Chapter 1

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.

1.1 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.

1.1.1 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
```

1.1.2 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 below:

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

1.2 Usage

1.2.1 Getting Started

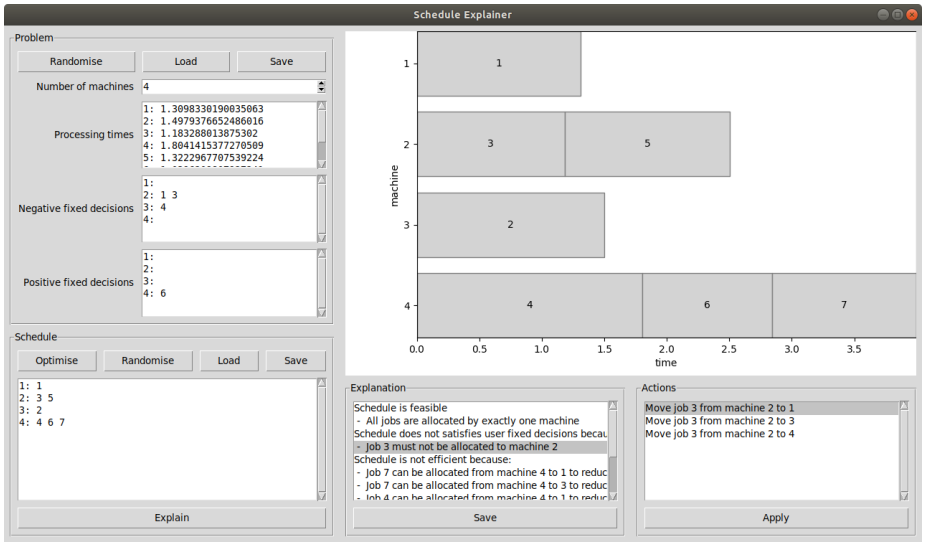

Figure 1.1: Tool GUI

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, 1 represents Alice, 2 represents Bob for nurses and 1 represents Charlie and 2 represents Dave for patients. Although index 1 represents both Alice and Charlie, they are used in different contexts, the problem is well-defined. The problem is to minimise the total completion time, which intuitively is the longest time any nurse has to work.

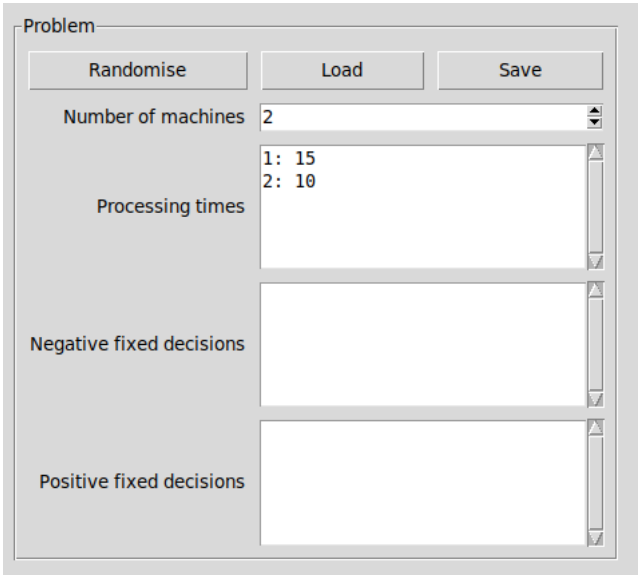

Figure 1.2: Example problem input

Each line in the processing time textbox represents one job. The first line can be interpreted as: job 1 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.

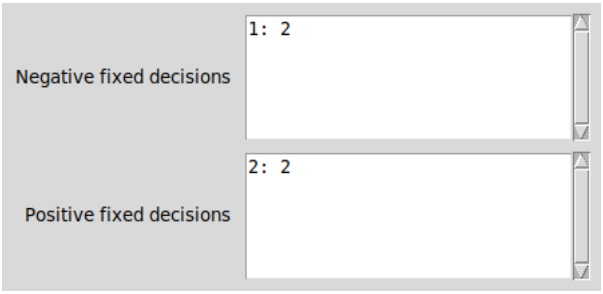

Figure 1.3: Example fixed decisions input

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 2. The first line of positive fixed decisions can be interpreted as: machine

2 cannot be allocated to job 2. In context of the example, this means Alice cannot be with Dave and Bob must be with Dave.

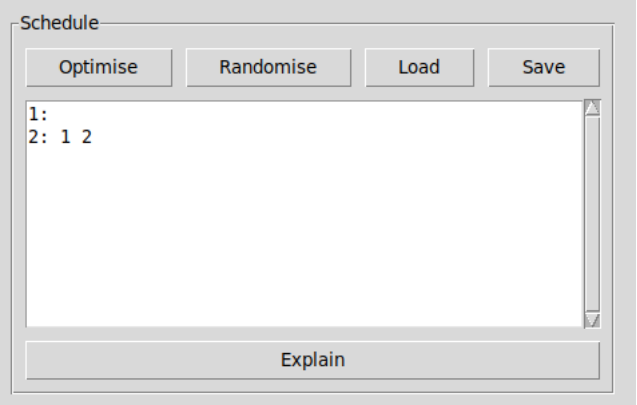

Figure 1.4: Example schedule input

