Process Scheduling Optimization: CPLEX Model and Heuristic Comparison (FCFS & SJF)

October 2025

Abstract

This project addresses a process scheduling optimization problem in a multi-core CPU system. Each process requires computational time and may have an associated priority/profit, while each CPU core can execute only one process at a time. The objective is to allocate processes optimally across cores to maximize the total executed priority or the number of completed processes within a fixed time limit.

A mathematical formulation equivalent to an Integer Linear Programming (ILP) problem — analogous to the classical Knapsack problem — is implemented using IBM CPLEX to obtain the optimal solution. The results are compared with two heuristic algorithms widely used in scheduling literature: First Come First Served (FCFS) and Shortest Job First (SJF), implemented in Python.

Experimental results show that while CPLEX guarantees optimality, FCFS and SJF can achieve near-optimal solutions with significantly lower computational cost. On average, SJF performed within 1.13% of the optimal solution (without priorities), while FCFS achieved competitive results when priorities were considered, with an average gap of 8.7%. Given their simplicity and speed — up to 99.5% faster than CPLEX — these heuristic methods offer a practical trade-off between accuracy and efficiency for real-time scheduling applications.

Keywords

Process scheduling, optimization, linear programming, knapsack problem, CPLEX, heuristic algorithms

1. Introduction to the problem

In modern computer systems, multiple processes compete for computational resources and must share the CPU, as they cannot utilize it simultaneously. Each process typically has an estimated execution time and may also have an associated priority or profit, while the CPU is equipped with a certain number of cores, each capable of executing only one process at a time.

The goal of **process scheduling** is to allocate processes to CPU cores in an optimal way — aiming to maximize either the total number of executed processes within a given time frame or the overall value of their priorities.

However, real-world scheduling algorithms must strike a balance: they seek to distribute processes efficiently to avoid deadlocks or starvation, but they cannot always achieve absolute optimality, as doing so would introduce undesirable latency into the system.

In this work, we address an **optimization problem** related to scheduling a set of processes on a system with four cores using **IBM CPLEX**, which guarantees the optimality of the computed solution. The obtained results are then compared with those produced by **heuristic methods**, which can find feasible and generally good — though not necessarily optimal — solutions. Specifically, two well-known scheduling algorithms from the literature are considered: **First Come First Served (FCFS)** and **Shortest Job First (SJF)**.

The FCFS algorithm executes processes in the order of their arrival, whereas the SJF algorithm assigns resources first to the processes with the shortest execution times.
Experiments are repeated under two scenarios: one that ignores process priorities and one that incorporates them, adapting the algorithms accordingly.
For simplicity, the parallelism of process-to-core assignment is not considered in this study.

2. Mathematical formulation

Notation

The scenario previously described can be formalized as an optimization problem, specifically an Integer Linear Programming (ILP) problem.

It is closely related to the Knapsack Problem, since we are given a set of elements (the processes), each associated with a weight (its execution time) and a benefit (its execution or its associated priority). The objective is to maximize the total benefit (either the number of executed processes or the total priority of executed processes) without exceeding the maximum capacity (the CPU time limit available to each core).

The problem can be defined as follows:

- n: number of CPU cores
- m: number of processes to be executed
- t_j: execution time required by process j
- p_j: priority or profit of process j (ranging from 1 to 10)
- T_{lim}: maximum execution time available for each core

With the following decision variables:

- $X_{ij} \in \{0, 1\}$: equals 1 if process *j* is assigned to core *i*, and 0 otherwise.

Objective function

Two alternative objective functions were defined for the experiments:
one that ignores process priorities and one that incorporates them, in order to evaluate the algorithms under different perspectives — *pure scheduling* versus *priority-based scheduling*.

The first objective function aims to **maximize the number of processes executed** within the time limit, regardless of their priorities.
Formally:

$$\boldsymbol{max} \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \sum_{\boldsymbol{j=1}}^{\boldsymbol{m}} \boldsymbol{xij}$$

The second objective function takes process priorities into account, aiming to maximize the total priority value of the executed processes:

$$\boldsymbol{max} \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \sum_{\boldsymbol{j=1}}^{\boldsymbol{m}} \boldsymbol{xij} * \boldsymbol{pj}$$

Both formulations yield comparable results when all processes have the same priority value.

Constraints

The following constraints are introduced to regulate process assignment to cores, respect time limits, and define the domain of the decision variables.

