ALC 2020/2021

1^{st} Project – Single Machine Scheduling with SAT/MaxSAT/PB

12/Oct/2020, version 1.0

Overview

The 1st ALC project is to develop a software tool for solving the Single Machine Scheduling (JFS) problem. In order to solve this problem, students must use solvers for Satisfiability (SAT), Maximum Satisfiability (MaxSAT), Pseudo-Boolean Satisfiability (PBS) or Pseudo-Boolean Optimization (PBO).

Problem Specification

In many real-time systems, each task may have several time requirements. In cases where it is not possible to meet all the time demands, one must schedule the tasks to execute in order to maximize the number of on-time tasks [1].

A real-time system is composed by a set of n tasks $\{\tau_1, \ldots, \tau_n\}$ that are to be executed in a uniprocessor machine. Hence, tasks must be executed in sequence. Each task τ_i is represented by a triple (r_i, p_i, d_i) where r_i denotes the release time (i.e. the earliest timestamp that task τ_i can start), p_i denotes the processing time (i.e. the task duration in the machine) and d_i denotes the deadline (i.e. the timestamp at which task τ_i must be completed). For all tasks you can assume that $r_i + p_i \leq d_i$, otherwise the task could never be completed.

Tasks can be split into fragments so that it can be interrupted in order to execute fragments of other tasks. Hence, a task τ_i can be split into k_i fragments where each fragment $j \in \{1 \ldots k_i\}$ takes p_i^j time units to be executed. Observe that the execution of a fragment is indivisible and that for all tasks τ_i we have $\sum_{j=1}^{k_i} p_i^j = p_i$. Note also that the order of execution of fragments cannot be changed, i.e. all fragments of a task must be executed in sequence.

Finally, it may be the case that tasks depend on the completion of other tasks. Hence, if task τ_i depends on task τ_j, this means that τ_j must be completed before τ_i starts. The overall goal is to find a schedule that maximizes the number of completed tasks.

Table 1 illustrates a problem instance with 4 tasks. For this example, the optimal solution is 3 and it is shown in Figure 1. Observe that if task 2 were choosen, then task 1 could not be completed in time. Therefore, if task 1 is not completed, task 3 could not be scheduled since it depends on task 1. Finally, observe that task 3 was split in two fragments in order to be able to schedule task 4.

	r_i	p_i	d_i	Fragments	Dependencies
Task 1	0	4	6	2, 1, 1	none
Task 2	2	3	5	2, 1	none
Task 3	3	2	9	1, 1	1
Task 4	5	3	8	1, 2	none

Table 1: Single Machine Scheduling

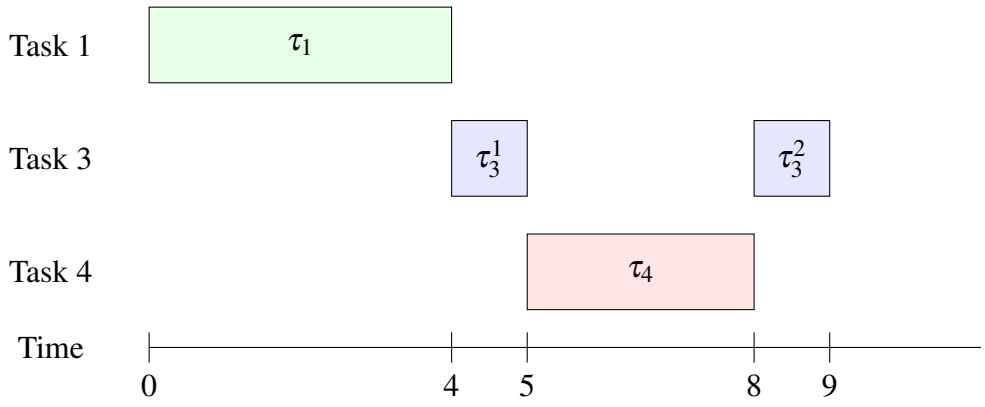

Figure 1: Single Machine Scheduling Solution

Project Goals

You are to implement a tool, or optionally a set of tools, invoked with command `proj1`. This set of tools should use a SAT/MaxSAT/PB solver to compute the single machine scheduling problem.

Your tool does not take any command-line arguments. The problem instance is to be read from the standard input.

Consider an instance file named `job.sms`. The tool is expected to be executed as follows:

```
proj1 < job.sms > solution.txt
```

The tool must write the solution to the standard output, which can then be redirected to a file (e.g., `solution.txt`).

The programming languages to be used are only C/C++, Java or Python. The formats of the files used by the tool are described below.

File Formats

You can assume that all input files follow the description provided in this document. There is no need to check if the input file is correct. Additionally, all lines (input or output) must terminate with the end-of-line character.

Input Format

The input file representing a problem instance is a text file that follows the following format:

- One line with an integer $n (n > 1)$ defining the number of tasks
- A sequence of n lines where the i^{th} line describes task i. Each line describing a task contains the following:
  - Four integers separated by spaces denoting respectively the release time (r_i), processing time (p_i), deadline time (d_i) and the number of fragments k_i.
  - A sequence of k_i integers separated by spaces denoting the processing time of each fragment.
- A sequence of n lines where the i^{th} line describes the dependencies of task i. Each line describing the dependencies contains the following:
  - An integer m_i defining the number of dependencies
  - A sequence of m_i integers separated by spaces denoting the tasks from which task i depends.

Output Format

The output of the program representing an optimal solution to the problem instance must comply with the following format:

- One line with an integer U defining the optimal number of tasks scheduled to the machine.
- A sequence of U lines where each line contains the schedule for each task assigned to the machine.
  - An integer i denoting the scheduled task
  - A sequence of k_i integers separated by the space character where each integer denotes the starting time of each fragment. The starting time follows the order of the fragments of task i.

Important: The final version to be submitted for evaluation must comply with the described output. Project submissions that do not comply will be severely penalized, since each

incorrect output will be considered as a wrong answer. An application that verifies if the output complies with the description will be available on the course's website.

Example

The file describing the problem in Table 1 is as follows:

```
4
0 4 6 3 2 1 1
2 3 5 2 2 1
3 2 9 2 1 1
5 3 8 2 1 2
0
0
1 1
0
```

The optimal solution corresponding to Figure 1 would be:

```
3
1 0 2 3
3 4 8
4 5 6
```

Additional Information

The project is to be implemented in groups of one or two students.

The project is to be submitted through the course website. Jointly with your code, you should submit a short text file describing the main features of your project.

The evaluation will be made taking into account correctness given a reasonable amount of CPU time (80%) and efficiency (20%).

The input and output formats described in this document must be strictly followed.

Project Dates

- Project published: 14/10/2020.
- Project due: 06/11/2020 at 23:59.

Omissions & Errors

Any detected omissions or errors will be added to future versions of this document. Any required clarifications will be made available through the course's official website.

Versions

14/Oct/2020, version 1.0: Original version.

References

[1] X. Liao, H. Zhang, M. Koshimura, R. Huang, and W. Yu. Maximum satisfiability formulation for optimal scheduling in overloaded real-time systems. In A. C. Nayak and A. Sharma, editors, *PRICAI 2019: Trends in Artificial Intelligence - 16th Pacific Rim International Conference on Artificial Intelligence, Cuvu, Yanuca Island, Fiji, August 26-30, 2019, Proceedings, Part I*, volume 11670 of *Lecture Notes in Computer Science*, pages 618–631. Springer, 2019.