

Lab 4 Parallel Machine Scheduling

Mateo Saenz Ortiz

November 26, 2024

Introduction

This report presents the solution to the Parallel Machine Scheduling Problem (P——Cmax), where the goal is to assign n tasks to m parallel machines while minimizing the makespan (Cmax). The mathematical model, results, and comparisons between IBM ILOG CPLEX and Python are discussed.

Mathematical Model

The mathematical model for the Parallel Machine Scheduling Problem is as follows:

Parameters

- n : Number of tasks
- m : Number of machines
- p_i : Processing time of task i , where $i \in \{1, 2, \dots, n\}$
- C_{\max} : Makespan (maximum completion time)

Decision Variables

- $x_{ij} \in \{0, 1\}$: Binary variable, where $x_{ij} = 1$ if task i is assigned to machine j , and 0 otherwise.
- C_j : Completion time of machine j , where $j \in \{1, 2, \dots, m\}$.

Objective Function

Minimize the makespan:

$$\min C_{\max}$$

Constraints

1. Task Assignment: Each task must be assigned to exactly one machine.

$$\sum_{j=1}^m x_{ij} = 1, \quad \forall i \in \{1, 2, \dots, n\}$$

2. Machine Completion Time: The completion time of a machine is the sum of the processing times of tasks assigned to it.

$$C_j = \sum_{i=1}^n p_i x_{ij}, \quad \forall j \in \{1, 2, \dots, m\}$$

3. Makespan Definition: The makespan is the maximum of all machine completion times.

$$C_{\max} \geq C_j, \quad \forall j \in \{1, 2, \dots, m\}$$

4. Binary Decision Variables: Ensure tasks are either assigned or not.

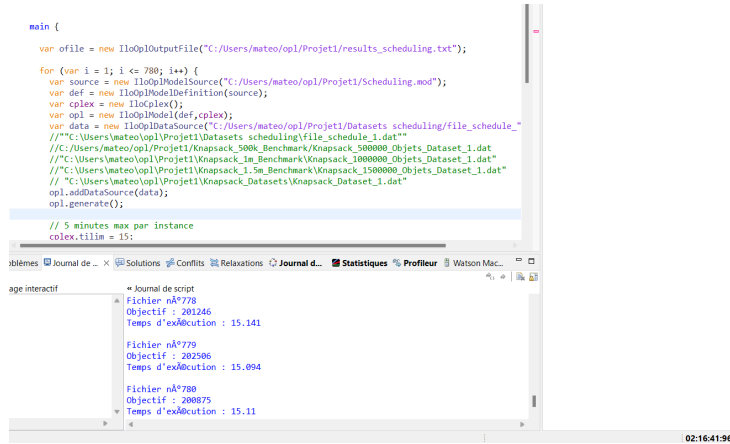
$$x_{ij} \in \{0, 1\}, \quad \forall i \in \{1, 2, \dots, n\}, \forall j \in \{1, 2, \dots, m\}$$

Methodology

IBM ILOG CPLEX

For the IBM ILOG CPLEX implementation:

- Data preparation was handled in Python, where task and machine information was extracted from text files and formatted into .dat files. These files facilitated easy reading in the CPLEX environment.
- The mathematical model was developed to represent the Parallel Machine Scheduling Problem. A loop was created to iterate through all datasets with a processing time limit of 15 seconds per instance.



```
main {
    var ofile = new IloOplOutputFile("C:/Users/mateo/opl/Projet1/results_scheduling.txt");
    for (var i = 1; i <= 780; i++) {
        var source = new IloOplModelSource("C:/Users/mateo/opl/Projet1/Scheduling.mod");
        var def = new IloOplModelDefinition(source);
        var cplex = new IloCplex();
        var opl = new IloOplModel(def, cplex);
        var data = new IloOplDataSource("C:/Users/mateo/opl/Projet1/Datasets_scheduling/file_schedule_";
        //C:/Users/mateo/opl/Projet1/Datasets_scheduling/file_schedule_1.dat"
        //C:/Users/mateo/opl/Projet1/Knapsack_500k_Benchmark/Knapsack_500000_Objects_Dataset_1.dat"
        //C:/Users/mateo/opl/Projet1/Knapsack_1m_Benchmark/Knapsack_1000000_Objects_Dataset_1.dat"
        //C:/Users/mateo/opl/Projet1/Knapsack_1.5m_Benchmark/Knapsack_1500000_Objects_Dataset_1.dat"
        //C:/Users/mateo/opl/Projet1/Knapsack_Datasets/Knapsack_Dataset_1.dat"
        opl.addDataSource(data);
        opl.generate();

        // 5 minutes max par instance
        cplex.time = 15;
    }
}
```

Journal de script

Fichier	n°	Objectif	Temps d'exécution
Fichier n°778		201246	15.141
Fichier n°779		202506	15.094
Fichier n°780		200875	15.11

02:16:41:96

Figure 1: Iloc process time and results

Python with Docplex

For the Python implementation:

- The `docplex` library was utilized for modeling and solving the problem.
- The local machine's CPLEX solver was integrated via its system path to execute the optimization.
- Three primary functions were created:
 1. A function to read each dataset from the specified folder.
 2. A function to apply the solver to the model and retrieve the results.
 3. A function to create a DataFrame and compare the results with the ILOG CPLEX implementation.

