Search and Planning 2024/25
Project – Test Scheduling Problem as a CSP

13/09/2024

Overview

The Search and Planning project is to develop an encoding for solving the Test Scheduling Problem (TSP) as a Constraint Satisfaction Problem (CSP). The problem was presented as the Industrial Modelling Challenge at CP2015.

Problem Specification

According to the specification of problem 073 in CSPlib [1], the TSP arises in the context of a testing facility. Several tests have to be performed in minimum time.

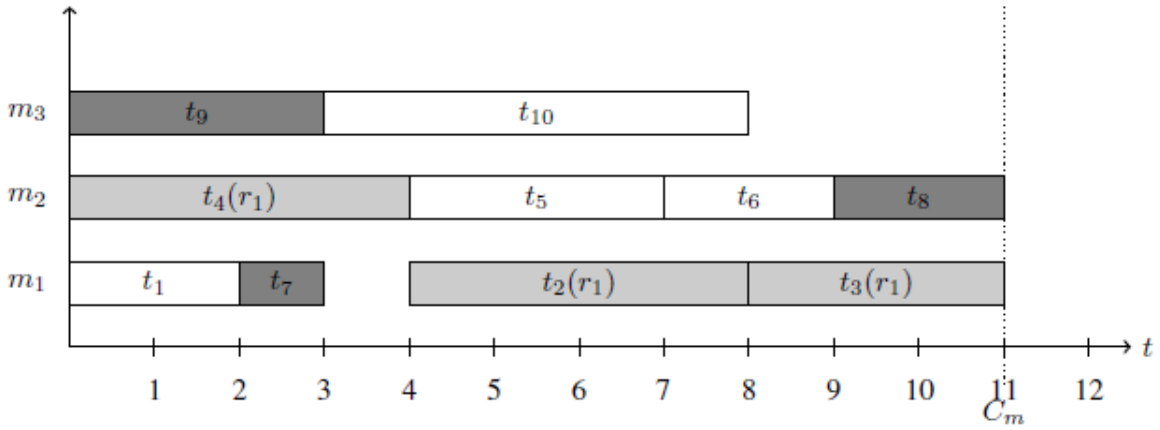
Each test has a given duration and needs to run on one machine. While the test is running on a machine, no other test can use that machine. Some tests can only be assigned to a subset of the machines, for others you can use any available machine. For some tests, additional, possibly more than one, global resources are needed. While those resources are used for a test, no other test can use them.

The objective is to **finish the set of all tests as quickly as possible**, i.e. all start times should be non-negative, and makespan should be minimized. The makespan is the difference between the start of the earliest test, and the end of the latest finishing test.

An example instance with 10 tests, 3 machines and 1 resource is shown bellow. The respective solution is shown on the next page.

Test	Duration	Executable on	Use of global resource
t_1	2	m_1, m_2, m_3	-
t_2	4	m_1, m_2, m_3	r_1
t_3	3	m_1, m_2, m_3	r_1
t_4	4	m_1, m_2, m_3	r_1
t_5	3	m_1, m_2, m_3	-
t_6	2	m_1, m_2, m_3	-
t_7	1	m_1	-
t_8	2	m_2	-
t_9	3	m_3	-
t_{10}	5	m_1, m_3	-

[1] https://www.csplib.org/Problems/prob073/



Note that there may exist more than one solution for this problem instance, but for all solutions the makespan has to be 11.

A more detailed description of TSP (although with a slightly different input format) can be found here: https://www.csplib.org/Problems/prob073/assets/description.pdf.

Project Goals

You are to implement a program in Python in file `proj.py`, that should take as argument the name of a file representing the input data.

This program should use the MiniZinc CSP solver (version 2.8.5 available from https://www.minizinc.org/) to minimize the makespan given the input data [2]. For example, for the problem instance given above the makespan is 11.

Your program is expected to be executed as follows:

```
python proj.py <input-file-name> <output-file-name>
```

For each specific input file, your program should write the solution to an output file. More information regarding the input and output format is provided in the next section.

The programming language to use is Python version 3.11.

Formats

Input Format

For the input, the data is provided as a text file in a Prolog-style having the following format:

- A few comment lines starting with % and containing information about the the number of tests, machines and resources, as well as the maximum makespan.

[2] You may consider using MiniZinc Python (https://python.minizinc.dev/).

- One line describing each test, mentioning its name, duration, required machines (optional) and required resources (optional). Assume that if no machines are required (represented as []) then any machine can be used to perform that test.

The running example has the following input:

```
% Number of tests      :  10
% Number of machines   :  3
% Number of resources  :  1
% Maximum makespan     :  15
test( 't1', 2, [], [])
test( 't2', 4, [], ['r1'])
test( 't3', 3, [], ['r1'])
test( 't4', 4, [], ['r1'])
test( 't5', 3, [], [])
test( 't6', 2, [], [])
test( 't7', 1, ['m1'], [])
test( 't8', 2, ['m2'], [])
test( 't9', 3, ['m3'], [])
test( 't10', 5, ['m1','m2','m3'], [])
```

Output Format

For the output, your program should provide a text file having the following format:

- Comment lines including the makespan.
- One line for each machine containing the number of tests, followed by the assigned tests. For each test, the start time and the resources used (optional) should be indicated.

A possible output for the running example is:

```
% Makespan :  11
machine( 'm1', 2, [('t9',0),('t10',3)])
machine( 'm2', 4, [('t4',0,'r1'),('t5',4),('t6',7),('t8',9)])
machine( 'm3', 4, [('t1',0),('t7',2),('t2',4,'r1'),('t3',8,'r1')])
```

Additional Example

Input:

```
% Number of tests      :  5
% Number of machines   :  2
% Number of resources  :  1
% Maximum makespan     :  10
```

```
test( 't1', 3, [], [])
test( 't2', 2, [], ['r1'])
test( 't3', 4, [], ['r1'])
test( 't4', 1, ['m1'], [])
test( 't5', 2, ['m2'], [])
```

Output:

```
% Makespan :  6
machine( 'm1', 3, [('t2',0),('t1',2),('t4',5,'r1')])
machine( 'm2', 2, [('t5',0),('t3',2,'r1')])
```

Additional Information

The project is to be implemented in groups of one or two students. Registration through Fenix is required.

The project code is to be submitted through the course website. You should submit a zip archive (with name `<group_number>.zip`) with the source code of your solution. The input and output formats described in this document must be strictly followed.

You should also submit a video with the maximum duration of 3+2 minutes. In that video, all the members of the group should appear. In the first 3 minutes, the work developed should be explained. In the last 2 minutes, a demo of the software developed, using some of the provided problem instances, should be included.

The evaluation will be made taking into account (i) correctness given a reasonable amount of CPU time (60%), (ii) quality of the solution (20%) and (iii) video (20%).

If needed, an oral evaluation for specific groups may take place.

Code Plagiarism Detection

The same or very similar projects will lead to failure in the course and, eventually, to the lifting of a disciplinary process. If you use AI-based tools or existing web solutions you should describe its use in the video.

Project Dates

- Project published: 13/09/2024.
- Fenix group registration: 23/09/2024.
- Project due: 18/10/2024 (code 17h00 and video 23h59).

Checkpoints

There will be a few checkpoints in the practical classes, although no evaluation will be given. The checkpoints are as follows:

1. Week 23-27 September: read input, define variables.
2. Week 30 September-4 October: implement constraints.
3. Week 7-11 October: guarantee correctness.
4. Week 14-18 October: improve performance.

Omissions & Errors

Any required clarifications will be made available through the course's official website.