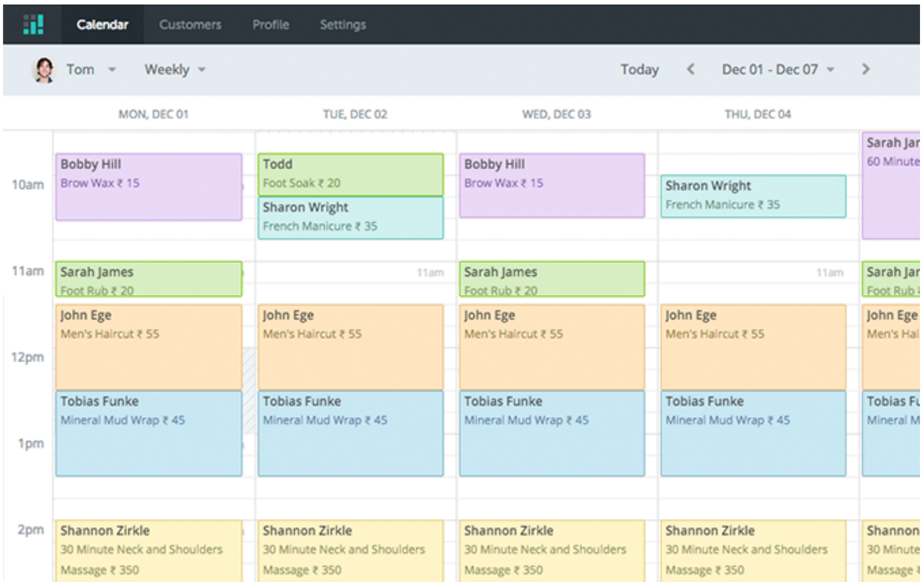


Figure 2.7: Setmore interactive interface [13]

Setmore is a commercial online application that records appointments, schedules and employees. The application is designed for small business such as in health-care [13], where managers can organise appointments on a calender. Makespan schedules are modelled daily where employees are machines and appointments are jobs. The intuitive interface enables users to quickly glance at appointments and their times, it is clear graphically when two appointments overlap. However, under the condition that one employee is able to attend at most one appointment, the user is presented with an error message. Messages also occur when resources are fully-booked or unavailable. Scheduling error messages alert the user during data input, which may prove problematic when a user wishes to input a large number of appointments. Appointments cannot be over-allocated because each appointment has at most assignment one by interface restrictions.

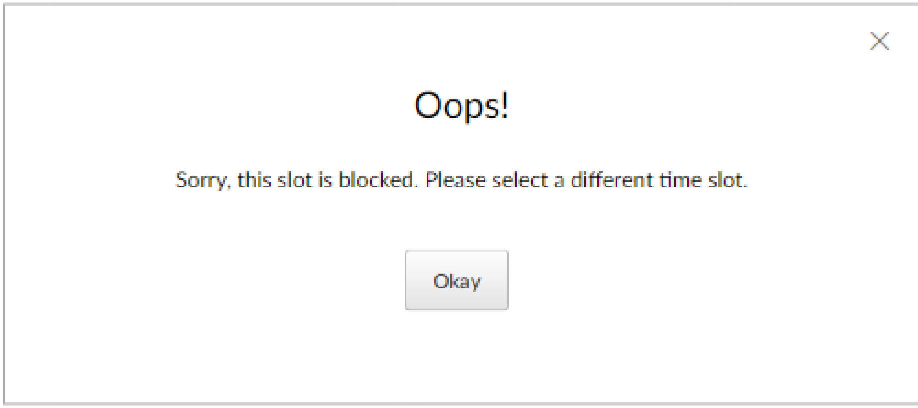


Figure 2.8: Overlapping appointment allocation error message

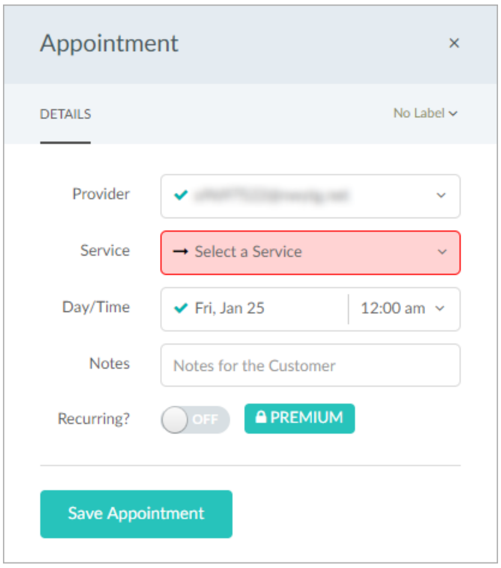


Figure 2.9: Data input of an appointment where there are no possible services, or possible assignments

We will assess the application under its free-trail, which may limit its explanative functionality. A key observation is that there is no emphasis on the concept of an optimal schedule. Because the tool is designed for small businesses and optimality is not well-defined for arbitrary objectives, the user can graphically inspect and improve a schedule with respect to the user's notion of optimality. Modification of an existing schedule is well-facilitated within its interface, where appointments can be moved or swapped between employees. Explanations for unfeasible schedules are limited to data input verification and validation error messages.

2.5.2 LEKIN

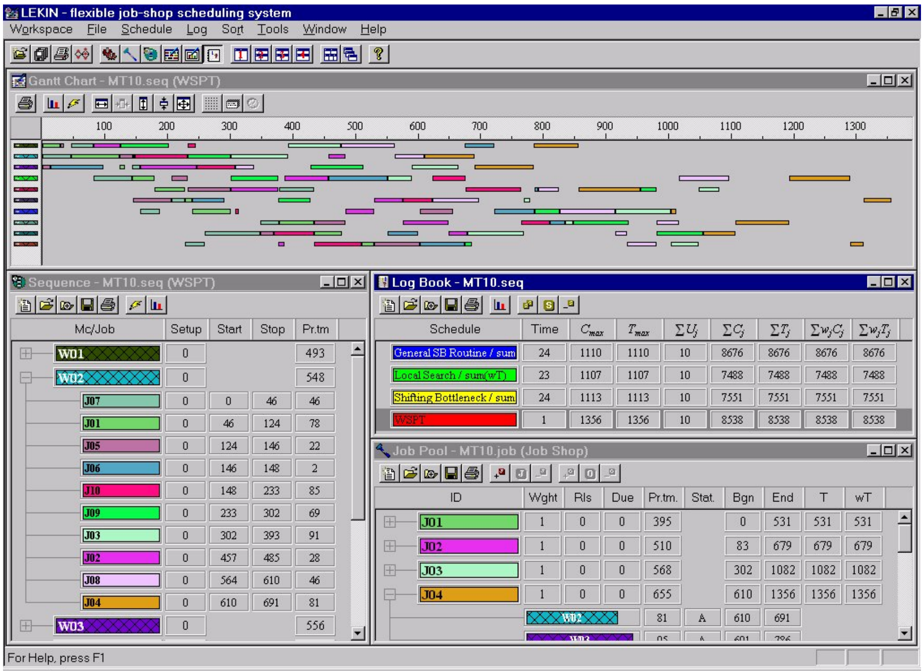


Figure 2.10: LEKIN interactive interface [14]

LEKIN is a academic-oriented scheduling application to teach students scheduling theory and its applications. The application was developed at the Stem School of Business, NYU [14]. It features numerous optimisation algorithms specialised for scheduling and draws inspiration from academic for rules and heuristics [4]. The tool supports single machines, parallel machines and flexible job shop settings.

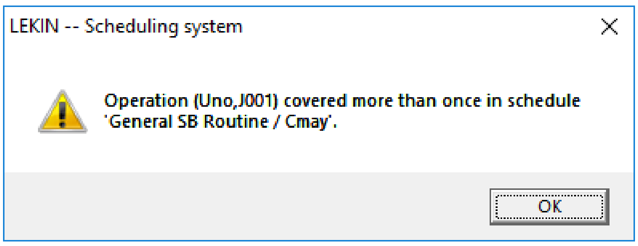


Figure 2.11: Over-allocation of a job to multiple machines

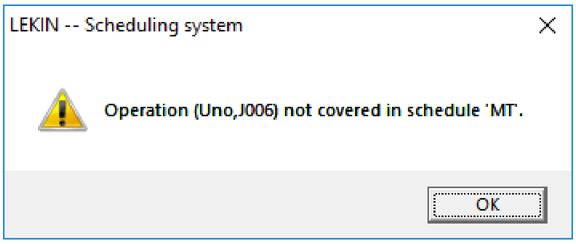


Figure 2.12: A job is not allocated to any machine

The application validates a schedule's feasibility at data input. When a schedule is infeasible, error messages appear as in Figures 2.11 and 2.12. The application computes optimal schedules, but has no functionality to verify its optimality. While our approach is to weaken optimality with efficiency, LEKIN takes an alternative approach to compute scheduling performance metrics, as shown in Figure 2.13, such as makespan completion time, tardiness and weighted metrics over machines. The advantage is that these metrics can be easily and quickly be computed and are intuitive to non-technical users given some background reading. However, these metrics are global across all machines, and give no indication to improving schedules. Non-technical users may run one of the provided optimisers to improve metrics, but no explanation is offered between the pre-optimised and the post-optimised schedules.

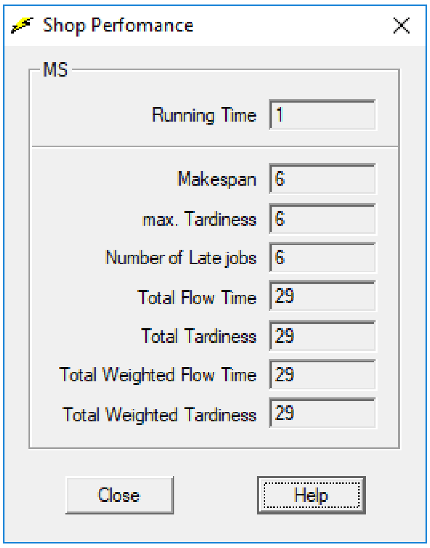


Figure 2.13: Performance metrics of an schedule

2.5.3 Comparison

It is clear that both application offers limited explanations, explicitly by error messages and implicitly by cascade charts. However, both methods require human intuition, which is unfeasible for large schedules. Therefore, for effective knowledge transfer, localised text explanations and cascade charts help users to focus on key machines and jobs.

Chapter 3

Design and Implementation

In this chapter, we will describe and discuss the technical implementation of the tool. First, we discuss our practical implementation methodology. Secondly, we define our implemented algorithms, alongside theorems to state their correctness.

3.1 Design Decisions

To choose a programming language to develop the tool, we compared languages against the requirements of the tool. The language should be interface-able with popular optimisation solvers. Interfaces are written for C++, Java, Python and other popular languages. We choose Python for its lightweight development and support for a wide range of libraries.

We balance between program speed and development time. A tool written in C may be fast but time-consuming to develop. The purpose is to demonstrate the argumentation while giving insight of future directions and short-comings. We prioritise the usability of the tool over optimal performance.

There are many powerful and efficient solvers such as CPLEX and GLPK [3]. To solver large problems, commercial solvers are used over open-source solvers for their superior speed. However, users may not have access to a commercial solver. To accommodate a wider audience, we use Pyomo to interface to both commercial and open-source solvers.

The tool features a GUI to aid its accessibility. Users such as hospital managers can often use a suitably-designed GUI without extensive training of the tool.

3.2 Structure

The tool is developed with many components, represented as individual Python files.

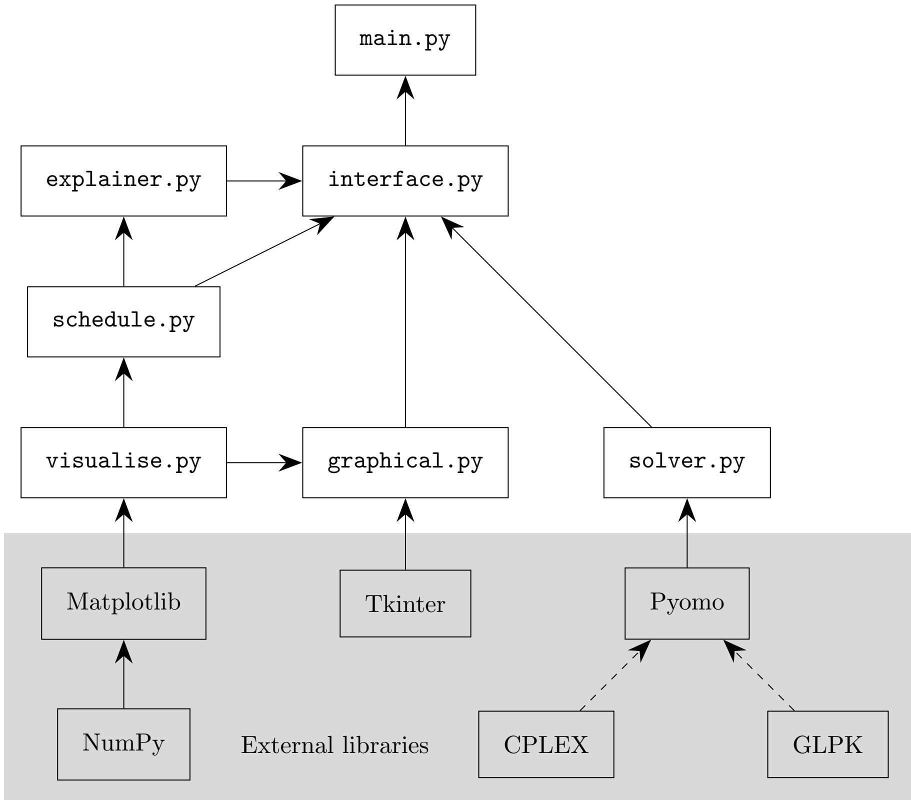


Figure 3.1: The graph illustrates the functional dependency between modules in the code-base of the tool. A solver is required for full functionality of the tool. This could be CPLEX or GLPK.

3.3 Algorithms

3.3.1 Notation

The tool uses Boolean tensors as data structures to store schedules and argumentation constructs. We will use the operators below in computing conflict-freeness, stability and aggregating explanations. The definitions below share notions with linear algebra over matrices.

Definition 14. Let $\mathbf{0}^{d_1,\ldots,d_n}$ be the zero-valued tensor. The dimensions may be omitted if clear.
For example:

$$\mathbf{0}^{2\times 2} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Definition 15. Let $\ominus$ be the element-wise logical negation operator over a Boolean tensor. Formally, $\ominus$ is a prefix unary function such that $\ominus\mathbf{x} = \mathbf{y}$ iff $\forall i_1 \in [\![1, d_1]\!] \ldots \forall i_n \in [\![1, d_n]\!]\ x_{i_1,\ldots,i_n} = 1 - y_{i_1,\ldots,i_n}$ where $\mathbf{x}$ and $\mathbf{y}$ have the same dimensions $d_1, \ldots d_n$.

For example:

$$\ominus \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Definition 16. Let $\owedge$ be the element-wise logical and operator over Boolean tensors. Formally, $\owedge$ is a infix binary function such that $\mathbf{x} \owedge \mathbf{y} = \mathbf{z}$ iff $\forall i_1 \in [\![1, d_1]\!] ... \forall i_n \in [\![1, d_n]\!]$ $x_{i_1,...,i_n} \times y_{i_1,...,i_n} = z_{i_1,...,i_n}$ where $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ have the same dimensions $d_1, ..., d_n$.

For example:

$$\begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} \owedge \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

$\owedge$ can be interpreted as the Boolean tensor adaptation of the Hadamard product.

Definition 17. Let $\ovee$ be the element-wise logical or operator over Boolean tensors. Formally, $\ovee$ is a infix binary function such that $\mathbf{x} \ovee \mathbf{y} = \mathbf{z}$ iff $\forall i_1 \in [\![1, d_1]\!] ... \forall i_n \in [\![1, d_n]\!]$ $x_{i_1,...,i_n} - x_{i_1,...,i_n} \times y_{i_1,...,i_n} + y_{i_1,...,i_n} = z_{i_1,...,i_n}$ where $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ have the same dimensions $d_1, ..., d_n$.

For example:

$$\begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} \ovee \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

3.3.2 Summary

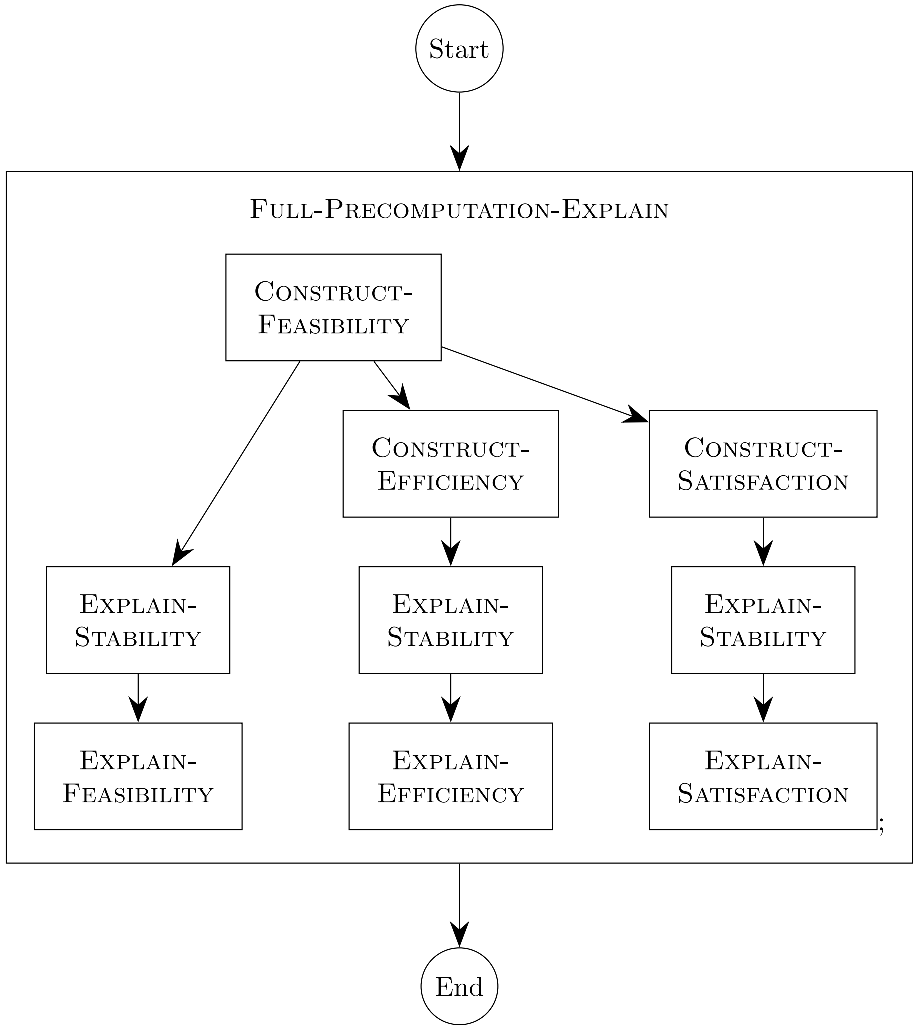


Figure 3.2: The graph summaries the required execution order of sub-functions in the FULL-PRECOMPUTATION-EXPLAIN algorithm. Nested rectangles denote nested function calls.