Total (root+branch&cut) = 15.89 sec. (3162.84 ticks)
Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings](#)...

	File	Machines	Tasks	Processing Times	Optimal Makespan	Upper Bound (UB)	Lower Bound (LB)	Gap	Execution Time	Task Assignment
0	NU_1_0010_05_0.txt	5	10	[99, 90, 96, 98, 96, 95, 98, 97, 95, 1]	193.0	193.0	193.000000	0.000000	1.766	[[1, 0, 0, 0, 0], [0, 0, 1, 0, 0], [0, 1, 0, 0, 0]]
1	NU_1_0010_05_1.txt	5	10	[97, 90, 99, 90, 98, 91, 100, 94, 93, 15]	189.0	189.0	189.000000	0.000000	0.453	[[0, 1, 0, 0, 0], [0, 0, 1, 0, 0], [0, 0, 0, 1, 0]]
2	NU_1_0010_05_2.txt	5	10	[92, 91, 94, 95, 92, 94, 100, 93, 90, 3]	186.0	186.0	186.000000	0.000000	0.453	[[0, 0, 0, 1, 0], [0, 0, 0, 0, 1], [0, 0, 0, 1, 0]]
3	NU_1_0010_05_3.txt	5	10	[97, 97, 91, 93, 92, 91, 97, 90, 91, 7]	188.0	188.0	188.000000	0.000000	0.187	[[0, 0, 0, 1, 0], [0, 1, 0, 0, 0], [0, 0, 0, 1, 0]]
4	NU_1_0010_05_4.txt	5	10	[91, 97, 98, 93, 94, 97, 95, 95, 100, 10]	191.0	191.0	191.000000	0.000000	0.515	[[1, 0, 0, 0, 0], [0, 0, 0, 1, 0], [0, 0, 0, 0, 0]]
5	NU_1_0010_05_5.txt	5	10	[92, 93, 96, 100, 97, 93, 94, 92, 94, 13]	189.0	189.0	189.000000	0.000000	1.281	[[0, 0, 1, 0, 0], [1, 0, 0, 0, 0], [0, 1, 0, 0, 0]]
6	NU_1_0010_05_6.txt	5	10	[99, 92, 95, 90, 96, 98, 93, 92, 95, 3]	188.0	188.0	188.000000	0.000000	1.125	[[0, 0, 0, 0, 1], [0, 0, 1, 0, 0], [1, 0, 0, 0, 0]]
7	NU_1_0010_05_7.txt	5	10	[97, 99, 92, 97, 96, 95, 94, 94, 91, 20]	190.0	190.0	190.000000	0.000000	0.610	[[0, 1, 0, 0, 0], [1, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
8	NU_1_0010_05_8.txt	5	10	[90, 91, 100, 100, 95, 95, 98, 94, 94, 13]	190.0	190.0	190.000000	0.000000	0.438	[[1, 0, 0, 0, 0], [0, 1, 0, 0, 0], [0, 0, 0, 0, 0]]
9	NU_1_0010_05_9.txt	5	10	[91, 95, 96, 98, 98, 94, 92, 95, 96, 9]	190.0	190.0	190.000000	0.000000	0.312	[[1, 0, 0, 0, 0], [0, 1, 0, 0, 0], [1, 0, 0, 0, 0]]

Spaces: 4 CRLF Cell 5 of 9 Mite unsupported Q ()

Figure 2: Python process result

Results

Comparison of Results

The comparison between the results from ILOG CPLEX and Python is summarized below:

- **UB Differences:** Instances where the upper bounds differed:
 - Filename 14: UB CPLEX = 937, UB Python = 936
 - Filename 17: UB CPLEX = 944, UB Python = 943
 - Filename 30: UB CPLEX = 474, UB Python = 473
- **Gap Mean Comparison:**
 - Mean Gap (CPLEX): 0.0076
 - Mean Gap (Python): 0.0069

- Difference: -0.0007
- **Execution Time Mean Comparison:**
 - Mean Execution Time (CPLEX): 10.11 seconds
 - Mean Execution Time (Python): 10.39 seconds
 - Difference: 0.29 seconds

Analysis

The analysis reveals that:

- Differences in the results for larger instances can be attributed to the 15-second processing time limit. Despite this, the solutions obtained from both approaches are highly similar.
- In the model after 50 task with the same 5 machines the time of the solver take more time arriving to the limit time, because of the election of the same quantity of machines with x10 the number of task, if we get bigger the number of machines availables, the execution time will be less.
- Execution time was slightly better in the CPLEX implementation, but the difference is not significant.
- The Python script offers more flexibility and simplifies the analysis process, making it a powerful alternative for solving and comparing results.

Conclusion

This study highlights the effectiveness of both IBM ILOG CPLEX and Python in solving the Parallel Machine Scheduling Problem. While CPLEX excels in computation time for larger instances, Python's integration with `docplex` provides an accessible and analytical approach. The comparison demonstrates that both methods yield comparable results, validating the robustness of the proposed model.

The time process can be better with the up of the machines but with a limit time of 15 seconds the solver have a really good performance.