

Job Scheduling in MiniZinc

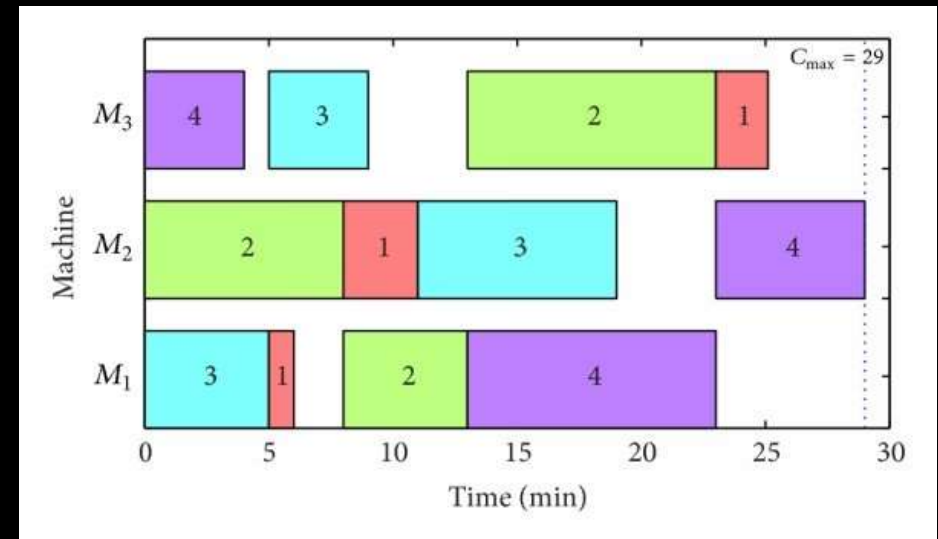
Knowledge Representation Project – Università della Calabria
Master Degree in Artificial Intelligence and Data Science
a.y. 2021/2022

Job Scheduling in MiniZinc Kr Project

- Job Scheduling Problems
- MiniZinc Models
- Java Execution Functions

Job Scheduling Problems

- **Job Scheduling (JS)** or **Job Problem (JP)** is a popular optimization problem in computer science and operational research.
- This focus on assigning jobs to one or more resources or machines at particular times.

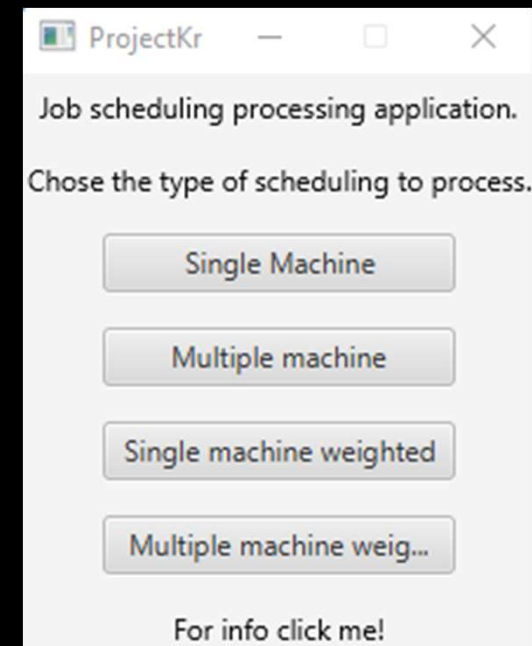


The schedule of 4 job and 3 machines.

Job Scheduling Problems

- There are several type of job scheduling problems.
- Our application is focused on process **single** or **multiple** machine(s) problem with or without assigning **weights** to jobs.
- In addition, the objective function aims to minimize the sum of **completion times**.

- The home page allows users to select the type of job scheduling problem to process.



MiniZinc Models

Our project requires four MiniZinc models (one per type of job scheduling problem)

- Single Machine Model
 - Single Machine with weights Model
 - Multiple Machines Model
 - Multiple Machines with weights Model
- For example the *jobScheduling_MultipleMachines_weights.mzn* file is structured as shown in the next slide.