Each process $\boldsymbol{j}$ can be assigned to at most one core $\boldsymbol{i}$:

$$\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{xij} \leq \mathbf{1}, \quad \forall \boldsymbol{j} \in \{\mathbf{1}, \dots, \boldsymbol{m}\}$$

The total execution time of processes assigned to a core i cannot exceed the time limit T_{lim} :

$$\sum_{j=1}^{m} tj * xij \leq Tlim, \quad \forall i \in \{1, \dots, n\}$$

The decision variables, as already mentioned, are binary:

$$xij \in \{0, 1\}, \qquad \forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, m\}$$

This formulation models the **process scheduling** problem as a maximization task — either of the number of executed processes or of their cumulative priority value — subject to execution time and assignment constraints.
The model enables the computation of an **optimal solution** for a given set of processes and cores using **CPLEX**, which will serve as a benchmark for comparison with the heuristic algorithms **FCFS** and **SJF**, in order to highlight possible differences in solution quality.

3. Algorithms

Data Generation

To begin the experimentation phase, a dedicated **data generation function** was implemented to create the input set on which the scheduling problem is solved.
This dataset includes:

- A set of **CPU cores**, each with a defined **time limit**
- A set of **processes**, each characterized by an **execution time** and an associated **priority value**

The generation of this data depends on the following input parameters:

- Number of CPU cores
- Number of processes to be scheduled
- Time limit assigned to each core
- Maximum execution time allowed for randomly generated processes

Although the data generation process relies on random sampling, it is **pseudo-random**, as the use of a fixed **seed value** ensures reproducibility of experiments.

This allows for a consistent comparison between different algorithms, since all methods operate on identical datasets across runs.

```python
def generaDati(numeroCore, numeroProcessi, tempoCore, tempoMaxProcessi):

    np.random.seed(59)
    Tlim= [tempoCore] * numeroCore
    t = np.random.randint( low: 1, tempoMaxProcessi, numeroProcessi)  # Tempi di esecuzione dei processi
    p = np.random.randint( low: 1,  high: 10, numeroProcessi)  # Priorità dei processi
    nomi = [f'processo_{i}' for i in range(numeroProcessi)]

    dati = {
        "Tlim": Tlim,
        "processi": [
            {"nome": nomi[i], "t": t[i], "p": p[i]} for i in range(numeroProcessi)
        ]
    }

    return dati
```

Model Creation

The model used to solve the process scheduling problem was implemented using **DOcplex**, and it is capable of finding the **optimal solution** associated with a specific **objective function value** and a **process–core assignment configuration**.

At the initial stage, the input data (as previously described) are extracted, and the **optimization model** is instantiated through the Model class:

```python
def create_model(dati):

    Tlim = dati["Tlim"]  # Tempo massimo per ogni core
    processi = dati["processi"]  # Dati processi

    numeroCore = len(Tlim)
    numeroProcessi = len(processi)

    t = np.array([p["t"] for p in processi])  # Tempi di esecuzione
    p = np.array([p["p"] for p in processi])  # Priorità
    nomi = [p["nome"] for p in processi]  # Nomi processi

    # Creazione modello
    model = Model("process_scheduling")
```

As illustrated earlier, two different **objective functions** were defined.
They differ in the goal they pursue:

- The first one aims to **maximize the total priority value** of the processes executed within the time limit.
- The second one ignores priority and instead aims to **maximize the total number of processes executed**.

```
#ATTENZIONE: in seguito ci sono 2 funzioni obbiettivo, lasciarne solo 1 non commentata

    # Funzione obiettivo per massimizzare il valore totale delle priorità dei processi eseguiti
    model.maximize(model.sum(p[j] * x[i, j] for i in range(numeroCore) for j in range(numeroProcessi)))

    # Funzione obiettivo per massimizzare il numero di processi eseguiti
    model.maximize(model.sum(x[i, j] for i in range(numeroCore) for j in range(numeroProcessi)))
```

The decision variables x_{ij} are **binary**, taking value **1** if process ***j*** is assigned to core ***i*,** and **0** otherwise:

```
# Variabili binarie: 1 se il processo j è assegnato al core i, 0 altrimenti
x = model.binary_var_matrix(numeroCore, numeroProcessi, name="x")
```

A constraint is then added to ensure that each process ***j*** can be assigned to **at most one core *i*:**

```
# Vincolo 1: Ogni processo può essere assegnato a un solo core
for j in range(numeroProcessi):
    model.add_constraint(model.sum(x[i, j] for i in range(numeroCore)) <= 1,  ctname: f"process_assignment_{j}")
```

Finally, another constraint ensures that the **time limit** T_{lim} of each core ***i*** is **not exceeded** by the total execution time of all processes assigned to it:

```
# Vincolo 2: Tempo totale per core non deve superare il limite
for i in range(numeroCore):
    model.add_constraint(model.sum(t[j] * x[i, j] for j in range(numeroProcessi)) <= Tlim[i],  ctname: f"core_time_limit_{i}")

return model
```

Heuristic Algorithms

The heuristic algorithms used to compare against the CPLEX model, as mentioned in the introduction, are **First Come First Served (FCFS)** and **Shortest Job First (SJF)**.
These methods do **not guarantee optimality**, but they are capable of producing feasible and often good solutions in much less time.