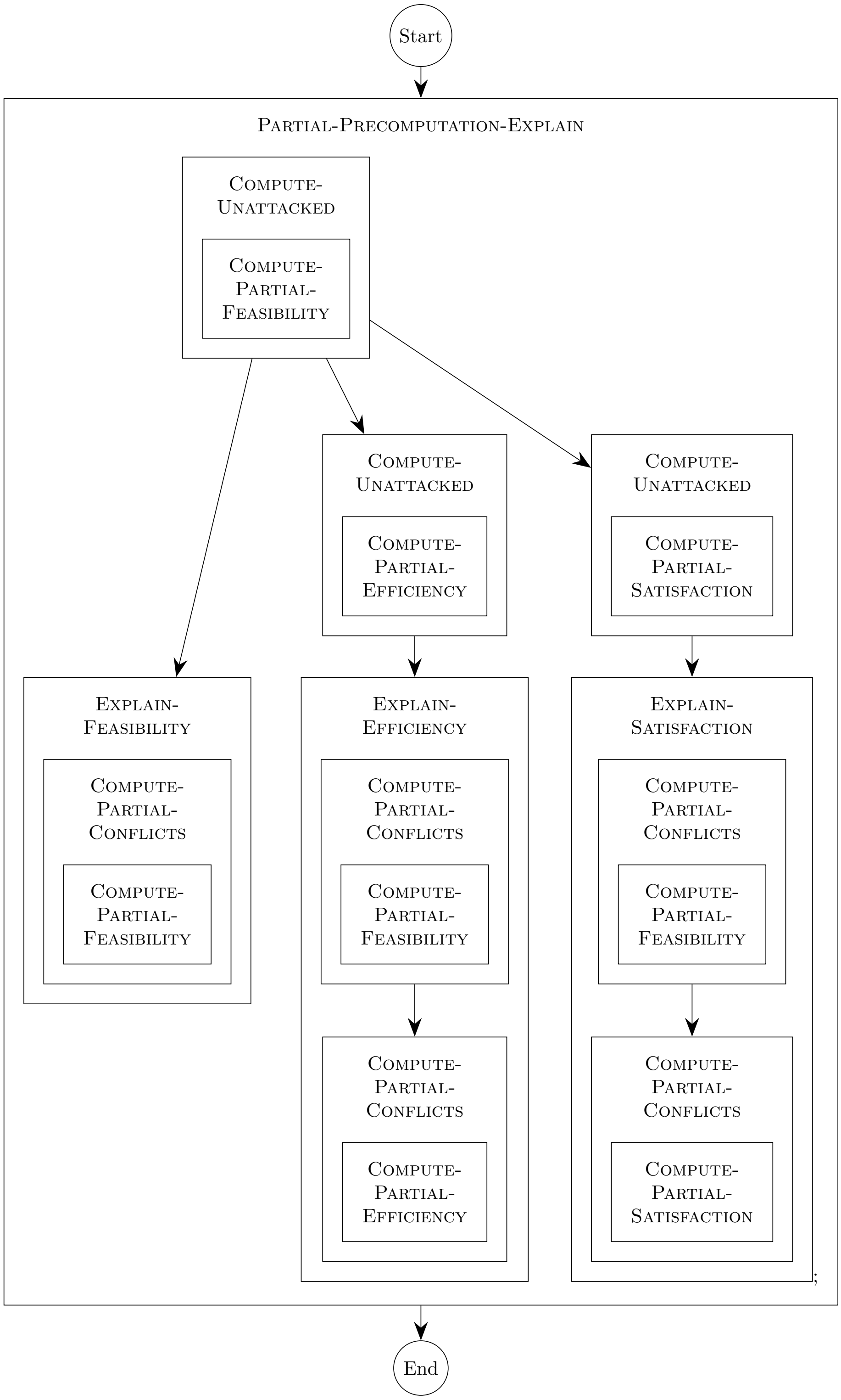


Figure 3.3: The graph summaries the required execution order of sub-functions in the Partial-Precomputation-Explain algorithm.

3.3.3 Framework Construction

The AAFs are constructed using similar definitions in paper [5]. The definitions are reprinted for feasibility and fixed decisions only. Take arbitrary $i_1, i_2 \in \mathcal{M}$ and $j_1, j_2 \in \mathcal{J}$. For the following framework definitions, let $Args = \mathcal{M} \times \mathcal{J}$.

Definition 18. The feasibility framework $\langle Args, \rightsquigarrow_F \rangle$ is defined such that $\langle i_1, j_1 \rangle \rightsquigarrow_F \langle i_2, j_2 \rangle$ iff $i_1 \neq i_2 \wedge j_1 = j_2$.

Definition 19. The efficiency framework $\langle Args, \rightsquigarrow_S \rangle$ is defined such that $\langle i_1, j_1 \rangle \rightsquigarrow_S \langle i_2, j_2 \rangle$ iff $\langle i_1, j_1 \rangle \rightsquigarrow_F \langle i_2, j_2 \rangle \wedge \neg\text{FDASEP}(i_1, i_2, j_1) \vee \text{FDAPEP}(i_1, i_2, j_1, j_2)$ where:

- Fixed decision aware single exchange property: $\text{FDASEP}(i_1, i_2, j_1, D)$ iff

$$\begin{aligned} & C_{i_1} = C_{\max} \\ \wedge\; & x_{i_1,j_1} = 1 \\ \wedge\; & C_{i_1} > C_{i_2} + p_{j_1} \\ \wedge\; & \langle i_1, j_1 \rangle \notin D^+ \\ \wedge\; & \langle i_2, j_1 \rangle \notin D^- \end{aligned}$$

- Fixed decision aware pair-wise exchange property: $\text{FDAPEP}(i_1, i_2, j_1, j_2, D)$ iff

$$\begin{aligned} & C_{i_1} = C_{\max} \\ \wedge\; & x_{i_1,j_1} = 1 \\ \wedge\; & x_{i_2,j_2} = 1 \\ \wedge\; & i_1 \neq i_2 \\ \wedge\; & j_1 \neq j_2 \\ \wedge\; & p_{j_1} > p_{j_2} \\ \wedge\; & C_{i_1} + p_{j_2} > C_{i_2} + p_{j_1} \\ \wedge\; & \langle i_1, j_1 \rangle \notin D^+ \\ \wedge\; & \langle i_2, j_2 \rangle \notin D^+ \\ \wedge\; & \langle i_2, j_1 \rangle \notin D^- \\ \wedge\; & \langle i_1, j_2 \rangle \notin D^- \end{aligned}$$

The paper [5] defines $\rightsquigarrow_S$ as an optimality framework. This report refers to $\rightsquigarrow_S$ as an efficiency framework, as its stability is determined by necessary but not sufficient conditions for optimality. In addition, the paper does considers efficiency and satisfaction to fixed decisions independently, we extend the notion of efficiency to respect these decisions. With the paper's naive definition of efficiency, the tool would recommend exchanges which violate fixed decisions. Definition 19 distinguishes FDASEP with SEP (definition 5) and FDAPEP with PEP (definition 6).

Definition 20. The user fixed decision framework $\langle Args, \rightsquigarrow_D \rangle$ is defined such that $\langle i_1, j_1 \rangle \rightsquigarrow_S \langle i_2, j_2 \rangle$ iff $\langle i_1, j_1 \rangle \rightsquigarrow_F \langle i_2, j_2 \rangle \wedge \neg\text{DP}^+(i_1, i_2, j_1, j_2) \vee \text{DP}^-(i_1, i_2, j_1, j_2)$ where:

- Positive decision property: $\mathrm{DP}^{+}(i_1, i_2, j_1, j_2)$ iff $\langle i_2, j_2 \rangle \in D^{+}$
- Negative decision property: $\mathrm{DP}^{-}(i_1, i_2, j_1, j_2)$ iff $\langle i_1, j_1 \rangle \in D^{-} \wedge i_1 = i_2 \wedge j_1 = j_2$.

The attack relations $\leadsto_F$, $\leadsto_S$ and $\leadsto_D$ are defined on $Args^2$. In practice, the tool uses a Boolean tensor representation of these attacks.

Definition 21. Let $\twoheadrightarrow_F$ be the data structure to store $\leadsto_F$ such iff $\langle i_1, j_1 \rangle \leadsto_F \langle i_2, j_2 \rangle \iff \twoheadrightarrow_{F\ i_1, j_1, i_2, j_2} = 1$.

Definition 22. Let $\twoheadrightarrow_S$ be the data structure to store $\leadsto_S$ such iff $\langle i_1, j_1 \rangle \leadsto_S \langle i_2, j_2 \rangle \iff \twoheadrightarrow_{S\ i_1, j_1, i_2, j_2} = 1$.

Definition 23. Let $\twoheadrightarrow_D$ be the data structure to store $\leadsto_D$ such iff $\langle i_1, j_1 \rangle \leadsto_D \langle i_2, j_2 \rangle \iff \twoheadrightarrow_{D\ i_1, j_1, i_2, j_2} = 1$.

Algorithm 1

function CONSTRUCT-FEASIBILITY(m, n)
    $\twoheadrightarrow_F \leftarrow \mathbf{0}^{(m \times n)^2}$
    for $j \in \mathcal{J}, i_1, i_2 \in \mathcal{M}$ **do**
        if $i_1 \neq i_2$ **then**
            $\twoheadrightarrow_{F\, i_1, j, i_2, j} \leftarrow 1$
        end if
    end for
    return $\twoheadrightarrow_F$
end function

$\twoheadrightarrow_F$ can be constructed trivially in a dense data structure in $\mathcal{O}(m^2 n^2)$ computational complexity, because of the complexity of zero-initialising $\twoheadrightarrow_F$. This can be constructed in $\mathcal{O}(m^2 n)$ space complexity using a sparse data structure, but results worse computational complexity.

Algorithm 2

function Construct-Efficiency(m, n, $\mathbf{p}$, $\mathbf{x}$, D, $\twoheadrightarrow_F$)
    $\mathbf{C} \leftarrow \mathbf{x} \cdot \mathbf{p}$
    $C_{\max} \leftarrow \max(\mathbf{C})$
    $\twoheadrightarrow_S \leftarrow \twoheadrightarrow_F$
    for $i_1 \in \mathcal{M}$ **do**
        if $C_{i_1} = C_{\max}$ **then**
            for $j_1 \in \mathcal{J}$ **do**
                if $x_{i_1,j_1} = 1$ **then**
                    for $i_2 \in \mathcal{M}$ **do**
                        if $\mathrm{FDASEP}(i_1, j_1, i_2, D)$ **then**
                            ${\twoheadrightarrow_S}_{i_1,j_1,i_2,j_1} \leftarrow 0$
                        end if
                        for $j_2 \in \mathcal{J}$ **do**
                            if $\mathrm{FDAPEP}(i_1, j_1, i_2, j_2, D)$ **then**
                                ${\twoheadrightarrow_S}_{i_1,j_2,i_2,j_2} \leftarrow 1$
                            end if
                        end for
                    end for
                end if
            end for
        end if
    end for
    return $\langle \twoheadrightarrow_S, \mathbf{C} \rangle$
end function

The construction of $\twoheadrightarrow_S$ is expensive because of the explicit for-loops to iterate over the $\mathcal{M}^2\mathcal{J}^2$ space to compute the edges that satisfy PEP and to copy $\twoheadrightarrow_F$. An optimisation by computing SEP outside of the j_2 loop, because PEP is invariant of j_2. We return the value of $\mathbf{C}$ because it will be used later, rather an recompute its value when necessary

Algorithm 3

function Construct-Satisfaction(m, n, D, $\twoheadrightarrow_F$)
    $\twoheadrightarrow_D \leftarrow \twoheadrightarrow_F$
    for $\langle i, j \rangle \in D^-$ **do**
        ${\twoheadrightarrow_S}_{i,j,i,j} \leftarrow 1$
    end for
    for $\langle i_1, j_1 \rangle \in D^+$ **do**
        for $i_2 \in \mathcal{M}, j_2 \in \mathcal{J}$ **do**
            ${\twoheadrightarrow_D}_{i_2,j_2,i_1,j_1} \leftarrow 0$
        end for
    end for
    return $\twoheadrightarrow_D$
end function

If D is assumed to be satisfiable, then D^+ has at most n decisions while D^- has at most $(m-1)n$ decisions. However, if D is not necessarily satisfiable to account for poorly-formulated user problems, so in general D^+ and D^- has at most mn decisions.

3.3.4 Verifying Stability

Stability can be computed by checking whether E exists within a set of all possible stable extensions of some $\langle Args, \twoheadrightarrow\rangle$. However, a schedule cannot be reasoned on without understanding whether E may be stable on $\langle Args, \twoheadrightarrow\rangle$. Existing solutions require a complication pipeline using answer set solvers. To make the implementation of the tool easier, we adapt the stability computation to schedules into a concise algorithm.

Algorithm 4

function EXPLAIN-STABILITY($\mathbf{x}$, $\twoheadrightarrow$, $\bar{\mathbf{u}}$, $\bar{\mathbf{c}}$)
  $\mathbf{u} \leftarrow$ COMPUTE-UNATTACKED($\mathbf{x}$, $\twoheadrightarrow$, $\bar{\mathbf{u}}$)
  for $i \in \mathcal{M}, j \in \mathcal{J}$ **do**
    $c_{i,j} \leftarrow$ COMPUTE-PARTIAL-CONFLICTS($\mathbf{x}$, $\twoheadrightarrow_{i,j}$, $\bar{c}_{i,j}$)
  end for
  return $\langle \mathbf{u}, \mathbf{c} \rangle$
end function

Algorithm 5

function COMPUTE-UNATTACKED($\mathbf{x}$, $\twoheadrightarrow$, $\bar{\mathbf{u}}$)
  $\mathbf{u} \leftarrow \circleddash\ \mathbf{x}$
  for $i \in \mathcal{M}, j \in \mathcal{J}$ **do**
    if $x_{i,j} = 1$ **then**
      $\mathbf{u} \leftarrow \mathbf{u} \owedge \circleddash \twoheadrightarrow_{i,j}$
    end if
  end for
  $\mathbf{u} \leftarrow \mathbf{u} \owedge \circleddash\ \bar{\mathbf{u}}$
  return u
end function

Algorithm 6

function COMPUTE-PARTIAL-CONFLICTS($\mathbf{x}$, $\twoheadrightarrow_{i,j}$, $\bar{c}_{i,j}$)
  $c_{i,j} \leftarrow \mathbf{0}^{m \times n}$
  if $x_{i,j} = 1$ **then**
    $c_{i,j} \leftarrow \mathbf{x} \owedge \twoheadrightarrow_{i,j}$
  end if
  $c_{i,j} \leftarrow c_{i,j} \owedge \circleddash\ \bar{c}_{i,j}$
  return $c_{i,j}$
end function

The function EXPLAIN-STABILITY returns two tensors, $\mathbf{u}$ encode the unattacked nodes and $\mathbf{c}$ encode the edges are not conflict-free. $\bar{\mathbf{u}}$ and $\bar{\mathbf{c}}$ represent node and edges to ignore from returned values respectively, which are useful in tailoring explanations to particular constraints. By default, $\bar{\mathbf{u}} = \mathbf{0}$ and $\bar{\mathbf{c}} = \mathbf{0}$. The function uses $\mathbf{x}$ rather than its equivalent representation E because $\mathbf{x}$ can be manipulated directly from an optimiser in its tensor form unlike E. This results in improved performance. In addition, it is assumed that $E \subseteq Args$ so $Args$ does not need to be a parameter.

The auxiliary functions COMPUTE-UNATTACKED and COMPUTE-PARTIAL-CONFLICTS are defined such that at most $\mathcal{O}(mn)$ memory is allocated. This will be discussed in the following subsections. We state that these algorithms are correct, their proofs are in Appendix B.

Lemma 1 (COMPUTE-UNATTACKED is correct)**.** If COMPUTE-UNATTACKED($\mathbf{x}$, $\twoheadrightarrow$, $\bar{\mathbf{u}}$) $= \mathbf{u}$, then

$$\begin{aligned}\forall k \in \mathcal{M}\ \forall \ell \in \mathcal{J}\ \bar{u}_{k,\ell} = 0 \implies \Big(u_{k,\ell} = 1 \iff \neg \exists k' \in \mathcal{M}\ \exists \ell' \in \mathcal{J} \\ x_{k,\ell} = 0 \\ \wedge\ x_{k',\ell'} = 1 \\ \wedge\ \twoheadrightarrow_{k',\ell',k,\ell} = 1\Big) \\ \wedge\ \bar{u}_{k,\ell} = 1 \implies u_{k,\ell} = 0\end{aligned}$$

Lemma 2 (COMPUTE-PARTIAL-CONFLICTS is correct)**.** If COMPUTE-PARTIAL-CONFLICTS($\mathbf{x}$, $\twoheadrightarrow_{i,j}$, $\bar{\mathbf{c}}_{i,j}$) $= c_{i,j}$, then

$$\begin{aligned}\forall k \in \mathcal{M}\ \forall \ell \in \mathcal{J}\ c_{i,j,k,\ell} = 1 \iff x_{i,j} = 1 \\ \wedge\ x_{k,\ell} = 1 \\ \wedge\ \twoheadrightarrow_{i,j,k,\ell} = 1 \\ \wedge\ \bar{c}_{i,j,k,\ell} = 0\end{aligned}$$

Theorem 1 (COMPUTE-STABILITY is correct)**.** Let E be an extension on $Args$ that represents a schedule S such that $E \approx S$, with an assignment matrix $\mathbf{x}$.

If COMPUTE-STABILITY($\mathbf{x}$, $\twoheadrightarrow$, $\bar{\mathbf{u}}$, $\bar{\mathbf{c}}$) $= \langle \mathbf{u}, \mathbf{c} \rangle$, then $\mathbf{u}$ encodes the non-ignored unattacked arguments and $\mathbf{c}$ encode the non-ignored conflicting attacks.

Formally,

$$\begin{aligned}&\Big(\forall \langle k_1, \ell_1 \rangle \in Args \setminus E \\ &\quad \bar{u}_{k_1,\ell_1} = 0 \implies \big(\exists \langle k_2, \ell_2 \rangle \in E\ \langle k_2, \ell_2 \rangle \rightsquigarrow \langle k_1, \ell_1 \rangle \iff u_{k_1,\ell_1} = 0\big) \\ &\wedge\ \bar{u}_{k_1,\ell_1} = 1 \implies u_{k_1,\ell_1} = 0\Big) \\ &\Big(\forall \langle k_1, \ell_1 \rangle, \langle k_2, \ell_2 \rangle \in E \\ &\quad \bar{c}_{k_1,\ell_1,k_2,\ell_2} = 0 \implies \big(\langle k_1, \ell_1 \rangle \rightsquigarrow \langle k_2, \ell_2 \rangle \iff c_{k_1,\ell_1,k_2,\ell_2} = 1\big) \\ &\wedge\ \bar{c}_{k_1,\ell_1,k_2,\ell_2} = 1 \implies c_{k_1,\ell_1,k_2,\ell_2} = 1\Big)\end{aligned}$$

Corollary 1 (COMPUTE-STABILITY computes stability)**.**
If COMPUTE-STABILITY($\mathbf{x}$, $\twoheadrightarrow$, $\mathbf{0}$, $\mathbf{0}$) $= \langle \mathbf{0}, \mathbf{0} \rangle$ iff E is stable on $\langle Args, \rightsquigarrow \rangle$

Proof. The result holds trivially from Theorem 1, with $\bar{\mathbf{u}} = \mathbf{0}$ and $\bar{\mathbf{c}} = \mathbf{0}$. $\square$

3.3.5 Explanation

Explanations are given in italics. Implementation of algorithms use Python's `print()` to collect explanations over all algorithms in the tool's output.

Algorithm 7

```
function EXPLAIN-FEASIBILITY(u, c)
    if m = 0 then
        if n = 0 then
            There are no jobs, the schedule is trivially feasible.
        else
            There are no machines to allocate to jobs.
        end if
    else
        y ← 0^n
        z ← 0^{n×m}
        for i_1, i_2 ∈ M, j ∈ J do
            if c_{i_1,j,i_2,j} = 1 then
                y_j ← 1
                z_{j,i_1} ← 1
                z_{j,i_2} ← 1
            end if
        end for
        if u_0^T = 0 ∧ y = 0 then
            All jobs are allocated to exactly one machine.
        else
            for j ∈ J do
                if u_{0,j} = 1 then
                    Job j is not allocated to any machine.
                end if
                if y_j ≠ 0 then
                    Job j is over-allocated to machines {i | i ∈ M, z_{j,i} = 1}.
                end if
            end for
        end if
    end if
end function
```

The paper [5] does not state explanations for trivial cases when $m = 0$ or $n = 0$. The above algorithm handles these cases with additional explanations. A problem with the naive implementation of generating an explanation for each conflict in $\mathbf{c_F}$ results in k^2 explanations for k conflicting machines for a job. This results in superfluous text for the user. To summarise these explanations, the algorithm constructs a pseudo-schedule $\mathbf{z}$ which can be interpreted as $\mathbf{x}$ transposed and rows filtered if $\sum_{i \in \mathcal{M}} x_{i,j} > 1$ for all jobs j. Afterwards, the algorithm prints the non-zero indices of $\mathbf{c}'$, which refer to the machines that causes over-allocation.

The algorithm features two optimisations. The variable $\mathbf{y}$ represent over-allocated jobs. $y_j = \mathbf{0}$ is faster to compute than its equivalent $z_j = \mathbf{0}$ because y_j is an scalar aggregate over conflicting machines, unlike the vector z_j. Likewise, by the construction of $\mathbf{u_F}$, we can exploit $u_0^T = \mathbf{0} \iff \mathbf{u} = \mathbf{0}$ because $u_0^T = \frac{1}{m}\mathbf{u}^T \cdot \mathbf{1}^m$.