MiniZinc Models

jobScheduling_MultipleMachines_weights.mzn file

```
1 int: njobs;
2 int: nmachines;
3 set of int: jobs=1..njobs;
4 set of int: machines=1..nmachines;
5 set of int: dom = 0..1;
6
7 array[jobs] of int: p;%input tempi di processamento
8 array[jobs] of int: w;%input dei pesi
9 array[machines] of int: wMachines;%input dei pesi sulle macchine
10
11 constraint assert(njobs > 0, "ValueError: njobs must be non-negative, you have to schedule at least 1 job");
12 constraint assert(nmachines > 1, "ValueError: machines must be more than one, otherwise select single machine option");
13 constraint assert(nmachines < njobs, "ValueError: machines must be less than jobs");
14 constraint assert(forall(i in jobs)(p[i]>0), "ValueError: processing times must be at least 1");
15 constraint assert(forall(i in jobs)(w[i]>0), "ValueError: weights must be at least one");
16 constraint assert(forall(i in machines)(wMachines[i]>0), "ValueError: weights on Machines must be at least 1, otherwise select multiple machine option");
17
18
19 array[machines,jobs] of var dom: assign;% tempi di completamento
20
21 var int: v;
22 constraint forall(i in machines)(assign[i,i]=1);
23 constraint forall(j in jobs)(sum(i in machines)(assign[i,j]) = 1);
24 constraint forall(i in machines)(v>=sum(j in jobs)(assign[i,j]*p[j]));
25
26 %vincolo sulle macchine che possono processare job con pesi <= wMachines[i]
27 constraint forall(i in machines, j in jobs)(assign[i,j]*w[j] <= wMachines[i]);
28
29 solve minimize v;
30
31 output[
32   show_int(3, assign[i,j]) ++ if j==njobs then "\n" else " " endif | i in machines, j in jobs
33 ];
34 output["total completion time minimized =\{(v)\n"];
```

Java Execution Functions

- To invoke the **MiniZinc Solver** in Java it is necessary to run a simple command line.

```
> Minizinc.exe --solver Gecode filename.dzn filename.mzn
```

Java Execution Functions

- In our code the function `runSolver(String s)` executes the right model in base of the matching of the type passed as parameter.

```
public String runSolver(String s){
    String cmd = "C:\\Program Files\\MiniZinc\\minizinc.exe --solver Gecode jobSchedulingInput.dzn";
    Runtime run = Runtime.getRuntime();
    Process pr = null;
    String l = "";
    try {
        pr = run.exec(cmd);
        try {
            pr.waitFor();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        BufferedReader buf = new BufferedReader(new InputStreamReader(pr.getInputStream()));
        String line = "";
        while (true) {
            if (!((line=buf.readLine())!=null)) break;
            System.out.println(line);
            l+="\n"+line;
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    System.out.println(l);
    return l;
}
```


Java Execution Functions

- This function is called when all the text field are filled and the user click on the button “Execute”.
- In the right part of the window it is shown the output of the MiniZinc Solver.

The screenshot shows a Java application window titled "Graph". The window has a red title bar with standard Windows window controls (minimize, maximize, close). The main content area is divided into two parts. On the left, there is a form for inputting job data. It starts with a label "Press enter to confirm" and a text field for "# Jobs" containing the value "4". Below this are two buttons: "Execute" (highlighted with a blue border) and "Example". Under the buttons, there are four rows of input fields for job details: Job 1 (5, W1: 2), Job 2 (4, W2: 6), Job 3 (4, W3: 3), and Job 4 (2, W4: 4). On the right side of the window, the output of the MiniZinc Solver is displayed in red text. It shows "completion time per jobs = [2, 6, 10, 15]" and "total weighted completion time minimized = 104", followed by a dashed line and a line of equals signs.

| Job | Weight | Completion Time |
|-------|--------|-----------------|
| Job 1 | 5 | 2 |
| Job 2 | 4 | 6 |
| Job 3 | 4 | 10 |
| Job 4 | 2 | 15 |

completion time per jobs = [2, 6, 10, 15]
total weighted completion time minimized = 104

=====



Demo

To download the app:
[GitHub](#)

THANKS

Project and Presentation by Boca Paolo, Canonaco Martina, Fazio
Francesco, Moscato Giuseppe and Passarelli Chiara.