Both algorithms were implemented in **Python** in two versions:

- one that considers **process priorities**,
- and one that **ignores** them.

In particular:

- **FCFS** executes processes **in order of arrival**;
- **SJF** executes processes **in order of increasing execution time**.

In the **priority-aware versions**, the algorithms behave similarly, but in the case of **SJF**, if two processes have the same execution time, the one with the **higher priority** is executed first.
In the case of FCFS, priorities do not affect execution, since ties in arrival order are not possible in this implementation.

Both **FCFS** and **SJF** take as input the same data structure used by the CPLEX model and return a dictionary containing:

- the **allocation scheme** (mapping of processes to cores),
- the **total execution time** of each core,
- and a value comparable to the CPLEX **objective function**:
  either the **number of processes executed within the time limit**, or the **total priority value** of those executed processes, depending on whether priorities are considered.

```
return {
    "allocation": core_allocation,
    "tempiCores": tempiCores,
    "processiEseguiti": processiEseguiti,
    "prioritaTotale": prioritaTotale
}
```

Example on a Reduced Dataset

Before comparing the three solution methods on multiple randomly generated datasets, a preliminary test was performed on a **small input set**, ignoring process priorities.
The dataset was defined as follows:

- **4 cores**, each with a **time limit** of 40
- **10 processes** to be scheduled
- **Execution times** ranging from **1 to 40**
- **Process priorities** ranging from **1 to 10**

When solved using the **CPLEX model**, DOcplex automatically generated a log file containing:

- the **number of variables and constraints**,
- the **problem type**,
- the **objective value**,
- **slack values**, and other relevant details.

```
Model: process_scheduling
 - number of variables: 40
   - binary=40, integer=0, continuous=0
 - number of constraints: 14
   - linear=14
 - parameters: defaults
 - objective: maximize
 - problem type is: MILP
// This file has been generated by DOcplex
// model name is: process_scheduling
```

For this dataset, the model found the **optimal solution**, corresponding to an **objective function value of 8** — meaning that **8 out of 10 processes** were successfully scheduled within the time limit. The optimal configuration is represented by the subset of variables x_{ij} that take value **1**, indicating that process ***j*** is assigned to core ***i***:

```
objective: 8
status: OPTIMAL_SOLUTION(2)
  x_0_1=1
  x_0_6=1
  x_1_2=1
  x_1_4=1
  x_2_0=1
  x_2_3=1
  x_3_5=1
  x_3_9=1
```

Running the same dataset through the **heuristic algorithms** produced **suboptimal results**, confirming their **feasibility** but not **optimality** compared to the CPLEX solution:

```
Numero di Processi eseguiti con FCFS:
7
Numero di Processi eseguiti con SJF:
7
```

4. Results and conclusion

To evaluate the performance of the **CPLEX model** and the **FCFS** and **SJF** algorithms, ten tests were conducted on datasets of **increasing size**.
The datasets were generated pseudo-randomly based on the following parameters:

NOME	SEED	CORE	PROCESSI	Tlim	Tmax PROCESSO
s1	11	4	50	100	24
s2	19	4	100	220	30
s3	23	4	100	270	36
s4	38	4	200	500	50
s5	48	4	200	400	40
s6	59	4	200	500	30
s7	65	4	250	500	44
s8	76	4	250	550	45
s9	88	4	250	600	45
s10	93	4	250	700	43

Maximizing the Number of Executed Processes

The first set of experiments was carried out **without considering process priorities**, focusing instead on **maximizing the total number of processes executed** within the time limit.
The obtained results are summarized below:

DATI	CPLEX	FCFS	SJF	PERCENTUALE FCFS	PERCENTUALE SJF
s1	41	37	40	-0.098	-0.024
s2	75	64	74	-0.147	-0.013
s3	76	66	75	-0.132	-0.013
s4	126	90	125	-0.286	-0.008
s5	127	86	125	-0.323	-0.016
s6	157	137	155	-0.127	-0.013
s7	152	91	151	-0.401	-0.007
s8	151	99	150	-0.344	-0.007
s9	161	112	160	-0.304	-0.006
s10	176	130	175	-0.261	-0.006

The solutions produced by the **CPLEX model** are **optimal** for the corresponding optimization problems, while those found by the **heuristic algorithms** are **suboptimal**.