Algorithm 8

1: **function** EXPLAIN-EFFICIENCY($\mathbf{p}$, $\mathbf{C}$, $\mathbf{u}$, $\mathbf{c}$)
2: $\quad i_1 \leftarrow$ first argmax of $\mathbf{C}$
3: $\quad$ reasons $\leftarrow$ empty list
4: $\quad$ **for** $i_2 \in \mathcal{M}, j_1 \in \mathcal{J}$ **do**
5: $\quad\quad$ **if** $u_{i_2,j_1} = 1$ **then**
6: $\quad\quad\quad$ reason $\leftarrow$ *Job j_1 can be allocated to machine i_2.*
7: $\quad\quad\quad$ append reason to reasons
8: $\quad\quad$ **end if**
9: $\quad\quad$ **for** $j_2 \in \mathcal{J}$ **do**
10: $\quad\quad\quad$ **if** $c_{i_1,j_1,i_2,j_2} = 1$ **then**
11: $\quad\quad\quad\quad$ reason $\leftarrow$ *Job j_1 and j_2 can be swapped with machines i_1 and i_2.*
12: $\quad\quad\quad\quad$ append reason to reasons
13: $\quad\quad\quad$ **end if**
14: $\quad\quad$ **end for**
15: $\quad$ **end for**
16: $\quad$ sort reasons by $\langle$reduction, processing time$\rangle$
17: $\quad$ **if** reasons is empty **then**
18: $\quad\quad$ *All jobs satisfy single and pairwise exchange properties.*
19: $\quad$ **else**
20: $\quad\quad$ *Output reasons*
21: $\quad$ **end if**
22: **end function**

The algorithm has $\mathcal{O}(mn^2 \log(mn^2))$ computational complexity, arising from sorting the reasons generated. Explanation of efficiency results in at most m^2n^2 lines, which grows quickly for large schedules. To make this easier for the user to understand, we sort the explanations by its reduction, the amount the total completion time will reduce when an single or pairwise exchange occurs. This will highlight the most significant improvements for the user. This justifies the increased complexity with the logarithmic factor.

A key limitation with sorting by reduction, is in the cases of multiple critical machines. In this case, all reductions are zero. This is because single or pair-wise exchange results in local optimisations of the same objective value. To find a strictly more optimal schedule, we need to look k steps ahead, where k is the number of critical machines. To solve this, the tool will need to generate instructions of k actions, of single and pair-wise exchanges. For an arbitrary large schedule, this will cause an exponential explosion in k of the explanation length. We continue the assumption that exponential tractable complexity is not feasible, so therefore, an explanation for a strictly more efficient schedule is not feasible.

An alternative solution is to restrict the explanation space by giving local explanations. Hence in the algorithm, we consider only one critical machine, as in line 2. This reduces the computational complexity by a factor of m, which is significant because efficiency is the most expensive schedule property to explain.

Algorithm 9

```
function EXPLAIN-SATISFACTION($D$, $\mathbf{u}$, $\mathbf{c}$)
    for $j \in \mathcal{J}$ do
        if $\exists i \in \mathcal{M}\ \langle i, j\rangle \notin D^-$ then
            Job $j$ cannot be allocated to any machine.
        end if
        if $D^-$ and $D^+$ are not disjoint then
            Job $j$ subject to conflicting negative and positive fixed decisions.
        end if
        if $|\{i \in \mathcal{M} \mid \langle i, j\rangle \in D^+\}| > 1$ then
            Job $j$ cannot be allocated to multiple machines.
        end if
    end for
    $\mathbf{y} \leftarrow \mathbf{0}^{m\times n}$
    for $i \in \mathcal{M}$ do
        for $j \in \mathcal{J}$ do
            $\mathbf{y} \leftarrow \mathbf{y} \circledvee c_{i,j}$
        end for
    end for
    if $\mathbf{u} = \mathbf{0} \wedge \mathbf{y} = \mathbf{0}$ then
        All jobs satisfy user fixed decisions.
    else
        for $i \in \mathcal{M}, j \in \mathcal{J}$ do
            if $u_{i,j}$ then
                Job $j$ must be allocated to machine $i$.
            end if
            if $y_{i,j}$ then
                Job $j$ must not be allocated to machine $i$.
            end if
        end for
    end if
end function
```

The variable $\mathbf{y}$ refers to allocations not satisfying D^+. Because of the relaxation that D is not assumed to be satisfiable, we must check the sufficient conditions for this, and generate their explanations if necessary.

Algorithm 10

function FULL-PRECOMPUTATION-EXPLAIN(m, n, $\mathbf{p}$, D, $\mathbf{x}$)
    $\twoheadrightarrow_F \leftarrow$ CONSTRUCT-FEASIBILITY(m, n)
    $\langle \mathbf{u_F}, \mathbf{c_F} \rangle \leftarrow$ EXPLAIN-STABILITY($\mathbf{x}$, $\twoheadrightarrow_F$, $\mathbf{0}$, $\mathbf{0}$)
    EXPLAIN-FEASIBILITY($\mathbf{u_F}$, $\mathbf{c_F}$)
    $\langle \twoheadrightarrow_S, \mathbf{C} \rangle \leftarrow$ CONSTRUCT-EFFICIENCY(m, n, $\mathbf{p}$, $\mathbf{x}$, D, $\twoheadrightarrow_F$)
    $\langle \mathbf{u_S}, \mathbf{c_S} \rangle \leftarrow$ EXPLAIN-STABILITY($\mathbf{x}$, $\twoheadrightarrow_S$, $\mathbf{u_F}$, $\mathbf{c_F}$)
    EXPLAIN-EFFICIENCY($\mathbf{p}$, $\mathbf{C}$, $\mathbf{u_S}$, $\mathbf{c_S}$)
    $\twoheadrightarrow_D \leftarrow$ CONSTRUCT-SATISFACTION(m, n, $\mathbf{x}$, $\twoheadrightarrow_F$)
    $\langle \mathbf{u_D}, \mathbf{c_D} \rangle \leftarrow$ EXPLAIN-STABILITY($\mathbf{x}$, $\twoheadrightarrow_D$, $\mathbf{u_F}$, $\mathbf{c_F}$)
    EXPLAIN-SATISFACTION($\mathbf{u_D}$, $\mathbf{c_D}$)
end function

The above high-level algorithm generates explanations for feasibility, efficiency and satisfaction while summarising the interaction of construction, stability and explanation functions. The function is named with full precomputation because all frameworks are fully constructed before explanations.

3.3.6 Memory Limitations

A full framework requires at least m^2n^2 bytes space in memory. For, $m = n = 256$ this requires 4GiB. One solution is not to construct frameworks and compute their stability in sequence, but rather inline partial framework construction into frameworks. This reduces the memory complexity to $\mathcal{O}(mn)$, while keeping the same computational complexity. This obviously will be slower to compute, but this method is more scalable. Therefore, we need to modify any function requiring a data object of size m^2n^2 such as $\mathbf{c}$ and $\mathbf{x}$.

The framework construction functions are modified to compute a sub-graph from a node, given its indices. For example, CONSTRUCT-FEASIBILITY is replaced by CONSTRUCT-PARTIAL-FEASIBILITY.

Algorithm 11

function PARTIAL-PRECOMPUTATION-EXPLAIN(m, n, $\mathbf{p}$, D, $\mathbf{x}$)
    function $\twoheadrightarrow'_F(i, j)$
        return CONSTRUCT-PARTIAL-FEASIBILITY(m, n, i, j)
    end function
    function $\mathbf{c'_F}(i, j)$
        return CONSTRUCT-PARTIAL-CONFLICTS($\mathbf{x}$, $\twoheadrightarrow'_F$, $\mathbf{0}$)
    end function
    function $\twoheadrightarrow'_S(i, j)$
        return CONSTRUCT-PARTIAL-EFFICIENCY(m, n, $\mathbf{p}$, $\mathbf{x}$, D, i, j)
    end function
    function $\mathbf{c'_S}(i, j)$
        return CONSTRUCT-PARTIAL-CONFLICTS($\mathbf{x}$, $\twoheadrightarrow'_S$, $\twoheadrightarrow'_F$)
    end function
    function $\twoheadrightarrow'_D(i, j)$
        return CONSTRUCT-PARTIAL-SATISFACTION(m, n, D, i, j)
    end function
    function $\mathbf{c'_D}(i, j)$
        return CONSTRUCT-PARTIAL-CONFLICTS($\mathbf{x}$, $\twoheadrightarrow'_D$, $\twoheadrightarrow'_F$)
    end function
    $\mathbf{u_F} \leftarrow$ COMPUTED-UNATTACKED($\mathbf{x}$, $\twoheadrightarrow'_F$, $\mathbf{0}$)
    EXPLAIN-FEASIBILITY($\mathbf{u_F}$, $\mathbf{c'_F}$)
    $\mathbf{u_S} \leftarrow$ COMPUTED-UNATTACKED($\mathbf{x}$, $\twoheadrightarrow'_S$, $\mathbf{u_F}$)
    EXPLAIN-EFFICIENCY($\mathbf{u_S}$, $\mathbf{c'_S}$)
    $\mathbf{u_D} \leftarrow$ COMPUTED-UNATTACKED($\mathbf{x}$, $\twoheadrightarrow'_D$, $\mathbf{u_F}$)
    EXPLAIN-SATISFACTION($\mathbf{u_D}$, $\mathbf{c'_D}$)
end function

3.4 Testability

To ensure the the tool is robust, we used unit tests and regression tests to verify code quality. The unit tests were implemented with `pytest`, a commonly used Python library. Although it would be ideal to automate testing of the GUI, this is beyond the scope of this academic project. We have setup typical problem and schedule data sets, including non-well defined and defined problems. Due to the nature of mathematical optimisation, it is impossible to test every schedule, so we select the data sets to be representative of the tool's explanation features. An advantage to using regression tests, is that we can automate the comparison of different approaches: full-precomputation, partial-computation and naive. This will allow us to prove that the non-argumentative and argumentative approaches are functionally equivalent.

Chapter 4

Schedule Properties and Argumentation

Schedule properties such as feasibility are modelled using frameworks to explain the satisfaction of properties. The definitions of efficiency and fixed decision frameworks extend from the definition of the feasibility framework. In this chapter, this extension is interesting because, this can be generalised to reason about arbitrary number of properties, using a extended framework. We apply this generalisation to interval scheduling to illustrate applications of argumentation.

We use stability over other notions of good extensions such as admissibility and completeness to accurately model schedule constraints. This is because we know that existing problems, such as the stable marriage problem can be modelling using stability [10].

4.1 Frameworks

In order to reason about an arbitrary number of properties, we inductively construct the expressible properties over an extendable framework, denoted by $\langle Args, \rightsquigarrow_0 \rangle$. An arbitrary property P_k is modelled by the framework $\langle Args, \rightsquigarrow_k \rangle$. To be correct, we must preserve that extension E is stable on $\langle Args, \rightsquigarrow_0 \cup \rightsquigarrow_k \rangle$ if E is also stable on $\langle Args, \rightsquigarrow_0 \rangle$ and $P(S)$ holds. Let $\rightsquigarrow, \rightsquigarrow_1, \rightsquigarrow_2 \subseteq Args^2$ be arbitrary frameworks.

Definition 24. A framework $\langle Args, \rightsquigarrow \rangle$ stability-models a schedule property P iff for all extensions E and corresponding schedules S, E is stable on $\langle Args, \rightsquigarrow \rangle$ $\Leftrightarrow P(S)$

Definition 25. A framework $\langle Args, \rightsquigarrow \rangle$ conflict-models a schedule property P iff for all extensions E and corresponding schedules S, E is conflict-free on $\langle Args, \rightsquigarrow \rangle \Leftrightarrow P(S)$

These definitions are used to make concise proofs.

Proposition 2. $\rightsquigarrow_F$ stability-models feasibility [5].

Proposition 3. $\leadsto_S$ stability-models feasibility and efficiency [5].

Proposition 4. $\leadsto_D$ stability-models feasibility and satisfaction of fixed decisions [5].

Definition 26. A schedule property P is conflict-modellable iff there exists a framework that conflict-models P.

Definition 27. A schedule property P is stability-modellable iff there exists a framework that stability-models P.

Definitions 26 and 27 intuitively specifies that a property can be verified using AAFs. In context of schedules, stability-modellable constraints are useful because it shows an application of argumentation. However, a stability-modellable constraint is not equivalent to using any argumentation framework. For example, the constraint $a+b+c+d \leq 2$ is not stability-modellable because stability does not count the number of conflicting attacks or unattacked arguments. But in value-based argumentation frameworks, it is possible to define an altered form of stability that is sensitive to attack weights. To find an extension in this weighted-stability, this problem can be reduced into a subset sum problem, which cannot be solved in polynomial time.

Lemma 3. E is conflict-free on $\langle Args, \leadsto_1\rangle$ and on $\langle Args, \leadsto_2\rangle$ iff E is conflict-free on $\langle Args, \leadsto_1 \cup \leadsto_2\rangle$.

Proof. To prove the forward implication, assume E is conflict-free on $\langle Args, \leadsto_1\rangle$ and on $\langle Args, \leadsto_2\rangle$. To aim for a contradiction, assume E is not conflict-free on $\langle Args, \leadsto_1 \cup \leadsto_2\rangle$. Then there exists $e_1, e_2 \in E$ such that $e_1(\leadsto_1 \cup \leadsto_2)e_2$. Then $e_1 \leadsto_1 e_2$ or $e_1 \leadsto_2 e_2$. Both cases lead to a contradiction, so E is conflict-free on $\langle Args, \leadsto_1 \cup \leadsto_2\rangle$.

To prove the backward implication, assume E is conflict-free on $\langle Args, \leadsto_1 \cup \leadsto_2\rangle$. To aim for a contradiction, assume E is not conflict-free on $\langle Args, \leadsto_1\rangle$. Then there exists $e_1, e_2 \in E$ such that $e_1 \leadsto_1 e_2$. Then $e_1(\leadsto_1 \cup \leadsto_2)e_2$, which contradicts the most recent assumption. Therefore, E is conflict-free on $\langle Args, \leadsto_1\rangle$, and also conflict-free on $\langle Args, \leadsto_2\rangle$ by similar argument. $\square$

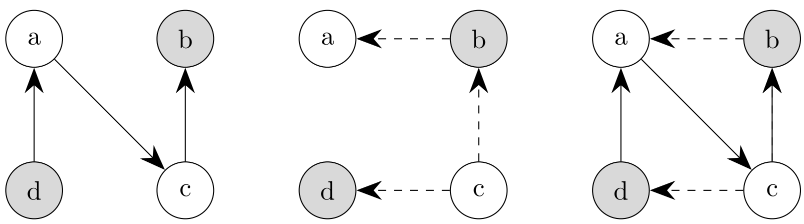


Figure 4.1: Lemma 3 states that given a conflict-free extension over two attack sets on the same arguments, the extension is conflict-free on the merged framework. The figure illustrates this by merging the left and middle frameworks to produce the right framework.

Lemma 4. If E is stable on $\langle Args, \rightsquigarrow_1 \rangle$ and E is conflict-free on $\langle Args, \rightsquigarrow_2 \rangle$, then E is stable on $\langle Args, \rightsquigarrow_1 \cup \rightsquigarrow_2 \rangle$.

Proof. Assume E is stable on $\rightsquigarrow_1$ and E is conflict-free on $\rightsquigarrow_2$. By definition of stability, $\forall a \in Args \setminus E \ \exists e \in E \ e \rightsquigarrow_1 a$. Then $\forall a \in Args \setminus E \ \exists e \in E \ e(\rightsquigarrow_1 \cup \rightsquigarrow_2)a$. So every argument not in E is attacked by some argument in E. E is conflict-free on $\rightsquigarrow_1$ because E is stable on $\rightsquigarrow_1$. Since E is conflict-free on $\rightsquigarrow_1$ and on $\rightsquigarrow_2$, we use Lemma 3 to show that E is also conflict-free on $(\rightsquigarrow_1 \cup \rightsquigarrow_2)$. Therefore E is stable on $\langle Args, \rightsquigarrow_1 \cup \rightsquigarrow_2 \rangle$. □

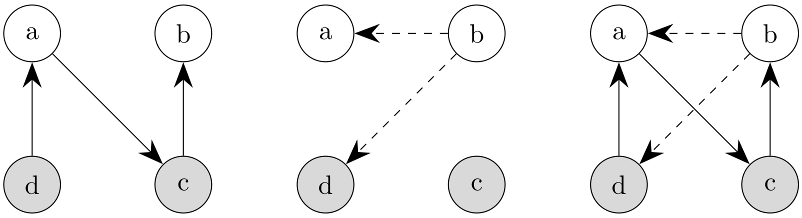


Figure 4.2: Lemma 4 allows stable extensions of attacks to grow with conflict-free attacks. The left framework is the base framework and the right framework is the extended framework.

Lemma 5. If E is stable on $\langle Args, \rightsquigarrow_1 \cup \rightsquigarrow_2 \rangle$, then E is conflict-free on $\langle Args, \rightsquigarrow_1 \rangle$.

Proof. Assume E is stable on $\langle Args, \rightsquigarrow_1 \cup \rightsquigarrow_2 \rangle$. By definition of stability, E is conflict-free on $\langle Args, \rightsquigarrow_1 \cup \rightsquigarrow_2 \rangle$. By Lemma 3, E is conflict-free on $\langle Args, \rightsquigarrow_1 \rangle$. □

Figure 4.3: Lemma 5 states that given a stable extension, removing attacks preserves the extension's conflict-freeness, as shown from left to right.

Lemma 6. If E is stable on $\langle Args, \rightsquigarrow_1 \cup \rightsquigarrow_2 \rangle$ and $\forall a \in Args \setminus E \ (\exists e \in E \ e \rightsquigarrow_2 a) \implies (\exists e \in E \ e \rightsquigarrow_1 a)$, then E is stable on $\langle Args, \rightsquigarrow_1 \rangle$.

Proof.

$$
\begin{aligned}
& E \text{ is stable on } \langle Args, \rightsquigarrow_1 \cup \rightsquigarrow_2\rangle \\
& \wedge \forall a \in Args \setminus E\ (\exists e \in E\ e \rightsquigarrow_2 a) \implies (\exists e \in E\ e \rightsquigarrow_1 a) \\
\implies & E \text{ is conflict-free on } \langle Args, \rightsquigarrow_1 \cup \rightsquigarrow_2\rangle \\
& \wedge \forall a \in Args \setminus E\ \exists e \in E\ e(\rightsquigarrow_1 \cup \rightsquigarrow_2) \\
& \wedge \forall a \in Args \setminus E\ (\exists e \in E\ e \rightsquigarrow_2 a) \implies (\exists e \in E\ e \rightsquigarrow_1 a) \\
& \textit{definition of stability} \\
\implies & E \text{ is conflict-free on } \langle Args, \rightsquigarrow_1\rangle \\
& \wedge \forall a \in Args \setminus E\ \exists e \in E\ (e \rightsquigarrow_1 a \vee e \rightsquigarrow_2 a) \\
& \wedge \forall a \in Args \setminus E\ (\exists e \in E\ e \rightsquigarrow_2 a) \implies (\exists e \in E\ e \rightsquigarrow_1 a) \\
& \textit{definition of} \cup \\
\implies & E \text{ is conflict-free on } \langle Args, \rightsquigarrow_1\rangle \\
& \wedge \forall a \in Args \setminus E\ ((\exists e \in E\ e \rightsquigarrow_1 a) \vee (\exists e \in E\ e \rightsquigarrow_2 a)) \\
& \wedge \forall a \in Args \setminus E\ (\exists e \in E\ e \rightsquigarrow_2 a) \implies (\exists e \in E\ e \rightsquigarrow_1 a) \\
& \textit{distribute} \vee \textit{ over } \exists \\
\implies & E \text{ is conflict-free on } \langle Args, \rightsquigarrow_1\rangle \\
& \wedge \forall a \in Args \setminus E\ ((\exists e \in E\ e \rightsquigarrow_1 a) \vee (\exists e \in E\ e \rightsquigarrow_1 a)) \\
& \textit{substitute} \rightsquigarrow_2 \textit{ to} \rightsquigarrow_1 \\
\implies & E \text{ is conflict-free on } \langle Args, \rightsquigarrow_1\rangle \\
& \textit{idempotency of} \vee \\
& \wedge \forall a \in Args \setminus E\ \exists e \in E\ e \rightsquigarrow_1 a \\
\implies & E \text{ is stable on } \langle Args, \rightsquigarrow_1\rangle \\
& \textit{definition of stability}
\end{aligned}
$$

□

Figure 4.4: Lemma 6 states that given a stable extension, removing attacks on multi-attacked arguments preserves the extension's stability, as shown from left to right.