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, 1 and 2. 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.

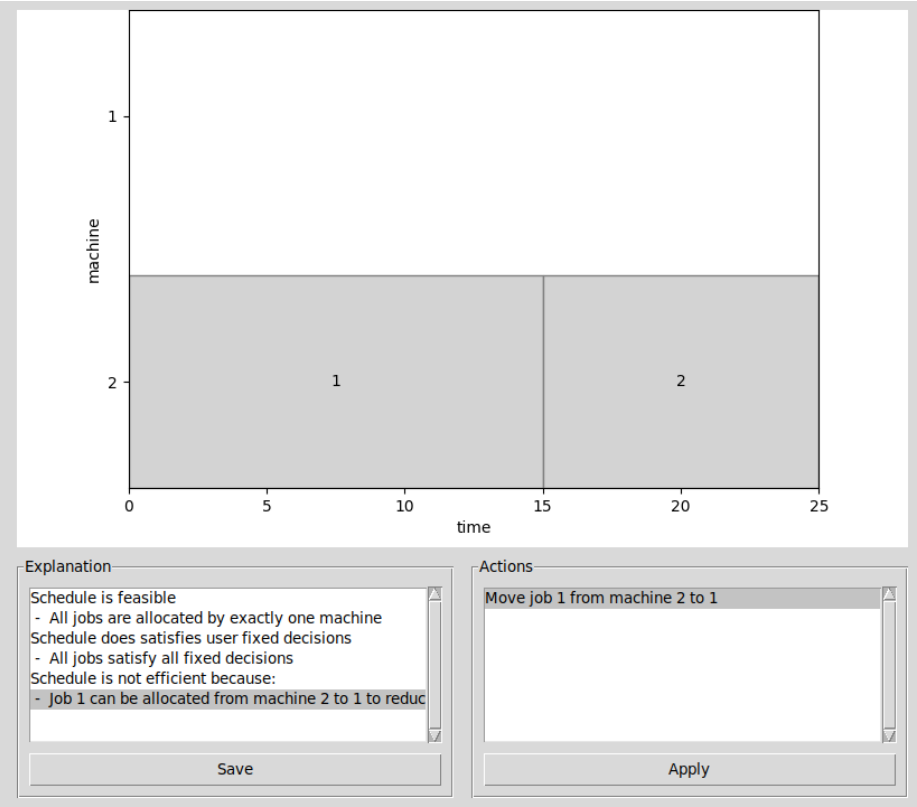

Figure 1.5: Example explanation output

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.

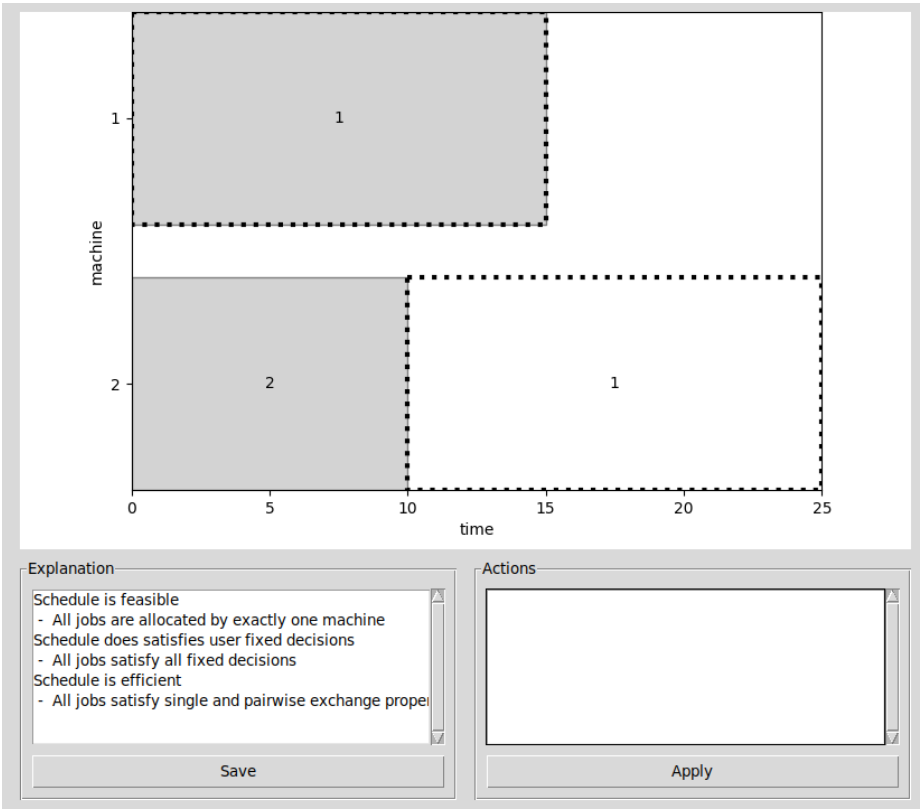

Figure 1.6: Example explanation output

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.

1.2.2 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]
    [-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
-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;
1: 1.0535984963443499
2: 1.5450274194370683
3: 1.0110591477247763
4: 5.180887650551162
5: 1.0613085418860866
;
1:
2: 2
3:
4: 2 3 4 5
;
1: 1
2:
3:
4:
```

Formally, this represents $m = 4$, $\mathbf{p} = \begin{bmatrix}1.05 & 1.54 & 1.01 & 5.18 & 1.06\end{bmatrix}^T$, $D^- = \{\langle 2,2\rangle, \langle 4,2\rangle, \langle 4,3\rangle, \langle 4,4\rangle, \langle 4,5\rangle\}$ and $D^+ = \{\langle 1,1\rangle\}$.

Random schedule given previous problem

```
> python3 main.py -p example.problem -R
1: 2
2: 4 5
3:
4: 1 3
```

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

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 1 must be allocated to machine 1
 -  Job 3 must not be allocated to machine 4
Schedule is not efficient because:
 -  Job 4 can be allocated from machine 2 to 3 to reduce by 1.06
 -  Job 5 can be allocated from machine 2 to 1 to reduce by 1.06
 -  Job 5 can be allocated from machine 2 to 3 to reduce by 1.06
```

1.3 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.