The percentage columns report **how much worse** the heuristic solutions are compared to the **optimal value** found by CPLEX:

DATI	CPLEX	FCFS	SJF	PERCENTUALE FCFS	PERCENTUALE SJF
s1	0.016	0.001329	0.000034	-0.917	-0.998
s2	0.047	0.000238	0.00013	-0.995	-0.997
s3	0.016	0.000052	0.000041	-0.997	-0.997
s4	0.031	0.000096	0.000085	-0.997	-0.997
s5	0.046	0.000096	0.000082	-0.998	-0.998
s6	0.016	0.000093	0.000085	-0.994	-0.995
s7	0.047	0.000123	0.000108	-0.997	-0.998
s8	0.031	0.000117	0.000106	-0.996	-0.997
s9	0.063	0.000123	0.000105	-0.998	-0.998
s10	0.077999	0.000248	0.000209	-0.997	-0.997

On average:

- The **FCFS** algorithm produced results **about 24% worse** than the optimal solution.
- The **SJF** algorithm performed significantly better, with results only **1.13% worse on average** — quite remarkable for a heuristic method, especially when considering its **execution time**, measured precisely down to the microsecond.

Execution time analysis clearly shows that, for these datasets, the **heuristic algorithms** are **significantly faster** than the **CPLEX model**, often by several orders of magnitude.

Maximizing the Total Priority

The tests were then repeated, this time **taking process priorities into account**, with the objective of **maximizing the total priority value** of all processes executed within the time limit.
The obtained results are shown below:

DATI	CPLEXP	FCFSP	SJFP	PERCENTUALE FCFS	PERCENTUALE SJF
s1	223	223	192	0	-0.139
s2	431	402	384	-0.067	-0.109
s3	430	406	375	-0.056	-0.128
s4	716	609	639	-0.149	-0.108
s5	742	647	647	-0.128	-0.128
s6	917	903	825	-0.015	-0.1
s7	835	701	746	-0.16	-0.107
s8	874	765	765	-0.125	-0.125
s9	889	782	777	-0.12	-0.126
s10	1007	956	894	-0.051	-0.112

In this scenario, the **FCFS** method was able to find an **optimal solution** for one of the test cases. In general, however, the heuristic algorithms again produced **suboptimal results**, with:

- **FCFS** performing **8.7% worse** on average than the optimal solution,
- and **SJF** performing **11.8% worse** on average.

When considering execution times, the heuristics once again proved to be **much faster** than the CPLEX model:

DATI	CPLEXP	FCFSP	SJFP	PERCENTUALE FCFS	PERCENTUALE SJF
s1	0.031	0.000026	0.000025	-0.999	-0.999
s2	0.016	0.000102	0.000133	-0.994	-0.992
s3	0.063	0.000054	0.000045	-0.999	-0.999
s4	0.032	0.000102	0.000088	-0.997	-0.997
s5	0.031	0.000097	0.000088	-0.997	-0.997
s6	0.031	0.000092	0.000088	-0.997	-0.997
s7	0.047	0.000119	0.000109	-0.997	-0.998
s8	0.063	0.000119	0.000109	-0.998	-0.998
s9	0.078	0.000118	0.000116	-0.998	-0.999
s10	0.047	0.000256	0.000253	-0.995	-0.995

Conclusion

In conclusion, considering that:

- The **heuristic methods** often came **very close to the optimal solutions** — particularly **SJF** in the non-priority scenario (average difference of **1.13%**) and **FCFS** in the priority-based scenario (average difference of **8.7%**);
- The **computational complexity** of the heuristics, ***O(n processes × m cores)***, is **significantly lower** than the **exponential complexity** of the CPLEX solver;
- Consequently, **FCFS** and **SJF** were able to obtain **feasible solutions** in **execution times up to 99.5% shorter** than CPLEX;

It follows that, in contexts where **solution speed is critical**, such as **process scheduling**, it can be **reasonable to trade optimality for efficiency** by using a heuristic method.

These approaches yield **feasible and near-optimal solutions** in a **fraction of the time**, thereby avoiding unnecessary **system latency** due to the search for an exact optimum that may offer **negligible improvement** over much faster heuristic results.