Theorem 2 (Union of modelling frameworks). Let $P_0, ..., P_K$ be schedule properties with K properties. Let $P_{[\![i,j]\!]}$ be an aggregate schedule property where for all schedules S, $P_{[\![i,j]\!]}(S) \iff \forall k \in [\![i,j]\!]\ P_k(S)$.

If $\rightsquigarrow_0$ stability-models P_0, and $\forall k \in [\![1,K]\!]\ \rightsquigarrow_k$ conflict-models P_k, and for all extensions E, $\forall a \in Args \setminus E\ \forall k \in [\![1, K]\!]\ \big((\exists e \in E\ e \rightsquigarrow_k a) \implies (\exists e \in E\ e \rightsquigarrow_0 a)\big)$, then $\left(\bigcup_{k=0}^{K} \rightsquigarrow_k\right)$ stability-models $P_{[\![0,K]\!]}$.

Proof. Take arbitrary $K \in \mathbb{N}$. To prove forward implication:

1. $\rightsquigarrow_0$ stability-models P_0 — given
2. $\forall k \in [\![1, K]\!]\ \rightsquigarrow_k$ conflict-models P_k — given
3. $\forall a \in Args \setminus E\ \forall k \in [\![1, K]\!]\ \big((\exists e \in E\ e \rightsquigarrow_k a) \implies (\exists e \in E\ e \rightsquigarrow_0 a)\big)$ — given
4. E is stable on $\langle Args, \bigcup_{k=0}^{K} \rightsquigarrow_k \rangle$ — assumption
5. $\forall a \in Arg \setminus E\ \left(\left(\exists e \in E\ e \left(\bigcup_{k=0}^{K} \rightsquigarrow_k\right) a\right) \implies (\exists e \in E\ e \rightsquigarrow_0 a)\right)$ — 3
6. E is stable on $\langle Args, \rightsquigarrow_0 \rangle$ — lemma 6, 4, 5
7. $P_0(S)$ — 1, 6
8. Take arbitrary $k \in [\![1, K]\!]$
    9. E is conflict-free on $\langle Args, \rightsquigarrow_k \rangle$ — lemma 5, 4
    10. P_k(S) — 2, 9
11. $\forall k \in [\![1, K]\!]\ P_k(S)$ — 8, 10
12. $P_{[\![0,K]\!]}(S)$ — 7, 11

To prove backward implication:

1. $\rightsquigarrow_0$ stability-models P_0 — given
2. $\forall k \in [\![1, K]\!]\ \rightsquigarrow_k$ conflict-models P_k — given
3. $P_{[\![0,K]\!]}(S)$ — assumption
4. $P_0(S)$ — 3
5. E is stable on $\langle Args, \rightsquigarrow_0 \rangle$ — 1, 4
6. Recursively over $k \in [\![1, K]\!]$
    7. $P_k(S)$ — 3
    8. E is conflict-free on $\langle Arg, \rightsquigarrow_k \rangle$ — 2, 7
    9. E is stable on $\langle Arg, \bigcup_{k'=0}^{k} \rightsquigarrow_{k'} \rangle$ — lemma 4, 5, 8
10. E is stable on $\langle Arg, \bigcup_{k=0}^{K} \rightsquigarrow_k \rangle$ — 6, 9

$\square$

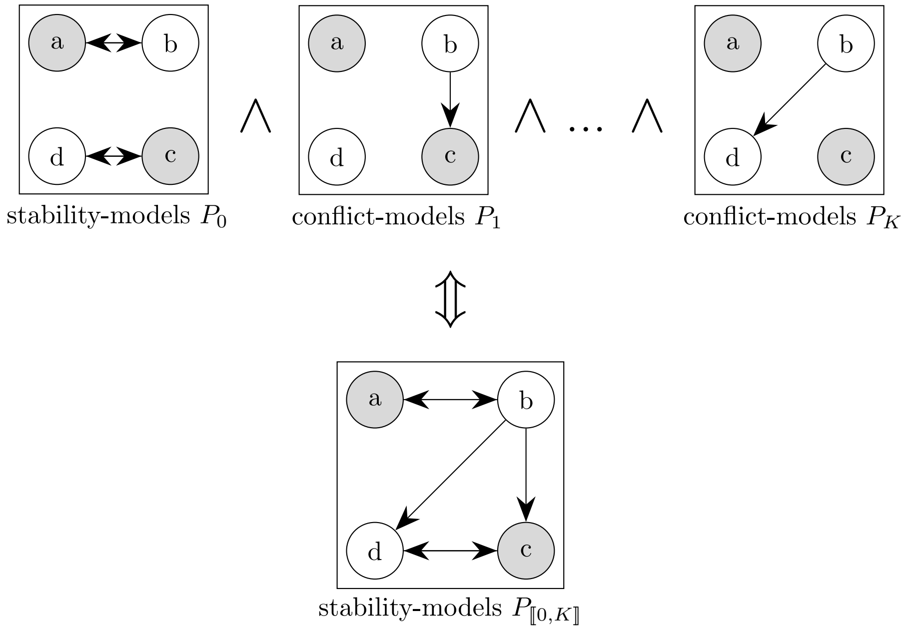


Figure 4.5: Theorem 2 allows manipulation of an aggregate property, $P_{[\![0,K]\!]}$ from carefully extending frameworks, while preserving stability. This theorem is a key statement in framing argumentation semantics for arbitrary scheduling problems.

Theorem 2 cannot be applied to $\rightsquigarrow_S$ or $\rightsquigarrow_D$ because they remove attacks from the $\rightsquigarrow_F$. The theorem does not capture removal of attacks from a commonly-extendable framework because there is ambiguity between the order of removal and insertion of attacks. Formally, $(\rightsquigarrow \cup \rightsquigarrow^+) \setminus \rightsquigarrow^- \neq (\rightsquigarrow \setminus \rightsquigarrow^-) \cup \rightsquigarrow^+$ for arbitrary frameworks $\rightsquigarrow, \rightsquigarrow^-, \rightsquigarrow^+$.

4.2 Interval Scheduling

Makespan schedules are extended to discrete time-indexed interval scheduling. We will show an application of Theorem 2 to interval scheduling. Let T be the exclusive upper-bound of indexed time where $\mathcal{T} = \{0, ..., T-1\}$. The assignment matrix $\mathbf{x} \in \mathcal{M} \times \mathcal{J} \times \mathcal{T}$ is extended such that $x_{i,j,t} = 1$ iff job j is starts work on machine i at time t. Each machine job pair $\langle i, j \rangle$ has a start time $s_{i,j} \in \mathcal{T}^{mn}$ and finish time $f_{i,j} \in \{0, ..., T\}^{mn}$, where j must be completed within the $[s_{i,j}, f_{i,j})$ interval. The objective is to minimise the total completion time.

$$\min_{\mathbf{x}} C_{\max} \text{ subject to:}$$

$$\forall i \in \mathcal{M}\ \forall j \in \mathcal{J}\ \forall t \in \mathcal{T}\ C_{\max} \geq x_{i,j,t}(t + p_j)$$

$$\forall j \in \mathcal{J}\ \sum_{i \in \mathcal{M}} \sum_{t \in \mathcal{T}} x_{i,j,t} = 1 \qquad \alpha$$

$$\forall i \in \mathcal{M}\ \forall t \in \mathcal{T}\ \sum_{j \in \mathcal{J}} \sum_{t'=\max\{t-p_j+1,0\}}^{t} x_{i,j,t'} \leq 1 \qquad \beta$$

$$\forall i \in \mathcal{M}\ \forall j \in \mathcal{J}\ \forall t \in \{0, ..., s_{i,j} - 1\}\ x_{i,j,t} = 0 \qquad \gamma$$

$$\forall i \in \mathcal{M}\ \forall j \in \mathcal{J}\ \forall t \in \{f_{i,j} - p_j + 1, ..., T - 1\}\ x_{i,j,t} = 0 \qquad \delta$$

$$\forall \langle i, j, t \rangle \in D^-\ x_{i,j,t} = 0 \qquad \varepsilon$$

$$\forall \langle i', j, t' \rangle \in D^+\ \forall i \in \mathcal{M} \setminus \{i'\}\ \forall t \in \mathcal{T}\ x_{i,j,t} = 0 \qquad \zeta$$

$$\forall \langle i, j, t' \rangle \in D^+\ \forall t \in \mathcal{T}$$
$$t \leq t' - p_j \lor t \geq t' + p_j \implies x_{i,j,t} = 0 \qquad \eta$$

α models feasibility, that all jobs must be allocated. β models that machines cannot process multiple jobs at the same time. γ and δ models the restriction of start and end times respectively. ε and ζ models negative and positive fixed decisions respectively. Equivalently, ζ can be modelled by $\forall \langle i, j, t' \rangle \in D^+\ \exists t \in \mathcal{T}\ x_{i,j,t} = 1$. ζ is defined as such to simplify the proof that the union of these properties is modellable. η models enforces that j is working on i at t, but does not specify the when j starts. Note that γ, δ, ζ and η can be modelled using ε. We use more constants to give better explanations, because each constraint have different explanations. For argumentation, let $Args = \mathcal{M} \times \mathcal{J} \times \mathcal{T}$.

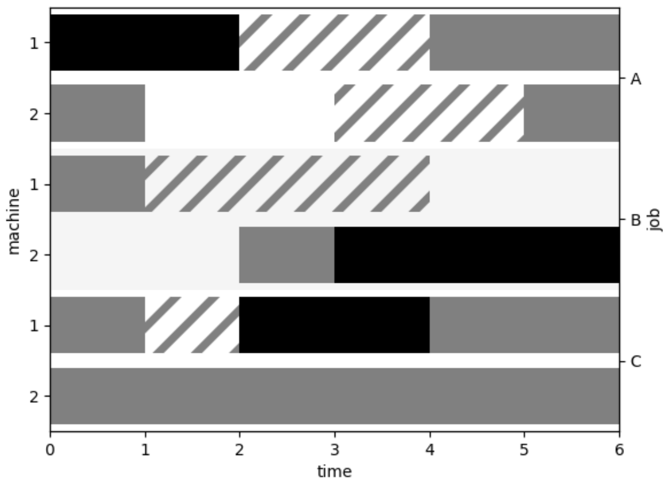


Figure 4.6: An interval schedule, where black areas are assignments and the grey areas show invalid slots because of negative fixed decisions or other job assignees.

Definition 28. Let $\rightsquigarrow_\alpha$ be the base-feasibility framework such that $\langle i, j, t\rangle \rightsquigarrow_\alpha \langle i', j', t'\rangle \Leftrightarrow i \neq i' \wedge j = j' \wedge t \neq t'$

$\rightsquigarrow_\alpha$ can be interpreted as an generalisation of $\rightsquigarrow_F$ with time.

Lemma 7. $\rightsquigarrow_\alpha$ stability-models α.

Proof. To prove forward implication: E is stable on $\langle Args, \rightsquigarrow_\alpha\rangle$. Take arbitrary $j \in \mathcal{J}$. To aim to contradict, assume $\sum_{i\in\mathcal{M}}\sum_{t\in\mathcal{T}} x_{i,j,t} > 1$. Then $\exists\langle i, j, t\rangle, \langle i', j, t'\rangle \in E$ where $x_{i,j,t} = 1$ and $x_{i',j,t'} = 1$ such that $i \neq i'$ or $t \neq t'$. By definition of $\rightsquigarrow_\alpha$, $\langle i, j, t\rangle \rightsquigarrow_\alpha \langle i', j, t'\rangle$. Hence E is not conflict-free, then E is not stable. By contradiction, $\sum_{i\in\mathcal{M}}\sum_{t\in\mathcal{T}} x_{i,j,t} \leq 1$. To aim to contradict, assume $\sum_{i\in\mathcal{M}}\sum_{t\in\mathcal{T}} x_{i,j,t} = 0$. Then $\forall i \in \mathcal{M}\ \forall t \in \mathcal{T}\ x_{i,j,t} = 0$. Then $\forall i \in \mathcal{M}\ \forall t \in \mathcal{T}\ \langle i, j, t\rangle \notin E$. Then E is not stable. By contradiction, $\sum_{i\in\mathcal{M}}\sum_{t\in\mathcal{T}} x_{i,j,t} > 0$. Therefore α holds.

To prove backward implication: From α, there is exactly one $i \in \mathcal{M}$ and $t \in \mathcal{T}$ such that $x_{i,j,t} = 1$. So E is conflict free. Also, for all j, $\langle i, j, t\rangle \in E$ attacks every other $\langle i, t\rangle$, so E is stable. $\square$

Definition 29. Let $\rightsquigarrow_\beta$ be the sequential-feasibility framework such that $\langle i, j, t\rangle \rightsquigarrow_\beta \langle i', j', t'\rangle \Leftrightarrow i = i' \wedge (t' \leq t \leq t' + p'_j \vee t \leq t' < t + p_j)$.

Lemma 8. $\rightsquigarrow_\beta$ conflict-models β.

Proof. To show E is conflict-free implies β: Take arbitrary $i \in \mathcal{M}$, $t \in \mathcal{T}$. To aim for a contradiction, assume $\sum_{j\in\mathcal{J}}\sum_{t'\in\max\{t-p_j+1,0\}} x_{i,j,t'} \geq 2$. Then there exists some $j_1, j_2 \in \mathcal{J}$ and some $t_1, t_2 \in \mathcal{T}$ such that $0 \leq t_1, t_2 \leq t$ and $x_{i,j_1,t_1} + x_{i,j_2,t_2} = 2$. Then $\langle i, j_1, t_1\rangle \in E$ and $\langle i, j_2, t_2\rangle \in E$. By conduction of β, then either $t_1 \leq t_2 \leq t_1 + p_1$ or $t_2 \leq t_1 \leq t_2 + p_2$. By definition of $\rightsquigarrow_\beta$, $\langle i, j_1, t_1 \rangle \rightsquigarrow_\beta \langle i, j_2, t_2 \rangle$. But this contradicts that E is conflict-free. Therefore β holds.

To show β implies E is conflict-free: Assume β holds. Take arbitrary $i \in \mathcal{M}$, $t \in \mathcal{T}$. Then there does not exists overlapping jobs j_1 and j_2 such that $x_{i,j_1,t_1} + x_{i,j_2,t_2} = 2$. Then $\langle i, j_1, t_1 \rangle \notin E$ and $\langle i, j_2, t_2 \rangle \notin E$. Therefore, E is conflict-free. $\square$

Definition 30. Let $\rightsquigarrow_\gamma$ be a start-feasibility framework such that $\rightsquigarrow_\gamma = \{\langle \langle i, j, t \rangle, \langle i, j, t \rangle \rangle \mid i \in \mathcal{M}, j \in \mathcal{J}, 0 \leq t < s_{i,j}\}$.

Definition 31. Let $\rightsquigarrow_\delta$ be a finish-feasibility framework such that $\rightsquigarrow_\delta = \{\langle \langle i, j, t \rangle, \langle i, j, t \rangle \rangle \mid i \in \mathcal{M}, j \in \mathcal{J}, f_{i,j} - p_j < t < T\}$.

Definition 32. Let $\rightsquigarrow_\varepsilon$ be the negative fixed decision feasibility framework such that
$\rightsquigarrow_\varepsilon = \{\langle \langle i, j, t \rangle, \langle i, j, t \rangle \rangle \mid \langle i, j \rangle \in D^-, t \in \mathcal{T}\}$.

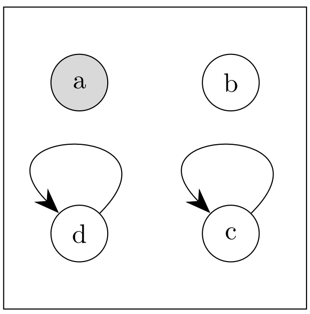
Definition 33. Let $\rightsquigarrow_\zeta$ be a positive fixed decision feasibility framework such that $\rightsquigarrow_\varepsilon = \{\langle \langle i, j, t \rangle, \langle i, j, t \rangle \rangle \mid i \in \mathcal{M}, \langle i', j, t' \rangle \in D^+, i \neq i', t \in \mathcal{T}\}$.

Definition 34. Let $\rightsquigarrow_\eta$ be a positive fixed decision feasibility framework such that $\rightsquigarrow_\varepsilon = \{\langle \langle i, j, t \rangle, \langle i, j, t \rangle \rangle \mid \langle i, j, t' \rangle \in D^+, t \in \mathcal{T}, t \leq t' - p_j \vee t \geq t' + p_j\}$.

Lemma 9. Let $\mathcal{A} \subseteq Args$ be the set of arbitrary negative fixed decisions. A schedule S satisfies these decisions if property $P_\mathcal{A}$ holds. Formally $P_\mathcal{A} \iff \forall a \in \mathcal{A}\ x_a = 0$. If $\rightsquigarrow_\mathcal{A}$ is defined by $\rightsquigarrow_\mathcal{A} = \{\langle a, a \rangle \mid a \in \mathcal{A}\}$, then $\rightsquigarrow_\mathcal{A}$ conflict-models $P_\mathcal{A}$.

Proof. To prove forward implication: Assume E is conflict free on $\langle Args, \rightsquigarrow_\mathcal{A} \rangle$. Take arbitrary $a \in \mathcal{A}$. To aim for a contradiction, assume $x_a = 1$. Then $a \in E$. By definition of $\rightsquigarrow_\mathcal{A}$, $a \rightsquigarrow_\mathcal{A} a$. But this contradicts E is conflict-free so $x_a = 0$. Therefore $P_\mathcal{A}(S)$ holds.

To prove backward implication: Assume $P_\mathcal{A}(S)$ holds. Take arbitrary $a \in \mathcal{A}$. To aim for a contradiction, assume $a \rightsquigarrow_\mathcal{A} a$. Then $a \in E$, so $x_a = 1$. This contradicts $P_\mathcal{A}(S)$, so E is conflict-free. $\square$



conflict-models $P_\mathcal{A}$ with $\mathcal{A} = \{c, d\}$

Figure 4.7: Self-attacking arguments cannot be members of stable extensions. Lemma 9 exploits self-attacks to conflict-model negative fixed decisions.

Lemma 10. For all extensions E, $\forall a \in \mathit{Args} \setminus E\ \forall \lambda \in \{\beta, \gamma, \delta, \varepsilon, \zeta, \eta\}\ \big((\exists e \in E\ e \rightsquigarrow_\lambda a) \implies (\exists e \in E\ e \rightsquigarrow_\alpha a)\big)$.

Proof. Take arbitrary extension E and arbitrary $a \in \mathit{Args} \setminus E$. If $\lambda \neq \beta$ and $\exists e \in E\ e \rightsquigarrow_\lambda a$, then $a = e$ from the definition of $\rightsquigarrow_\lambda$. But $e \notin \mathit{Args} \setminus E$. By contradiction, $\lambda = \beta$.

If $m = 0$ or $T = 0$, then $\mathit{Args} = \varnothing$, so the proof is trivial.

If $m = 1$ and $T = 1$, then by definition of $\rightsquigarrow_\beta$, $\rightsquigarrow_\beta = \varnothing$. So $\neg\exists e \in E\ e \rightsquigarrow_\beta a$. As the condition does not hold, $\exists e \in E\ e \rightsquigarrow_\alpha a$.

Otherwise, let $a = \langle i, j, t \rangle$. Because $m > 2$ or $T > 2$, then there exists i and t such that $\langle i, t \rangle \neq \langle i', t' \rangle$. By definition of $\rightsquigarrow_\alpha$, $\langle i', j, t' \rangle \rightsquigarrow_\alpha a$. $\square$

Theorem 3 (Interval schedule feasibility is stability-modellable). Let $\Lambda(S)$ iff $\forall \lambda \in \{\alpha, \beta, \gamma, \delta, \varepsilon, \zeta, \eta\}\ \lambda(S)$. Λ is stability-modellable.

Proof.

1. $\rightsquigarrow_\gamma$ conflict-models γ — definition of $\rightsquigarrow_\gamma$, lemma 9
2. $\rightsquigarrow_\delta$ conflict-models δ — definition of $\rightsquigarrow_\delta$, lemma 9
3. $\rightsquigarrow_\varepsilon$ conflict-models ε — definition of $\rightsquigarrow_\varepsilon$, lemma 9
4. $\rightsquigarrow_\zeta$ conflict-models ζ — definition of $\rightsquigarrow_\zeta$, lemma 9
5. $\rightsquigarrow_\eta$ conflict-models η — definition of $\rightsquigarrow_\eta$, lemma 9
6. $\left(\bigcup_{\lambda \in \{\alpha,\beta,\gamma,\delta,\varepsilon,\zeta,\eta\}} \rightsquigarrow_\lambda\right)$ stability-models Λ — lemma 7, lemma 8, 1, 2, 3, 4, 5, lemma 10, theorem 2
7. Λ is stability-modellable — 6

$\square$

We have shown an application of argumentation to scheduling with Theorem 2. Theorem 2 is relevant because we have shown that we can use argumentation with interval scheduling, as shown in Theorem 3. Theorem 2 is key in extensions to scheduling.

Chapter 5

Evaluation

5.1 Measures of Success

The success of the project can be measured by comparing the project results with objectives. We will also review the design choices in constructing the tool. The review will highlight the practical outcomes from using argumentation to explain scheduling.

Arguably, the most important outcome of this project is that the tool is functionality correct. This means the tool is required to explain schedules for feasibility, efficiency and satisfaction with respect to user decisions.

One objective is to implement an accessible tool. To measure accessibility, we can refer to the tractability complexity from explanations. The length of explanations either in the number of words or characters may be correlated with understandability. In addition, we conduct a survey targeted towards potential users. Because the understandability of explanations are difficult to measure, we will use open-ended questions. To carefully evaluate explanations, one would need to refer to cognitive science, which is beyond the scope of this project. We use a survey to measure the accessibility of our tool. Moreover, we can measure performance to reflect responsiveness and scalability of the tool [15]. This may be achieved by using profiling utilities to measure performance metrics such as execution time and memory consumption.

5.2 Complexity Results

We summarise our complexity results from Chapter 3.

Algorithm	Computational	Memory	Tractability
CONSTRUCT-FEASIBILITY	$\mathcal{O}(m^2n^2)$	$\mathcal{O}(m^2n^2)$	
CONSTRUCT-EFFICIENCY	$\mathcal{O}(m^2n^2)$	$\mathcal{O}(m^2n^2)$	
CONSTRUCT-SATISFACTION	$\mathcal{O}(m^2n)$	$\mathcal{O}(m^2n^2)$	
COMPUTE-UNATTACKED	$\mathcal{O}(m^2n^2)$	$\mathcal{O}(mn)$	
COMPUTE-PARTIAL-CONFLICTS	$\mathcal{O}(mn)$	$\mathcal{O}(mn)$	
EXPLAIN-STABILITY	$\mathcal{O}(m^2n^2)$	$\mathcal{O}(m^2n^2)$	
EXPLAIN-FEASIBILITY	$\mathcal{O}(mn^2)$	$\mathcal{O}(mn)$	$\mathcal{O}(mn)$
EXPLAIN-EFFICIENCY	$\mathcal{O}(mn^2\log(mn^2))$	$\mathcal{O}(mn^2)$	$\mathcal{O}(mn^2)$
EXPLAIN-SATISFACTION	$\mathcal{O}(mn)$	$\mathcal{O}(mn)$	$\mathcal{O}(mn)$
FULL-PRECOMPUTATION-EXPLAIN	$\mathcal{O}(m^2n^2\log(mn^2))$	$\mathcal{O}(m^2n^2)$	$\mathcal{O}(mn^2)$
PARTIAL-PRECOMPUTATION-EXPLAIN	$\mathcal{O}(m^2n^2\log(mn^2))$	$\mathcal{O}(mn^2)$	$\mathcal{O}(mn^2)$

Figure 5.1: Computational, memory and tractability complexity of algorithms using argumentation

Using an naive explanation approach only improves the computational complexity of verifying the feasibility property, from $\mathcal{O}(m^2n^2)$ to $\mathcal{O}(mn)$. Overall, we sse that PARTIAL-PRECOMPUTATION-EXPLAIN is better than FULL-PRECOMPUTATION-EXPLAIN by an order of m in memory.

5.3 Profiling

The tool's functionality is demonstrated in the user documentation guide in Appendix C. We will compare different implementation methods and their effectiveness using performance metrics. We will compare two algorithmic approaches to AAF construction with and the naive approach without argumentation. We implement the algorithms, then we profile for elapsed time and maximum allocated memory. The time and memory are measured with the Python's `cProfile` and `memory-profiler` modules respectively. The tool was executed on Department of Computing's virtual machines, with the specification of dual-core CPU at 2GHz with 2GiB RAM.

Time comparison:

- Elapsed time measurements are noisy.
- For less than 100 jobs, all approaches have approximately equal timings.
- From profiling, the tool takes 0.4 seconds on average to startup.
- Partial precomputation is 3% faster than full precomputation on average, excluding startup time.
- Naive is 18% faster than partial precomputation on average, excluding startup time.
- Both graphs hints at quadratic complexity.

Memory comparison:

- For less than 40 jobs, all approaches have approximately equal memory usage.

- From profiling, the tool uses 52MiB on average to startup.
- For a large number of jobs, partial-precomputation is 7 times more efficient than full precomputation excluding startup memory.
- Naive and partial-precomputation have less than 1% memory usage difference.
- Both graphs hints at quadratic complexity.

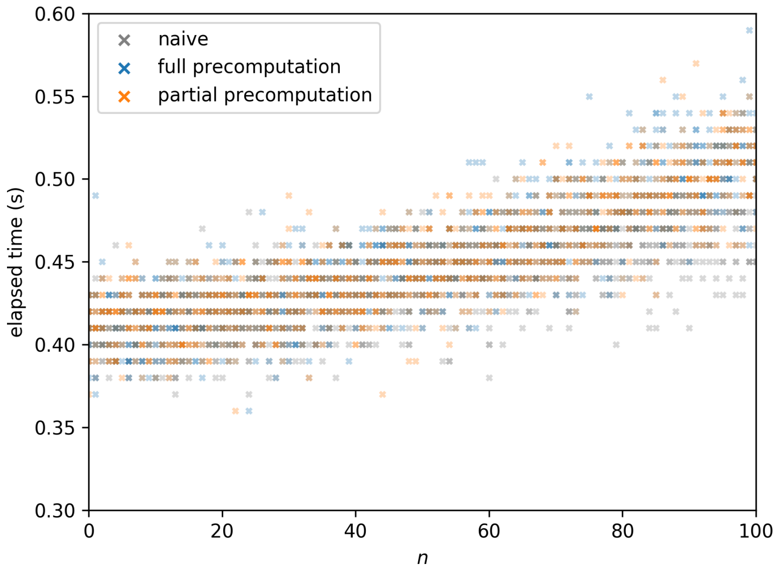


Figure 5.2: Elapsed time comparison where $m = 10$ and $0 \leq n \leq 100$ with 1000 samples.

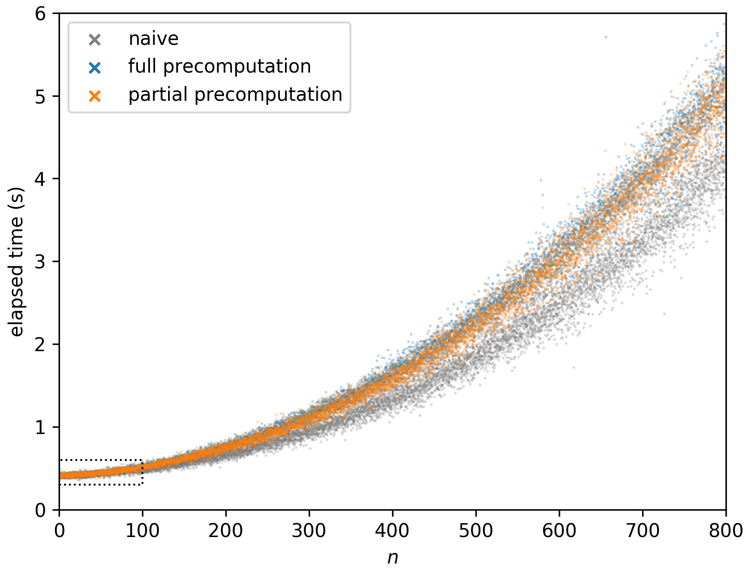


Figure 5.3: Elapsed time comparison where $m = 10$ and $0 \leq n \leq 800$ with 8000 samples.

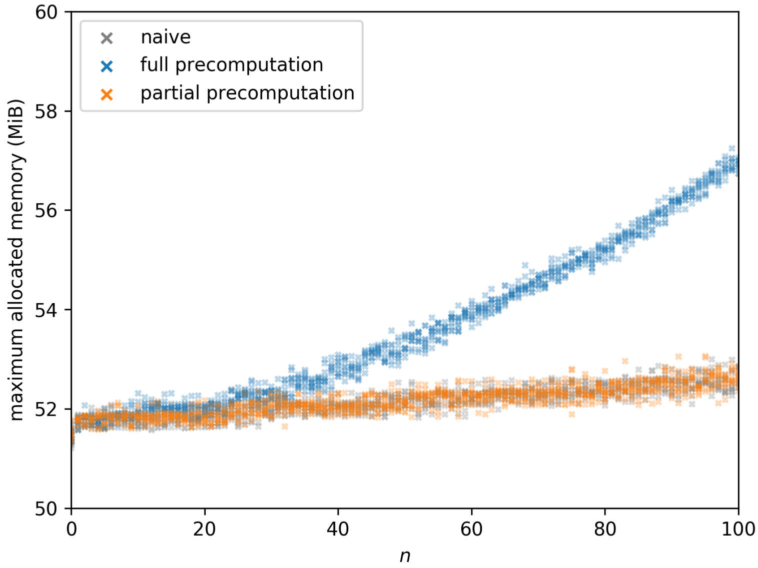


Figure 5.4: Maximum allocated memory comparison where $m = 10$ and $0 \leq n \leq 100$ with 1000 samples.

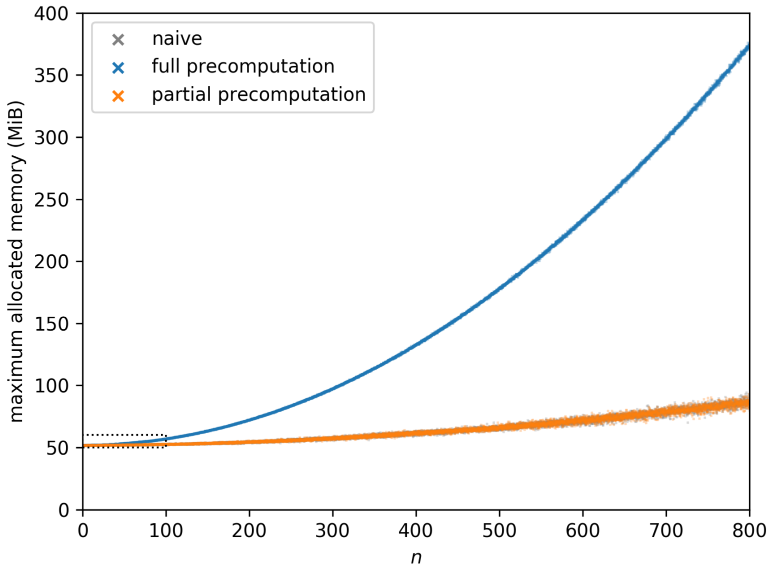


Figure 5.5: Maximum allocated memory comparison where $m = 10$ and $0 \leq n \leq 800$ with 8000 samples.

5.4 Questionnaire

We conduct surveys with three disjoint sample groups.

- **Group 1**: 10 people who are not given any explanations.
- **Group 2**: 12 people who are given simple explanations generated from the tool, but do not have interactive access to the tool.
- **Group 3**: 7 people who are given interactive access to the tool.

In all groups, we used the same questionnaire, which can be found in the appendix. Each question was carefully designed to be completed mentally while exploring concepts in makespan scheduling. We do not ask about feasibility because its validation is trivial.

- Question 1 shows an efficient but non-optimal schedule. The tool does not explain optimality, so users can see cases where the tool may not be helpful.
- Question 2 shows a schedule with more machines and jobs, to test whether users can find improvements in a non-trivial schedule. It is possible to optimise the schedule in one single exchange, but this may not be obvious.
- Question 3 shows an application of fixed decisions.
- Question 4 explores pairs of jobs, which can be modelled using fixed decisions. The question explores how intuition can be formed using additional constraints.
- Question 5 explores replanning of a schedule by insertion of a new job. Clearly, trivially inserting a new job does not result in a optimal schedule. It is possible to arrange the schedule in one single exchange before insertion to achieve an optimal updated schedule.

Although these five questions do not cover the tool's complete functionality, they seek explanations that require intuition, with or without the tool. This is important in assessing how users can operate the tool in non-trivial cases. In this survey, we asked a wide demographic, from GCSE students to graduates, including people who do have little knowledge of computer science or mathematics.

5.4.1 Group 1 Results

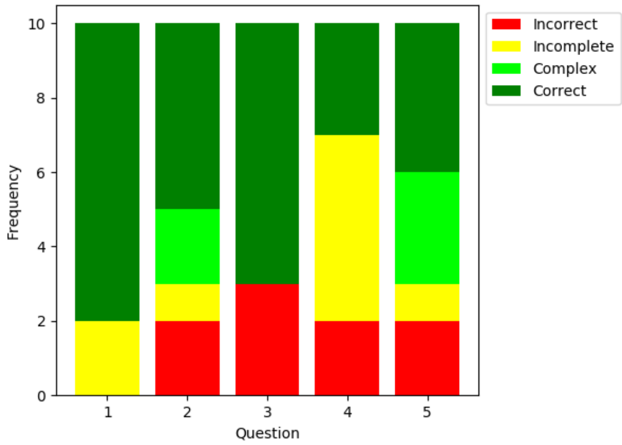


Figure 5.6: Aggregated Group 1 results from Section E.1

From Figure 5.6, we see that the majority of respondents are able to get most questions correct. Some responses are considered complex, when their answers are technically correct, but give more steps than necessary. This may hint that some intuitive ways to optimise a schedule may not minimise the number of job exchanges. This was evident in Question 2 and 5, that has many jobs. We can see in more complex schedules that users give more complex answers. Some responses are considered incomplete, where only the question was partially answered. This was evident in Question 4, where many respondents failed to identity either decision yields an optimal schedule.

5.4.2 Group 2 Results

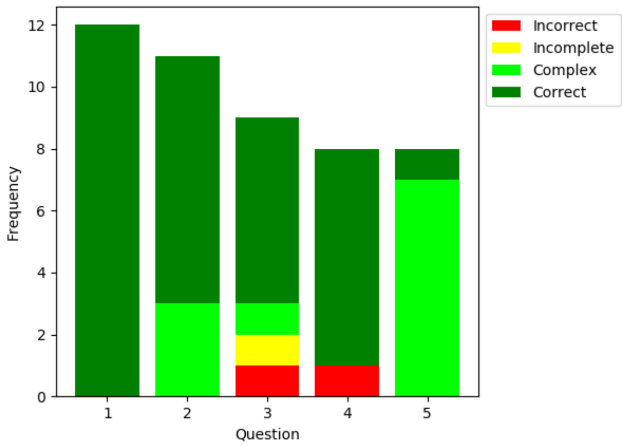


Figure 5.7: Aggregated Group 2 results from Section E.2

From Figure 5.7, we can make the following observations of Group 2 compared to Group 1:

- Group 2 respondents performed better than Group 1, and in less time, from 15 minutes 37 seconds to 9 minutes 12 seconds on average, excluding outliers.

- Group 1 respondents completed all questions, but there a few Group 2 respondents who did not answer the latter questions. This is illustrated the constant bar heights in Figure 5.6 and decreasing bar heights in Figure 5.7.

- When relevant explanations are given in Questions 2 and 5, there are more complex answers in Group 2 than Group 1. The answers suggests that respondents are likely to select the easiest to understand explanation, rather than the simplest optimal schedule. This is significant in Question 5, where many respondents used the first suggestion from the explanation without further intuition.

- Question 3 and 4 give situations in which respondents answer which situation results in a more optimal schedule. We can see that explanations significantly improve answers.

- In question 1, the explanation given was not relevant, so we would expect answers from Groups 1 and 2 would be similar because the underlying question is the same. From the data, we can see some variance between Groups 1 and 2.

5.4.3 Group 3 Results

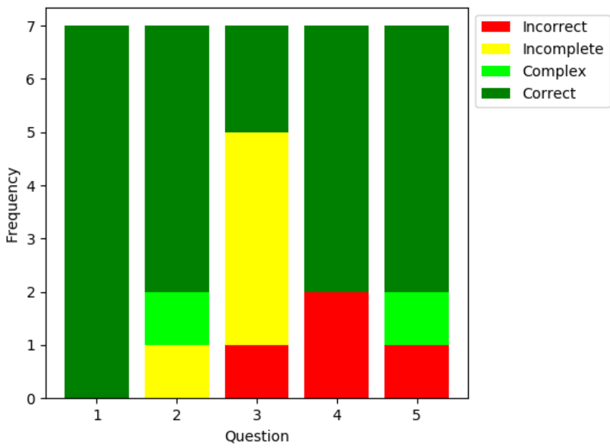


Figure 5.8: Aggregated Group 3 results from Section E.3

- Group 3 respondents took the longest to answer the questions, with 18 minutes and 27 seconds, excluding outliers. Users spent time understanding how to use the tool. To give a more fair comparison of time, users should not been able to start the questionnaire while reading the user guide.

- From question 3, many respondents said to simply move jobs to satisfy negative fixed decisions, without concern of the schedule's optimality. This suggests that users took the suggestions from the took too literally. As many users suggested local changes, many answers were considered incompletes.

5.4.4 Summary

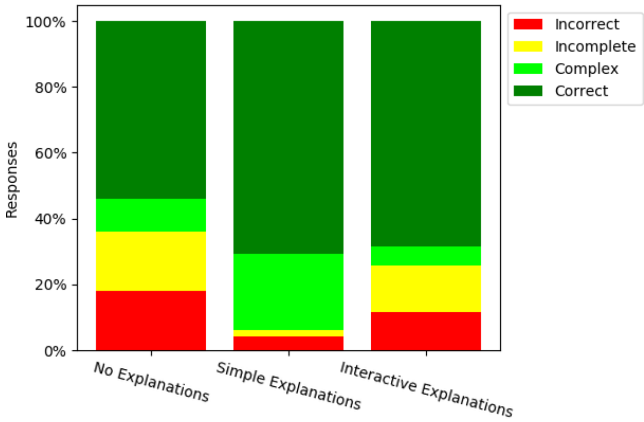


Figure 5.9: Aggregated results from Figures 5.6, 5.7, and 5.8

- Group 1 has the worst performance.
- Group 2 has the best performance, but many answers use are long and complex, often using the first obvious solution.
- Group 3 has the median performance, with many incomplete answers, using only one iteration of the tool.

5.4.5 Feedback

During development of the tool and questionnaires, I asked my colleagues and friends for feedback about the tool.

- In early versions of the tool, the GUI did not arrange textboxes and buttons into groups such as problem, schedule and explanation. Instead, all buttons and their functionality were exposed as an menu. This was confusing to new users because it was not evident which textboxes were input or output.
- The paper [5] represented machine job pair assignments as a pairs of integer indices. The tool internally uses indices to compute explanations, however exposing both machines and job as integer indices was confusing. Therefore, jobs are now represented alphabetically.

- The tool uses regular expressions to validate user input. Many users complained about the strict validation errors. We solve this by pre-processing the input by correcting missing or surplus white-space.

Chapter 6

Conclusion

6.1 Contributions

We restate our contributions from Section 1.4.

- We implement a new tool *Schedule Explainer*, as in Chapter 3.
- We define algorithms that balances between computational complexity and explanation expressibility.
- We give theoretical applications of argumentation with Theorem 2 and 3, which shows that argumentation frameworks can be overlapped to aggregate schedule properties.
- A discussion on the applicability of argumentation, as discussed in this chapter.

6.2 Limitations of Argumentation

We show that argumentation can interface between optimisation solvers and explanation generators. By computing the satisfaction of extensions, such as stability, solvers are exposed as white-box algorithms, transforming argumentation semantics into human-understandable explanations. However, this application of argumentation is flawed in practice.

- **Memory performance:** Abstract argumentation operates on directed graphs, whose data structure storage suffers from quadratic space complexity. Using the direct implementation of the paper [5], namely, the full-precomputation approach, at 256 machines and jobs, each framework requires 4GiB of memory. To solve this scalability issue, the tool instead operates on partitioned directed graphs, resulting in optimal linear space complexity. However, optimal complexity does not yield optimal performance, because at best, the partial-precomputation approach uses the same amount of memory as the non-argumentative approach, as shown by our empirical results.

- **Computational performance:** In order for argumentation semantics to correspond to schedule properties, extensions must be global to capture the space of the problem's linear constraints. Local extensions such as conflict-freeness and admissibility are insufficient to model makespan schedules. Hence, we use the stability extension, which suffers from quadratic computational complexity. This is inefficient compared to a non-argumentative approach, where makespan feasibility can be computed in linear complexity. Therefore, any argumentation approach regarding to makespan scheduling is less scalable than an non-argumentative approach.
- **Abstracted interface:** For argumentation to be exploited as a white-box, users should interact with the underlying argumentation. This is possible for small schedules where the number of machines, jobs and their attacks are limited. But for realistic schedule sizes of at least tens of machines and jobs, argumentation frameworks become too busy to be visualised clearly. As such, users will abstract the argumentative interface of the tool as a black box, so users may ignore the underlying argumentation semantics.
- **Functionality equivalence:** In a black box comparison of an argumentative approach versus a non-argumentative approach, both result in identical explanations given identical problems and schedules. This is verified with the tool by comparing results with and without the `--naive` flag over extensive test cases. In a practical setting, users will favour using the non-argumentative approach for general performance.
- **Implementation complexity:** The optimisation of the algorithms used in argumentative approaches results in more complex source-code. This is highlighted that the argumentative approaches are written in over approximately 300 lines of Python while the non-argumentative approach is written within 100 lines. In conjunction with the above functional equivalence statement, we can interpret the non-argumentative approach as a refactoring of the argumentative approaches.

While argumentation offers, soundness, completeness, polynomial tractability and polynomial computational complexity, a non-argumentative approach remains sufficiently viable.

6.3 Practicability of Argumentation

Without knowledge of argumentation, one could define mappings between mathematical constraints and templated-explanations to create an interactive tool. We argue that argumentation is unnecessary in the development process. However, argumentation may bring subtle benefits that are not expressible in makespan scheduling.

Argumentation brings an alternative representation of conflicts or constraints. This representation is competitive in recommender systems [16], where chains of reasons can be constructed. In abstract argumentation, chain of reasons are constructed when a extension is fixed-point computable. This is true with admissible and complete extensions. However, with stability and conflict-freeness, as in our algorithms, the counter-arguments for a stable or a conflict-free extension does not construct chains of arguments. A potential solution to improve explanations, is to verify that mathematical constraints are admissible-modellable. However, at the time of writing, the notion of admissible and linear constraints have little connection.

We have shown in Chapter 4 that is it possible to translate new constraints to new frameworks. Each constraint and their explanations are loosely-coupled by an AAF. We have attempted to link efficiency and fixed user decisions by defining fixed-decision aware exchange properties. This link was inspired by intuitive interpretation of the problem. More generally, schedules can be subject to multi-objective optimisation. If we consider conflicts between objectives, rather than conflicts between possible schedules under a mathematical constraint, then we use can argumentation to compromise between objectives. We can infer an explanation from the chain of arguments. However, the maximum chain of arguments is restricted to the number of schedule properties. Given we have only three properties in makespan scheduling, these explanations do not offer much beyond a simple inspection of a schedule.

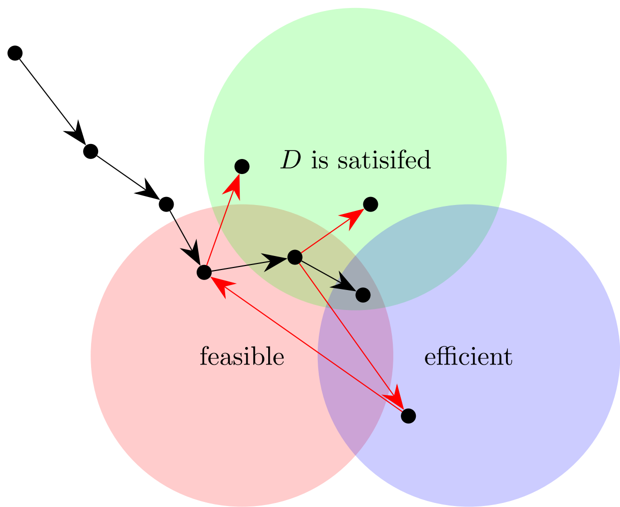


Figure 6.1: Venn diagram of makespan schedule properties with a local schedule optimisation path. Red arrows represent a schedule becoming unsatisfied of some property.

As illustrated in Figure 6.3, our tool can take a schedule, possibly empty, and iteratively improve on it. However, at certain intermediate schedules, we may not improve the schedule in trivial ways to satisfy schedule properties, possibly because of nurse preferences. This results in steps which contradict already-satisfied properties, as highlighted in red. Given an explanation constructed using argumentation, we can justify the compromise between schedule properties using preferences.

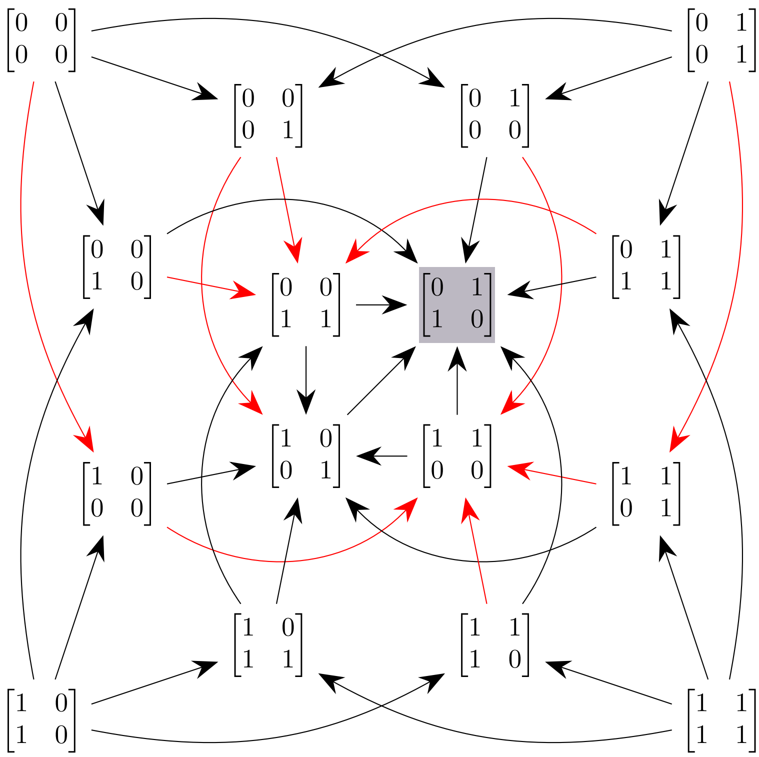


Figure 6.2: Complete makespan problem space for $m = n = 2$, $\mathbf{p} = [1, 1]$, $D^- = \{\langle 2, 1\rangle\}$ and $D^+ = \varnothing$. The shaded schedule is satisfies all properties.

It is possible that all possible improvements on a schedule may contradict already-satisfied properties. Consider figure 6.3, where we give a concrete problem. At schedule [[0 0] [0 1]], all improvements are marked as red. Suppose we have no preferences on schedules. In this case, we cannot justify making any improvements because we enforce all schedule properties to hold. There are better schedules, but they are inaccessible given our current schedule. In other words, we have reached an local optimal using explanations. This motivates to use global search, but as we stated before, this in intractable for arbitrary schedule spaces. Therefore, there is a clear trade-off between expressibility and computational tractability of explanations. Given the scope of this project, there is no conclusive method to tractably compute complete generalised explanations.

6.4 Discussion of Objectives

We complete the first objective (stated in section 1.2), that is to implement a tool satisfying the challenges mentioned (section 1.3). We revisit these challenges as follows:

1. **Trust:** We use both formal and practical verification methods to ensure our program is correct. We give proofs to our algorithms (section B) and a test framework (section 3.4).

2. **Accessibility:** From our survey, we have shown users are able to construct and manipulate problems and schedules to model scenarios.

3. **Applicability:** From out survey, we have shown users are apply to explanations to aid decision-making.

4. **Knowledge transfer:** In the tool, we provide both a cascade chart alongside an interactive explanation. The feedback from users suggested that explanations were relatively clear given a diagram. The extent in which the interface provides the best knowledge transfer is not known.

5. **Background:** We have been able to construct a functional tool with limited implementation details from the paper [5].

From the survey, we have shown that explanations allows better reasoning of makespan schedules. The survey responses suggests that the tool is better suited to comparing and understanding different scheduling cases, than understanding how to optimise a schedule. This contradicts the original premise where optimisation of a particular schedule is the focus. Given we implement the tool with focus on polynomial algorithms, it is difficult to interface users with clear explanations for optimality or near-optimality of schedules. This implies that the tool would be better suited to comparing schedules. As this is a form of replanning, the tool is useful in decision-maker, but lacks expressibility for general replanning.

We make progress the second objective, that is to extend theoretical or practical capabilities of argumentation for scheduling. This is evident from Chapter 4. From section 6.3, we have shown further effort is challenging. The future of using argumentation for scheduling is unknown at the time of writing and future work may transform the paradigm first mentioned in the paper [5] into a disruptive technology.

6.5 Future Work

- **Space of stability-modellable linear programs:** We have shown that makespan and interval scheduling are stability-modellable. An interesting direction would be to explore the space of linear programming problems that are stability-modellable, or more generally, extension-modellable. This is particularly difficult because of limitations with local extensions and non-polynomial verification [5].

- **Implementation of interval scheduling:** We have shown that interval scheduling feasibility is stability-modellable. Due to the time constraints of this project, we have not been able to integrate makespan and interval scheduling into a single graphical tool.

- **Nurse preferences:** It is known that previous literature ignores nurses preferences, or attempts to simplify individual nurses preferences as groups [17]. A future direction would be to explore the modelling individual nurses preferences using existing argumentation methods [18, 19] with a focus on explanations.

- **Web GUI:** The Python GUI is clear for users from an academic environment. But in comparison with commercial tools, their interface allows users to reschedule by intuitive drag-drop interactions. This was beyond the scope of an academic project.

6.6 Summary

Explaining makespan schedules using argumentation is practically possible but not practically suitable.

Appendix A

Notation

$[\![\,\cdot\,,\,\cdot\,]\!]$	inclusive integer space	
$\ominus$	Boolean tensor not operator	definition 15
$\owedge$	Boolean tensor and operator	definition 16
$\ovee$	Boolean tensor or operator	definition 17
$\rightsquigarrow, \rightsquigarrow_k$	attack relation	section 2.2.1
$\rightsquigarrow_\alpha$	interval schedule base feasibility	definition 28
$\rightsquigarrow_\beta$	interval schedule sequential feasibility	definition 29
$\rightsquigarrow_\gamma$	interval schedule start feasibility	definition 30
$\rightsquigarrow_\delta$	interval schedule finish feasibility	definition 31
$\rightsquigarrow_\varepsilon$	interval schedule negative decision feasibility	definition 32
$\rightsquigarrow_\zeta$	interval schedule positive decision feasibility	definition 33
$\rightsquigarrow_\eta$	interval schedule positive decision feasibility	definition 34
$\rightsquigarrow_D$	makespan fixed decision attack relation	definition 18
$\rightsquigarrow_F$	makespan feasibility attack relation	definition 19
$\rightsquigarrow_S$	makespan efficiency attack relation	definition 20
$\twoheadrightarrow$	tensor implementation of $\rightsquigarrow$	section 3.3.3
$\twoheadrightarrow_D$	tensor implementation of $\rightsquigarrow_D$	definition 21
$\twoheadrightarrow_F$	tensor implementation of $\rightsquigarrow_F$	definition 22
$\twoheadrightarrow_S$	tensor implementation of $\rightsquigarrow_S$	definition 23
$\prec$	lexicographical ordering on $Args$	appendix B
$Args$	set of arguments	section 2.2.1
D	user fixed decisions	section 2.1.2
D^-	user negative fixed decisions	section 2.1.2
D^+	user positive fixed decisions	section 2.1.2
E	extension	section 2.2.1
$\mathbf{f}$	finishing times	section 4.2
i, i', i_1, i_2	machine index	section 4.2
j, j', j_1, j_2	job index	section 2.1.1
$\mathcal{J}$	set of jobs	section 2.1.1
m	number of machines	section 2.1.1
$\mathcal{M}$	set of machines	section 2.1.1
n	number of jobs	section 2.1.1
$\mathbf{p}$	processing times	section 2.1.1
P, P_0, P_k	schedule property	section 4.1
Q_l	post-condition on line l	appendix B

$\mathbf{s}$	starting times	section 4.2
S	schedule	section 2.1.1
t, t', t_1, t_2	time index	section 4.2
T	number of timeslots	section 4.2
$\mathcal{T}$	set of time indices	section 4.2
$\mathbf{x}$	assignment matrix	definition 1

Appendix B

Stability Algorithm Proof

In this chapter, we reprint lemmas and theorems in section 3.3.4. We will define some notation to make the proofs more clear. Let $\prec$ be the lexicographical ordering on $Args$, such that $\langle i_1, j_1\rangle \prec \langle i_2, j_2\rangle$ iff $i_1 < i_2 \lor i_1 = i_2 \land j_1 < j_2$. Let Q_l be the post-condition of line l, which states a property about the current state of execution through the algorithm. We will use logical variables k and ℓ to differentiate from program variables i and j, in functional correctness proofs using Hoore logic [20].

B.1 Compute-Unattacked is correct

```
function Compute-Unattacked(x, ↠, ū)
    u ← ⊖ x
    for i ∈ M, j ∈ J do
        if x_{i,j} = 1 then
            u ← u ⩓ ⊖ ↠_{i,j}
        end if
    end for
    u ← u ⩓ ⊖ ū
    return u
end function
```

Proof. To show:

$$\begin{aligned}\forall k \in \mathcal{M}\ \forall \ell \in \mathcal{J}\ \bar{u}_{k,\ell} = 0 \implies \Big(u_{k,\ell} = 1 \iff \neg\exists k' \in \mathcal{M}\ \exists \ell' \in \mathcal{J}\\ x_{k,\ell} = 0\\ \land\ x_{k',\ell'} = 1\\ \land\ \twoheadrightarrow_{k',\ell',k,\ell} = 1\Big)\\ \land\ \bar{u}_{k,\ell} = 1 \implies u_{k,\ell} = 0\end{aligned}$$

Take arbitrary Boolean tensors $\mathbf{x}$, $\twoheadrightarrow$ and $\bar{\mathbf{u}}$, such that their dimensions are valid. The parameters are required to be well-defined for the operators to be well-defined.

$$Q_2 : \forall k \in \mathcal{M}\ \forall \ell \in \mathcal{J}\ u_{k,\ell} = 1 \iff x_{k,\ell} = 0$$

From the definition of $\ominus$, $u_{k,\ell} = 1 - x_{k,\ell}$. Because $\mathbf{u}$ and $\mathbf{x}$ are Boolean-valued, then $u_{k,\ell} = 1 \iff x_{k,\ell} = 0$. So Q_2 holds.

$$\begin{aligned} Q_3 : \forall k \in \mathcal{M}\ \forall \ell \in \mathcal{J}\ \langle k, \ell\rangle \prec \langle i, j\rangle \implies (u_{k,\ell} = 1 \iff& \neg\exists k' \in \mathcal{M}\ \exists \ell' \in \mathcal{J} \\ & x_{k,\ell} = 0 \\ & \wedge\ x_{k',\ell'} = 1 \\ & \wedge\ \twoheadrightarrow_{k',\ell',k,\ell} = 1 \\ \langle k, \ell\rangle \succeq \langle i, j\rangle \implies (u_{k,\ell} = 1 \iff& x_{k,\ell} = 0) \end{aligned}$$

Q_3 is a loop invariant, that holds from Q_2 because initially, $i = 1$ and $j = 1$, so there are no $\langle k, \ell\rangle \prec \langle i, j\rangle$.

$$Q_4 : Q_3 \wedge x_{i,j} = 1$$

Q_4 holds from Q_3 because of the if condition.

$$\begin{aligned} Q_5 : \forall k \in \mathcal{M}\ \forall \ell \in \mathcal{J}\ \langle k, \ell\rangle \preceq \langle i, j\rangle \implies (u_{k,\ell} = 1 \iff& \neg\exists k' \in \mathcal{M}\ \exists \ell' \in \mathcal{J} \\ & x_{k,\ell} = 0 \\ & \wedge\ x_{k',\ell'} = 1 \\ & \wedge\ \twoheadrightarrow_{k',\ell',k,\ell} = 1 \\ \langle k, \ell\rangle \succ \langle i, j\rangle \implies (u_{k,\ell} = 1 \iff& x_{k,\ell} = 0) \end{aligned}$$

At line 5, any argument $\langle k, \ell\rangle$ that is attacked by $\langle i, j\rangle$ is marked as attacked, so $u_{i,j} = 0$. The attack from $\langle i, j\rangle$ is relevant because from Q_4, we know that $\langle i, j\rangle \in E$. But at the next iteration, values i and j are overwritten so we retain $\exists k', \ell'\ \langle k', \ell'\rangle \twoheadrightarrow \langle k, \ell\rangle$. So Q_5 holds from Q_4.

We also need to prove that Q_5 holds from Q_3 if the if condition fails. Q_5 holds because if $x_{i,j} = 0$, then there are no attacks from $\langle i, j\rangle \in E$.

$$\begin{aligned} Q_8 : \forall k \in \mathcal{M}\ \forall \ell \in \mathcal{J}\ u_{k,\ell} = 1 \iff& \neg\exists k' \in \mathcal{M}\ \exists \ell' \in \mathcal{J} \\ & x_{k,\ell} = 0 \\ & \wedge\ x_{k',\ell'} = 1 \\ & \wedge\ \twoheadrightarrow_{k',\ell',k,\ell} = 1 \end{aligned}$$

Q_8 holds from Q_3 because at the end of the loop, for any $\langle k, \ell\rangle \prec \langle i, j\rangle$. $\mathbf{u}$ will be filtered by $\bar{\mathbf{u}}$, so $\forall k \in \mathcal{M}\ \forall \ell \in \mathcal{J}\ \bar{u}_{k,\ell} = 1 \implies u_{k,\ell} = 0$. Therefore, the function is correct. $\square$

B.2 Compute-Partial-Conflicts is correct

1: **function** COMPUTE-PARTIAL-CONFLICTS($\mathbf{x}$, $\twoheadrightarrow_{i,j}$, $\bar{c}_{i,j}$)
2: $\quad c_{i,j} \leftarrow \mathbf{0}^{m \times n}$
3: $\quad$ **if** $x_{i,j} = 1$ **then**
4: $\quad\quad c_{i,j} \leftarrow \mathbf{x} \owedge \twoheadrightarrow_{i,j}$
5: $\quad$ **end if**
6: $\quad c_{i,j} \leftarrow c_{i,j} \owedge \textcircled{$\neg$}\, \bar{c}_{i,j}$
7: $\quad$ **return** $c_{i,j}$
8: **end function**

Proof. To show:

$$\begin{aligned} \forall k \in \mathcal{M}\ \forall \ell \in \mathcal{J}\ c_{i,j,k,\ell} = 1 \iff & x_{i,j} = 1 \\ & \wedge\ x_{k,\ell} = 1 \\ & \wedge\ \twoheadrightarrow_{i,j,k,\ell} = 1 \\ & \wedge\ \bar{c}_{i,j,k,\ell} = 0 \end{aligned}$$

At line 2, $c_{i,j,k,l} = 0$ by assignment. At line 3, if $x_{i,j} = 1$, then $\langle i, j\rangle \in E$ may be an attacker. At line 4, there is a conflict $\langle i, j\rangle \rightsquigarrow \langle k, \ell\rangle$ when $x_{i,j} = 1$, $x_{k,\ell} = 1$ and $\twoheadrightarrow_{i,j,k,\ell} = 1$. If so, $c_{i,j,k,\ell} = 1$. At line 6, $\bar{c}_{i,j,k,\ell} = 1 \implies c_{i,j,k,\ell} = 0$. Therefore, the function is correct.

□

B.3 Explain-Stability is correct

function EXPLAIN-STABILITY($\mathbf{x}$, $\twoheadrightarrow$, $\bar{\mathbf{u}}$, $\bar{\mathbf{c}}$)
$\quad \mathbf{u} \leftarrow$ COMPUTE-UNATTACKED($\mathbf{x}$, $\twoheadrightarrow$, $\bar{\mathbf{u}}$)
$\quad$ **for** $i \in \mathcal{M}, j \in \mathcal{J}$ **do**
$\quad\quad c_{i,j} \leftarrow$ COMPUTE-PARTIAL-CONFLICTS($\mathbf{x}$, $\twoheadrightarrow_{i,j}$, $\bar{c}_{i,j}$)
$\quad$ **end for**
$\quad$ **return** $\langle \mathbf{u}, \mathbf{c}\rangle$
end function

Proof. Assume EXPLAIN-STABILITY($\mathbf{x}$, $\twoheadrightarrow$, $\bar{\mathbf{u}}$, $\bar{\mathbf{c}}$) $= \langle \mathbf{u}, \mathbf{c}\rangle$. From inspection of the algorithm, $\mathbf{u}$ and each sub-matrix of $\mathbf{c_{i,j}}$ is assigned exactly once in EXPLAIN-STABILITY, so Q_2 and Q_5 holds, meaning we can use Lemmas 1 and 2.

$$Q_2 : \text{COMPUTE-UNATTACKED}(\mathbf{x}, \twoheadrightarrow, \bar{\mathbf{u}}) = \mathbf{u}$$

$$Q_5 : \forall k \in \mathcal{M}\ \forall \ell \in \mathcal{J}\ \text{COMPUTE-PARTIAL-CONFLICTS}(\mathbf{x}, \twoheadrightarrow_{k,\ell}, \bar{c}_{k,\ell}) = c_{k,\ell}$$

To show:

$$\begin{aligned}\alpha : \forall \langle k,\ell\rangle \in \mathit{Args} \setminus E \\ \bar{u}_{k,\ell} = 0 \implies \big(\exists \langle k',\ell'\rangle \in E\ \langle k',\ell'\rangle \rightsquigarrow \langle k,\ell\rangle \iff u_{k,\ell} = 0\big) \\ \wedge\ \bar{u}_{k,\ell} = 1 \implies u_{k,\ell} = 0\end{aligned}$$

1. Take arbitrary $\langle k,\ell\rangle \in \mathit{Args} \setminus E$
2. $x_{k,\ell} = 0$ — 1
3. Assume $\bar{u}_{k,\ell} = 0$ — assumption
    4. Assume $\exists \langle k',\ell'\rangle \in E\ \langle k',\ell'\rangle \rightsquigarrow \langle k,\ell\rangle$ — assumption
        5. Take arbitrary k,ℓ such that $\langle k',\ell'\rangle \rightsquigarrow \langle k,\ell\rangle$
        6. $x_{k',\ell'} = 1$ — 4
        7. $\twoheadrightarrow_{k',\ell',k,\ell}$ — 5
        8. $u_{k',\ell',k,\ell} = 0$ — 2, 3, 5, 7, Lemma 1
    9. Assume $u_{k',\ell',k,\ell} = 0$ — assumption
        10. $\exists k' \in \mathcal{M}\ \exists \ell' \in \mathcal{J} x_{k',\ell'} = 1 \wedge \twoheadrightarrow_{k',\ell',k,\ell} = 1$ — 9, Lemma 1
        11. Take arbitrary k',ℓ' such that $x_{k',\ell'} = 1 \wedge \twoheadrightarrow_{k',\ell',k,\ell} = 1$
            12. $\langle k',\ell'\rangle \in E$ — 11
            13. $\langle k',\ell'\rangle \rightsquigarrow \langle k,\ell\rangle$ — 11
        14. $\exists \langle k',\ell'\rangle \in E\ \langle k',\ell'\rangle \rightsquigarrow \langle k,\ell\rangle$ — 11, 12, 13
    15. $\exists \langle k',\ell'\rangle \in E\ \langle k',\ell'\rangle \rightsquigarrow \langle k,\ell\rangle \iff u_{k',\ell',k,\ell} = 0$ — 4, 8, 9, 14
16. $\bar{u}_{k,\ell} = 0 \implies \exists \langle k',\ell'\rangle \in E\ \langle k',\ell'\rangle \rightsquigarrow \langle k,\ell\rangle \iff u_{k',\ell',k,\ell} = 0$ — 3, 14
17. Assume $\bar{u}_{k,\ell} = 1$ — assumption
    18. $u_{k,\ell} = 0$ — 17, Lemma 1
19. $\bar{u}_{k,\ell} = 1 \implies u_{k,\ell} = 0$ — 17, 18
20. α — 16, 19

To show:

$$\begin{aligned}\beta : \forall \langle k,\ell\rangle, \langle k',\ell'\rangle \in E \\ \bar{c}_{k,\ell,k',\ell'} = 0 \implies \big(\langle k,\ell\rangle \rightsquigarrow \langle k',\ell'\rangle \iff c_{k,\ell,k',\ell'} = 1\big) \\ \wedge\ \bar{c}_{k,\ell,k',\ell'} = 1 \implies c_{k,\ell,k',\ell'} = 1\end{aligned}$$

1. Take arbitrary $\langle k,\ell\rangle, \langle k',\ell'\rangle \in E$
2. $\langle k,\ell\rangle \in E$ — 1

3. $\langle k', \ell' \rangle \in E$ 1

4. Assume $\bar{c}_{k,\ell,k',\ell'} = 0$ assumption

   5. Assume $\langle k, \ell \rangle \rightsquigarrow \langle k', \ell' \rangle$ assumption

      6. $\twoheadrightarrow_{k,\ell,k',\ell'}$ 5

      7. $c_{k,\ell,k',\ell'}$ 2, 3, 4, 6, Lemma 2

   8. Assume $c_{k,\ell,k',\ell'}$ assumption

      9. $\twoheadrightarrow_{k,\ell,k',\ell'}$ 8, Lemma 2

      10. $\langle k, \ell \rangle \rightsquigarrow \langle k', \ell' \rangle$ 9

   11. $\langle k, \ell \rangle \rightsquigarrow \langle k', \ell' \rangle \iff c_{k,\ell,k',\ell'} = 1$ 5, 7, 8, 10

12. $\bar{c}_{k,\ell,k',\ell'} = 0 \implies \big(\langle k, \ell \rangle \rightsquigarrow \langle k', \ell' \rangle \iff c_{k,\ell,k',\ell'} = 1\big)$ 4, 11

13. Assume $\bar{c}_{k,\ell,k',\ell'} = 1$ assumption

   14. $c_{k,\ell,k',\ell'} = 1$ 13, Lemma 2

15. $\bar{c}_{k,\ell,k',\ell'} = 1 \implies \big(\langle k, \ell \rangle \rightsquigarrow \langle k', \ell' \rangle \iff c_{k,\ell,k',\ell'} = 1\big)$ 13, 14

16. β 12, 15

We have shown that α and β holds, which intuitively means that $\mathbf{u}$ and $\mathbf{c}$ are computed correctly, respectively. $\square$

Appendix C

User Guide

The user guide is a tutorial for non-technical users to learn how to use the tool. The tool has been extensively tested on Linux, the tool has full functionality on Linux and basic functionality on Windows.

C.1 Installation

The tool has the following required dependencies:

- Python 3
- Tkinter
- NumPy
- Matplotlib
- Pillow

Tkinter is the default GUI package for Python and Matplotlib depends on NumPy so Tkinter and Matplotlib do not need to be installed explicitly. The tool has the following optional dependencies:

- Pyomo
- An optimiser, either CPLEX or GLPK

The optional dependencies are used to optimise schedules, these are not strictly required as users can input their own schedules. GLPK is recommended because of its licence and open-source development. The repository is found at `gitlab.doc.ic.ac.uk/mtl115/aes`. Please refer to the package's websites for troubleshooting. Alternatively, contact the author for assistance.

C.1.1 Linux Installation

The tool was tested on Ubuntu 18.04.1. Python is pre-installed so the following packages can be installed as below:

```
apt install python3-pip
apt install python-glpk
apt install glpk-utils
pip3 install matplotlib
pip3 install pillow
pip3 install pyomo
```

C.1.2 Windows Installation

The tool was tested on Windows 10. First, download Python from the official website, then setup the path environment variable so `python` can be executed on the command prompt. Afterwards, install the required dependencies as follows:

```
python -m pip install matplotlib
python -m pip install pillow
python -m pip install pyomo
```

If you install GLPK, ensure the path variable is set correctly so `pyomo` can access GLPK.

C.2 Usage

C.2.1 Getting Started

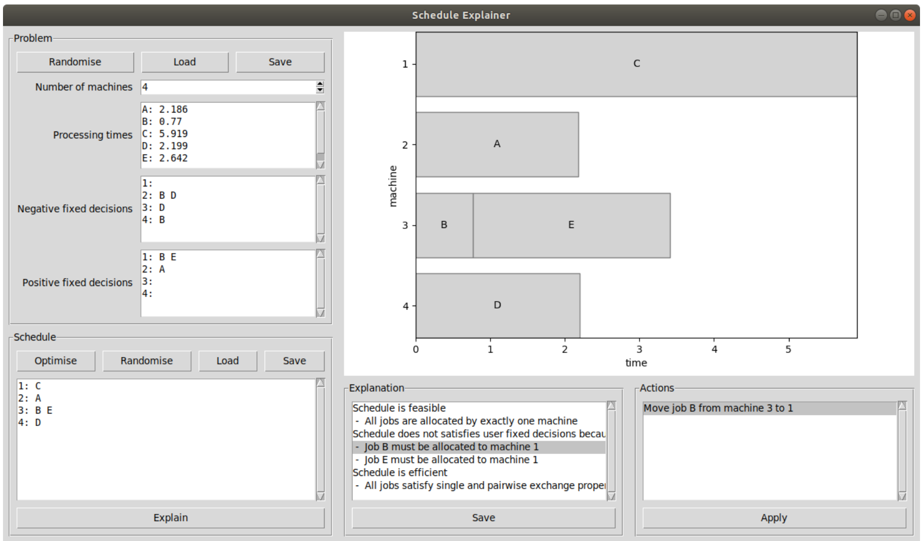


Figure C.1: Tool GUI

To start the tool, run `python3 main.py -g` on Linux or `python main.py -g` on Windows in the `src` directory supplied in the repository.

The makespan problem consists of the number of machines and job processing times. The tutorial will use a hospital setting, where nurses and patients are represented as machines and jobs respectively. Consider the following example where there are two nurses, Alice and Bob. and two patients, Charlie and Dave. Charlie's and Dave's appointment takes 15 and 10 minutes respectively. To enter the example in the tool, nurses and patients are indexed. Hence, A represents Alice, B represents Bob for nurses and 1 represents Charlie and 2 represents Dave for patients. Numbers are used to index machines and letters are used to index jobs. The problem is to minimise the total completion time, which intuitively is the longest time any nurse has to work. Machines must be integer-indexed and jobs must be alphabetically-indexed.

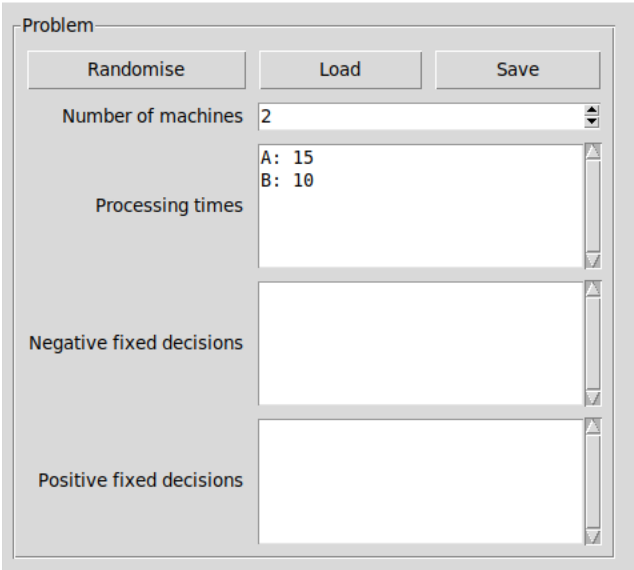


Figure C.2: Example problem input

Each line in the processing time textbox represents one job. The first line can be interpreted as: job A has processing time of 15 units, with following lines having similar interpretations. Negative fixed decisions represent jobs that cannot be assigned to machines. Positive fixed decisions represents jobs that much be assigned to a machine. Note that for all multi-line inputs, each line ends with a new line character.

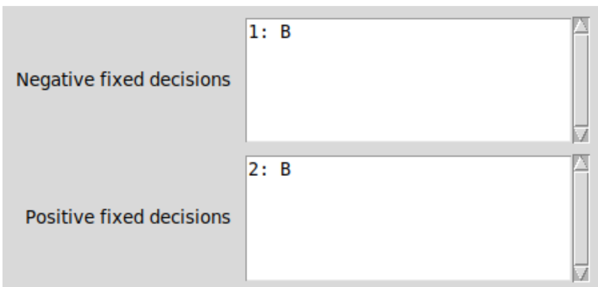


Figure C.3: Example fixed decisions input

Each line in each fixed decisions textbox represents one decision. The first line of negative fixed decisions can be interpreted as: machine 1 cannot be allocated to job B. The first line of positive fixed decisions can be interpreted as: machine 2 cannot be allocated to job B. In context of the example, this means Alice cannot be with Dave and Bob must be with Dave.

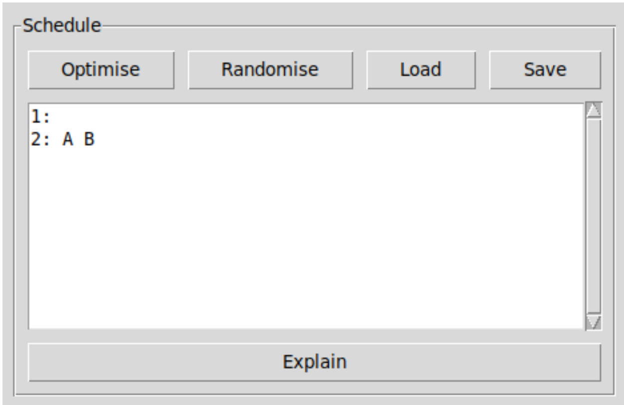


Figure C.4: Example schedule input

After defining the makespan problem, enter the above schedule. The schedule can be interpreted as: machine 1 has no allocated jobs; machine 2 have two allocated jobs, A and B. The `Optimise` button finds the optimal schedule using a solver, which is by default GLPK. To specify a solver, starting the tool with `python3 main.py -g -S SOLVER_NAME` where `SOLVER_NAME` is GLPK or CPLEX, for instance. Note that for large problems, optimisation may take a long time, so a solver time limit can be enforced by starting the tool with `python3 main.py -g -t TIME_LIMIT` where `TIME_LIMIT` is in seconds. The `Randomize` button generates some feasible schedule, which may violate fixed decisions. To explain the schedule, click the `Explain` button.

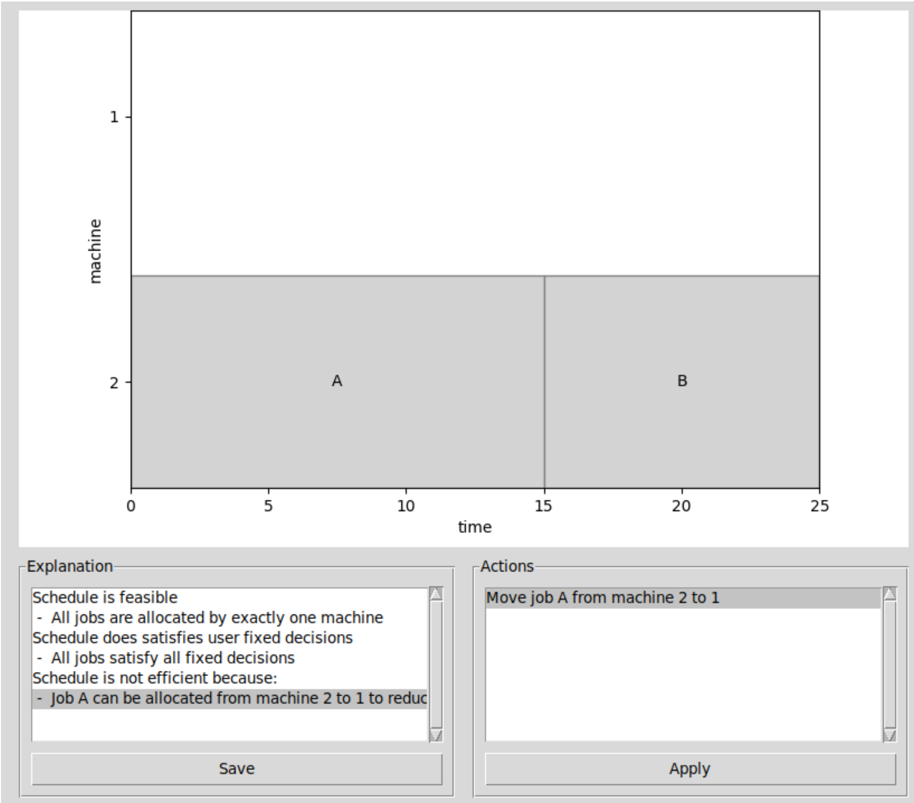


Figure C.5: Example explanation output

The explanation reasons on three concepts: feasibility, satisfaction of fixed decisions and efficiency. Feasibility ensures that each job is allocated once. Satisfaction of fixed decisions ensures the schedule does not violate negative and positive fixed decisions specified in the problem. Efficiency regards suggestions to improve the total completion time. To improve the schedule, select a line in the explanation listbox to address, then select a line in the actions listbox. An line of explanation may have many different approaches to address the problem or schedule. Click on the `Apply` button to improve the schedule.

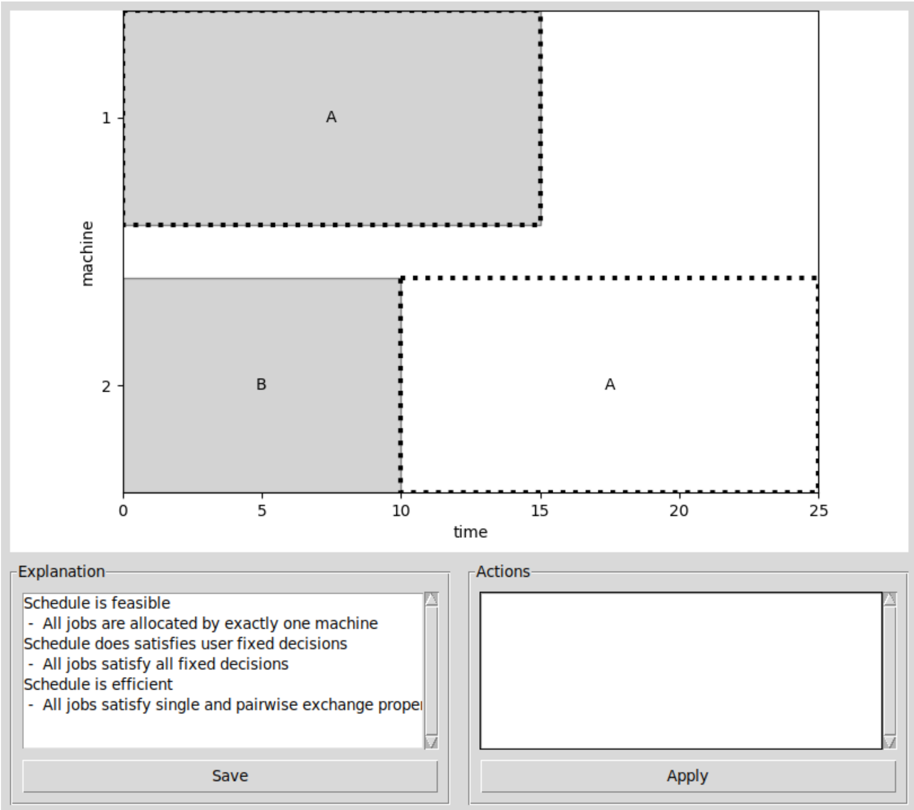


Figure C.6: Example explanation output

The example schedule only required one action to make the schedule efficient. However, many iterative actions may be required to reach an efficiency schedule. No further actions show that the schedule is feasible, satisfies fixed decisions and is efficient. The dot-highlighted boxes in the cascade chart illustrate newly and removed allocations compared to before the applying the action.

C.2.2 Command Line Examples

Help Command

```
> python3 main.py -h
usage: main.py [-h] [-g] [-e] [-p PROBLEM | -r [M]]
    [-O | -s SCHEDULE | -R] [-o OUTPUT] [--partial | --naive]
    [-t TIME_LIMIT] [-S SOLVER_NAME]

Explains makespan schedules using abstract argumentation
frameworks

optional arguments:
-h, --help            show this help message and exit
-g, --graphical       displays graphical user interface
-e, --explain         generate explanation
```

```
-p PROBLEM, --problem PROBLEM
-r [M], --random_problem [M]
    creates random problem with jobs and fixed decisions
    where M is the number of machines
-O, --optimise        uses SOLVER_NAME to find most efficient
schedule
-s SCHEDULE, --schedule SCHEDULE
-R, --random_schedule
-o OUTPUT, --output OUTPUT
    output filename for selected problem, schedule or
    explanation
--partial             use partial framework construction to
favour memory over CPU
--naive               do not use argumentation
-f, --fixed_decision_aware
    force efficiency to respect fixed decisions
-t TIME_LIMIT, --time_limit TIME_LIMIT
    maximum time for optimisation in seconds, use negative
    time_limit for infinite limit, default is unlimited
time
-S SOLVER_NAME, --solver SOLVER_NAME
    optimisation solver for schedule, default is 'glpk'
```

Random problem

```
> python3 main.py -r
4;
A: 1.822
B: 2.994
C: 6.444
D: 2.578
E: 2.386
;
1: B
2: A
3: D
4: B
;
1:
2: B
3: C
4:
```

Formally, this represents $m = 4$, $n = 5$, $\mathbf{p} = \begin{bmatrix} 1.822 & 2.994 & 6.444 & 2.578 & 2.386 \end{bmatrix}^T$, $D^- = \{\langle 1, 2\rangle, \langle 2, 1\rangle, \langle 3, 4\rangle, \langle 4, 2\rangle\}$ and $D^+ = \{\langle 2, 2\rangle, \langle 3, 3\rangle\}$.

Random schedule given previous problem

```
> python3 main.py -p example.problem -R
1: A
2: B C E
```

```
3: D
4:
```

Formally, this represents $\mathbf{x} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

Explantation given previous problem and schedule

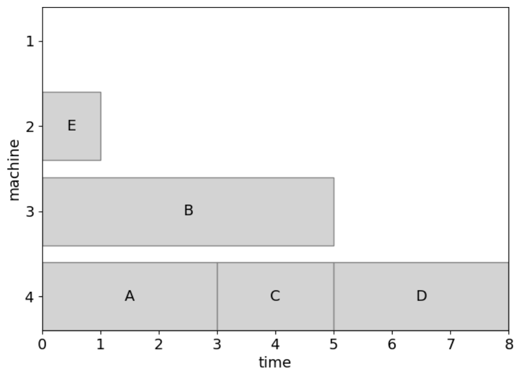
```
> python3 main.py -p example.problem -s example.schedule -e
Schedule is feasible
-  All jobs are allocated by exactly one machine
Schedule does not satisfies user fixed decisions because:
-  Job C must be allocated to machine 3
-  Job D must not be allocated to machine 3
Schedule is not efficient because:
-  Job C can be allocated from machine 2 to 4 to reduce by 5.38
-  Jobs C and D can be swapped with machines 2 and 3 to reduce by 3.87
-  Job C can be allocated from machine 2 to 1 to reduce by 3.56
-  Job C can be allocated from machine 2 to 3 to reduce by 2.8
-  Job E can be allocated from machine 2 to 1 to reduce by 2.39
-  Job E can be allocated from machine 2 to 3 to reduce by 2.39
-  Job E can be allocated from machine 2 to 4 to reduce by 2.39
```

C.3 Known Limitations

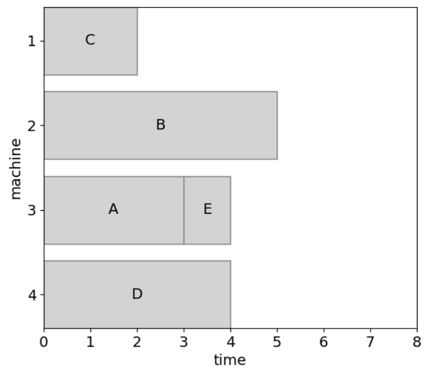
- Holding down space for a button that requires significant computation results in permanent depressed visual of the button. This is a issue with Tkinter.
- Indices must represent the full space of machines and jobs. For example, a job X in a schedule implies that all indices in $A, ..., X$ represents jobs.

Makespan Schedule Explanation Questionnaire

This questionnaire should take less than ten minutes. This will judge the general ability to understand and explain makespan schedule. Makespan schedules consist of m machines and n jobs. Every job is assigned to only one machine for the schedule to be feasible. Each job has a processing time. The objective is to minimise the longest collective processing time. Machines and jobs are denoted by integers $1, 2, 3, ...$ and by letters $A, B, C, ...$, respectively. A schedule is optimal when the longest collective processing time cannot be earlier.



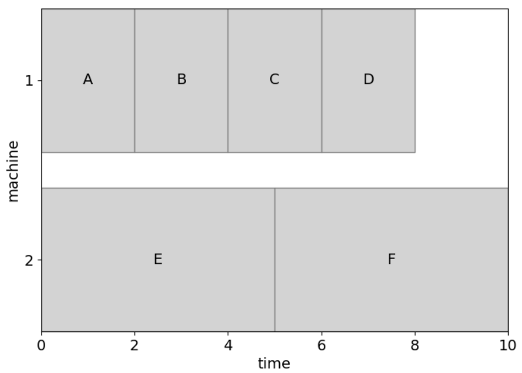
Schedule is not optimal because jobs A, C, D can be moved to machine 1.



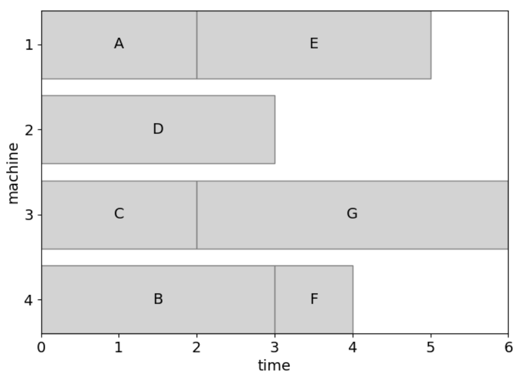
Schedule is optimal because no assignment can be improved.

Aim to spend at most one minute for each question. Some questions are difficult. Write your answers in the boxes.

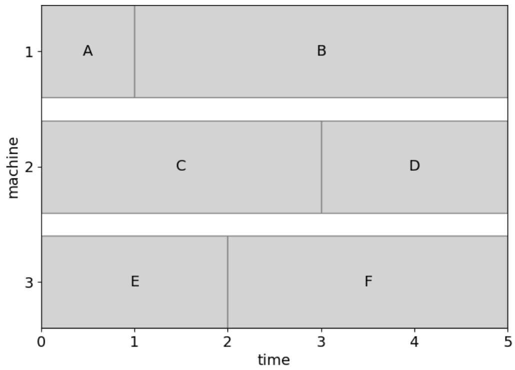
1. This schedule is not optimal. Suggest steps required to optimise this schedule.



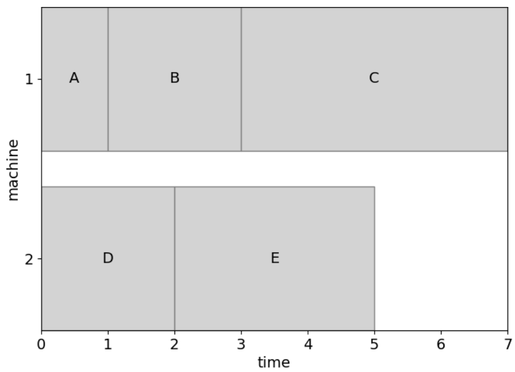
2. Is this schedule optimal? If not, suggest how it could be improved immediately.



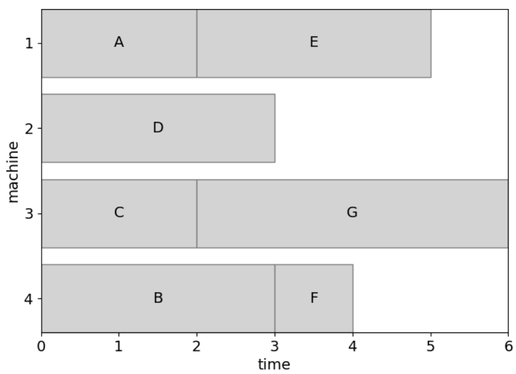
3. This schedule is optimal. Job A cannot be assigned to machine 1 or 2 anymore. How could the schedule be modified to agree with this constraint?



4. Either jobs C and D, or jobs B and E must be assigned to the same machine. Which constraint is results in a better schedule, and why?



5. A new job H of 5 time units needs to be scheduled. How could the schedule be modified to best accommodate this job?



Thank you for your time. Please send your answers to `myles.lee15@imperial.ac.uk`.

Appendix E

Survey Responses

E.1 No Explanations

Respondent	Time Taken
1	00:12:17
2	00:20:24
3	00:10:51
4	00:22:40
5	00:15:11
6	00:15:18
7	01:48:28
8	00:13:31
9	00:14:40
10	00:54:44

Question	Respondent	Answer
1	1	Switch C and D to machine 2 and F to machine 1.
	2	Give job F to machine 1 and give jobs C and D to machine 2
	3	CD and F be swapped
	4	Machine 1: A,B,E; Machine 2: C,D,F
	5	Move F to machine 1 and C and D to machine 2
	6	Jobs F should be given to machine 1 and C and D should be given to machine 2
	7	move E up and CD down
	8	Move A, B to E.
	9	1. E,A,B; 2. F,C,D
	10	Moving job F to machine 1, and jobs C and D to machine 2 will optimise the schedule.
2	1	Switch G to machine 4 and B to machine 3.
	2	Give job C to machine 2
	3	C be placed in machine 2
	4	Machine 1: A,E; Machine 2: D,C; Machine 3: G; Machine 4: B,F
	5	Move C to machine 2

	6	Jobs G and D should be swapped, schedule is not currently optimal
	7	1 - G; 2 - D, B; 3 - E, C; 4 -F, A
	8	A,C and F work together.
	9	move C to 2
	10	No this isn't optimal, having machine 2 handle jobs C and D, and machine 3 only handle job G would be an improvement.
3	1	Move A and B to machine 3 and move E and F to machine 1.
	2	Do all the shortest jobs first?
	3	EF and AB be swapped
	4	Machine 1: E,F; Machine 2: C,D; Machine 3: A,B
	5	Assign it to machine 3 along with job B, move E and F to machine 1
	6	Swap E and F with A and B
	7	1 - B, C; 2- D, E; 3- F, A
	8	Job A replace E
	9	swap machine 1's jobs with machine 3
	10	Swap the jobs for machines 1 and 3.
4	1	C and D, as this combination will add to a total time of 6, allowing A, B and E to also be added up to 6, thus creating the ideal possible schedule.
	2	B and E constraint is better as it results in less time total for the machine that has to do those jobs compared to if a machine had to do both C and D
	3	B and E as it means that the time split will be 6/6
	4	Machine 1: A,B,E; Machine 2: C,D
	5	Both at the same time work well. A B and E on machine 1 and C and D on machine 2. That results in a an optimum 6 time overall.
	6	If jobs C and D are assigned together then the total time will be shorter than if B and E are assigned together. So C and D result in a better schedule
	7	1 - C, D; 2 - A, B, E
	8	Put Job B with machine 2 is better.
	9	theyre the same
	10	Either constraint should result in a better schedule. We pick the first constraint and move C to machine 2, and then move E to machine 1 (so now the second constraint is also satisfied).
5	1	Switch D to machine 4 and F to machine 2. H can then be added to machine 2.
	2	Move job E to machine 2 and give job H to machine 1
	3	1 - AEF; 2 - (5 unit job); 3 - GC; 4 - BD
	4	Machine 1: A,E; Machine 2: D,B; Machine 3: C,G; Machine 4: H,F
	5	Move B to machine 2 and add H to machine 4
	6	A and D should be given to machine 1. H and F should be given to machine 2. C and G should be given to machine 3. B and E should be given to machine 4. The total time then remains unchanged at 6
	7	1 - A, F; 2 - B. E; 3 - G, C; 4 - H, D
	8	Move B machine 2 and new job in machine 4
	9	1. H; 2. G, A; 3. E, C, F; 4. D, B
	10	We will put job D in machine 3 and move job F to machine 2, and then also schedule job H to machine 2. This should keep the total time the same.

E.2 Simple Explanations

Respondent	Time Taken
1	00:09:41
2	00:01:13
3	00:03:16
4	00:06:25
5	00:08:06
6	00:11:41
7	00:03:25
8	00:09:22
9	00:05:26
10	00:09:14
11	00:12:02
12	00:11:02

Question	Respondent	Answer
1	1	Move job F to machine 1. move jobs C and D to machine 2.
	2	Move E to 1 and A, B to 2
	3	Move C and D to 2. Move F to 1.
	4	move F to 1; move C and D to 2
	5	Move C and D to 2 and move F to 1
	6	F to 1. C & D to 2.
	7	Move F to 1. Move C and D to 2.
	8	1: A, B, E 2: C, D, F
	9	move F to 1; move C and D to 2
	10	Move F to start at machine 1, and move job C, D to machine 2 after E
	11	A, B, and F to machine 1; C, D, and E to machine 2
	12	Move F to machine 1, move C and D to machine 2
2	1	Not optimal. Could be improved by moving job C to machine 2.
	2	Move C to 2
	3	
	4	move C to machine 2
	5	No, move C to 2
	6	C to 2. Total time would be 5h. I do not think this can be reduced further.
	7	No, because makespan can still be reduce by, e.g., moving Job C to machine 2.
	8	Move C to machine 2
	9	move C to D
	10	Jobs G and D can be swapped
	11	No; Jobs D and G can be swapped with machines 2 and 3 to reduce by 1.0
	12	Swap G and D from machine 3 and 4

3	1	Move job A to machine 3.
	2	...and then do nothing.
	3	
	4	
	5	Switch all the jobs of machine 1 and 3
	6	A & B to 3. C & D to 1. E & F to 2. (permutation of the above schedule)
	7	
	8	Swap the jobs of machines 1 and 3
	9	and move B to 3 next; move E and F to 1
	10	move job a to machine 3 then move job b to machine 3 as well, move E, F to machine 1
	11	Move job A to machine 3 and then job B to 3 and jobs E,F to machine 1
	12	Move A to machine 3 and move B to machine 3, move E and F to machine 1
4	1	Don't understand the question.
	2	
	3	
	4	either
	5	Both result in the same optimality
	6	As hinted above, we can create a schedule lasting 6h (which I think is optimal) that respects both constraints.
	7	
	8	Since C+D=A+B+E, the schedule can be optimal when both constraints are satisfied
	9	A with B and E too; C and D together
	10	C and D
	11	Both of them simultaneously
	12	Doesn't matter
5	1	Assign job H to machine 4, move job B to machine 2.
	2	
	3	
	4	H on 2; D on 4; F on 2
	5	Switch D and F
	6	1: A & E; 2: B & D; 3: C & G; 4: F & H; Total time = 6h
	7	
	8	
	9	swap D and F
	10	swap D and F and allocate H to machine 2
	11	Job F to machine 1; Job D to machine 4; and new job H to machine 2
	12	swap job D and F from machine 2 and 4

E.3 Interactive Explanations

Respondent	Time Taken
1	00:14:15
2	over a day
3	00:15:45
4	00:31:28
5	11:44:21
6	00:19:03
7	00:11:45

Question	Respondent	Answer
1	1	Swap c and d with f
	2	Machine 1->ABE; Machine 2 ->CDF
	3	move F to machine 1, Move C&D to machine 2
	4	Take F to machine 1, then bring C and D to machine 2
	5	Move job F to machine 1, move jobs A and B to machine B. (doesn't have to be those jobs specifically, just move a large job to 1 and two small jobs to 2)
	6	Both machines could be done an hour sooner. Put EAB on 1, FCD on 2.
	7	Move C and D to 2 and F to 1.
2	1	Not optimal. Move Job f to machine 2 then swap g and b from machines 3 and 4
	2	This schedule is not optimal. Optimal makespan is 5
	3	move C to machine 2
	4	Job C can be allocated from machine 3 to 2 to reduce by 1.0
	5	No, any of the suggested improvements can be done to make it optimal (since each one results in no machine taking more than 5 seconds and there is less than a total of 4 seconds ”spare” among the 4 machines, meaning it cannot be optimised further)
	6	Move C to machine 2.
	7	All of the above suggested changes are true. The schedule is not optimal.

3	1	Job a moves to machine 3. Then move job e from 3 to 1
	2	This is makespan optimal.
	3	move job A to machine 3 and then move E to 1 after 1 unit of time
	4	Move job A to machine 3 then re allocate job E
	5	Move Job A and B to machine 3, move job E and F to machine 1.
	6	Move A to 3, and E to 1.
	7	Also move B to 3 and then move E and F to 1.
4	1	B and e together as they have a total length of 5 which is shorter than c and d total length 6
	2	There is only one solutions fulfilling that constraint.
	3	C and D together
	4	Doesn't matter. They've both reached their optimal level.
	5	Doesn't matter. The example has shown that an optimal schedule can be made that follows both constraints, so the constraints don't worsen anything.
	6	Does not matter
	7	It doesn't matter.
5	1	Add h to machine 4 then move b from 4 to 2 making the longest time 6
	2	Swap H and G to reduce makespan by 1. Add E to machine 4, makespan is still7. Add 5 units of time to machine 1. Makespan is 7 (i think this is minimised)
	3	move job B to machine 2 and add H to machine 4
	4	Jobs D and F can be swapped with machines 2 and 4 to reduce by 2.0
	5	Move job H to machine 4, move job B to machine 2
	6	Put b on 2 and H on 5.
	7	Add H to 4 and move B to 2.

Bibliography

[1] E. K. Burke, P. D. Causmaecker, G. V. Berghe, and H. V. Landeghem, *The State of the Art of Nurse Rostering*, vol. 7. 2004.

[2] A. Rais and A. Viana, "Operations research in healthcare: a survey," *International Transactions in Operational Research*, vol. 18, no. 1, pp. 1–31, 2011.

[3] J. L. Gearhart, K. L. Adair, R. J. Detry, J. D. Durfee, K. A. Jones, and N. Martin, *Comparison of open-source linear programming solvers.* 2013.

[4] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems.* Springer, 2012.

[5] K. Cyras, D. Letsios, R. Misener, and F. Toni, "Argumentation for explainable scheduling," in *Thirty-Third AAAI Conference on Artificial Intelligence*, (Honolulu, Hawaii, USA), AAAI Press, 2019.

[6] S. Sohrabi, J. A. Baier, and S. A. McIlraith, *Preferred Explanations: Theory and Generation via Planning.* AAAI Press, 2011.

[7] P. Brucker, *Scheduling Algorithms.* Springer, 2007.

[8] A. W. Kolen, J. K. Lenstra, C. H. Papadimitriou, and F. C. Spieksma, *Interval scheduling: A survey*, vol. 54. Wiley Online Library, 2007.

[9] D. Walton, *Argumentation theory: A very short introduction.* Springer, 2009.

[10] P. M. Dung, "On the acceptability of arguments and its fundamental role in nonmonotonic reasoning and logic programming," pp. 852–859, 1993.

[11] M. Fox, D. Long, and D. Magazzeni, "Explainable planning," *CoRR*, vol. abs/1709.10256, 2017.

[12] E. W. Weisstein, "Adjacency matrix." `mathworld.wolfram.com/AdjacencyMatrix.html`, 2019. Accessed on 09/05/2019.

[13] "Free online appointment scheduling calender software." `https://www.setmore.com`, 2019. Accessed on 25/01/2019.

[14] M. Pinedo *et al.*, "Scheduling system." `http://web-static.stern.nyu.edu/om/software/lekin`, 2010. Accessed on 25/01/2019.

[15] F. Cerutti, S. Gaggl, M. Thimm, and J. Wallner, “Foundations of implementations for formal argumentation,” *The IfCoLog Journal of Logics and their Applications*, vol. 4, pp. 2623–2705, 10 2017.

[16] A. Rago, O. Cocarascu, and F. Toni, “Argumentation-based recommendations: Fantastic explanations and how to find them,” in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pp. 1949–1955, International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[17] M. L. D. Grano, D. J. Medeiros, and D. Eitel, “Accommodating individual preferences in nurse scheduling via auctions and optimization,” *Health Care Management Science*, vol. 12, pp. 228–242, oct 2008.

[18] L. Amgoud and C. Cayrol, “On the acceptability of arguments in preference-based argumentation,” in *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, (San Francisco, CA, USA), pp. 1–7, Morgan Kaufmann Publishers Inc., 1998.

[19] K. Cyras and F. Toni, “ABA+: assumption-based argumentation with preferences,” in *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016.*, pp. 553–556, 2016.

[20] C. A. R. Hoare, “An axiomatic basis for computer programming,” *Commun. ACM*, vol. 12, pp. 576–580, Oct. 1969.