Project-Report of Industrial Automation: Flexible Machine Routing in a Metal Processing Plant (MILP vs Heuristic)

Benedict Martus, David Charry

June 2025

1 Introduction

This report addresses a scheduling problem in a metal processing plant where custom industrial components (jobs) must be processed through three stages using any of five available machines (M_1 to M_5). Key complexities include **flexible routing** (a job can use any machine for a stage, and even reuse the same machine for different stages), **transfer times** between machines (which vary depending on the specific machines involved), and **machine occupation conflicts** (each machine can only process one job at a time). The primary objective is to **minimize the makespan**, defined as the completion time of the last finished job. We explore both an optimal Mixed-Integer Linear Programming (MILP) model and a heuristic approach to solve this problem.

2 Task 1: MILP Model Formulation

To optimally solve the flexible machine routing and scheduling problem, we formulate it as a Mixed-Integer Linear Programming (MILP) model. The objective is to minimize the makespan, $C_{\max}$, which is the completion time of the last finished job.

2.1 Sets and Indices

- $J = \{1, \ldots, n\}$: Set of jobs, where $n = 10$.
- $S = \{1, \ldots, n_s\}$: Set of stages, where $n_s = 3$.
- $M = \{1, \ldots, m\}$: Set of machines, where $m = 5$ (M_1 through M_5).

2.2 Parameters

- $p_{j,s,m}$: **Processing time** of job j at stage s on machine m. These are randomly generated between 5 and 15 minutes.
- T_{m_1,m_2}: **Transfer time** from machine m_1 to machine m_2. The matrix T is given as:

$$T = \begin{bmatrix} 0 & 3 & 5 & 4 & 6 \\ 3 & 0 & 2 & 4 & 5 \\ 5 & 2 & 0 & 3 & 4 \\ 4 & 4 & 3 & 0 & 2 \\ 6 & 5 & 4 & 2 & 0 \end{bmatrix}$$

- BIG_M: A sufficiently large positive number. Its value is dynamically set in the implementation to be the sum of all processing times plus the maximum transfer time, providing a safe upper bound.

2.3 Decision Variables

- $x_{j,s,m} \in \{0, 1\}$: Binary variable. $x_{j,s,m} = 1$ if job j at stage s is processed on machine m, and 0 otherwise.
- $t_{j,s} \geq 0$: Continuous variable. **Start time** of job j at stage s.
- $y_{(j_1,s_1),(j_2,s_2),m} \in \{0, 1\}$: Binary variable. Used for machine conflict resolution. $y_{(j_1,s_1),(j_2,s_2),m} = 1$ if job-stage combination (j_1, s_1) precedes job-stage combination (j_2, s_2) on machine m, and 0 otherwise. This variable is defined for every distinct pair of job-stage combinations $((j_1, s_1), (j_2, s_2))$.
- $C_{\max} \geq 0$: Continuous variable. The **makespan** of the schedule.

2.4 Objective Function

Minimize the makespan:

$$\min C_{\max}$$

2.5 Constraints

1. **Each job stage assigned exactly one machine:**
   Every job j at every stage s must be assigned to exactly one machine m.

$$\sum_{m=1}^{M} x_{j,s,m} = 1 \quad \forall j \in J, \forall s \in S$$

2. **Stage sequencing with transfer times:**
   The start time of a job j at stage $s+1$ must be greater than or equal to its finish time at stage s, plus the transfer time between the machines used for stage s and stage $s+1$. This constraint is activated only if job j is processed on machine m_1 for stage s and on machine m_2 for stage $s+1$.

$$t_{j,s+1} \geq t_{j,s} + p_{j,s,m_1} + T_{m_1,m_2} - BIG\_M \cdot (2 - x_{j,s,m_1} - x_{j,s+1,m_2}) \quad \forall j \in J, s \in \{1, \dots, n_s - 1\}, m_1 \in M, m_2 \in M$$

3. **Machine conflict constraints:**
   A machine can only process one job-stage combination at any given time. If two distinct job-stage combinations $((j_1, s_1), (j_2, s_2))$ are processed on the same machine m, their operations must not overlap. One must finish before the other can start. For every machine $m \in M$, and for every distinct pair of job-stage combinations $((j_1, s_1), (j_2, s_2))$:

$$t_{j_2,s_2} \geq t_{j_1,s_1} + p_{j_1,s_1,m} - BIG\_M \cdot \big((1 - y_{(j_1,s_1),(j_2,s_2),m}) + (1 - x_{j_1,s_1,m}) + (1 - x_{j_2,s_2,m})\big)$$
$$t_{j_1,s_1} \geq t_{j_2,s_2} + p_{j_2,s_2,m} - BIG\_M \cdot \big(y_{(j_1,s_1),(j_2,s_2),m} + (1 - x_{j_1,s_1,m}) + (1 - x_{j_2,s_2,m})\big)$$

   Interpretation: If $y_{(j_1,s_1),(j_2,s_2),m} = 1$, then (j_1, s_1) precedes (j_2, s_2) on machine m. If $y_{(j_1,s_1),(j_2,s_2),m} = 0$, then (j_2, s_2) precedes (j_1, s_1) on machine m. These constraints are only active if both job-stage combinations are assigned to machine m.

4. **Makespan definition:**
   The makespan $C_{\max}$ must be greater than or equal to the completion time of every job at its last stage.

$$C_{\max} \geq t_{j,n_s} + \sum_{m=1}^{M} p_{j,n_s,m} \cdot x_{j,n_s,m} \quad \forall j \in J$$

—

3 Task 2: Heuristic Solution

For many practical scheduling problems, especially as the problem size increases, solving an MILP model to optimality can become computationally prohibitive. Heuristics offer a viable alternative by providing good, though not necessarily optimal, solutions within a reasonable computation time.

3.1 Type of Heuristic: Greedy List Scheduling

We implemented a **greedy list scheduling heuristic**. This type of heuristic makes locally optimal decisions at each step without backtracking. The core idea is to always select the next operation (job-stage combination) and assign it to the machine that allows it to start earliest, considering machine availability and inter-machine transfer times.

The specific approach implemented is as follows:

1. Jobs are processed in a fixed order (e.g., Job 1, then Job 2, ..., up to Job n).

2. For each job, its stages are processed sequentially (e.g., Stage 1, then Stage 2, then Stage 3).

3. For a given job and stage, the heuristic evaluates all available machines to find which machine would allow the earliest possible start time. The earliest start time for an operation on a machine considers:

   - The time the machine becomes free from its previous task.
   - The completion time of the current job's previous stage.

- The transfer time required to move the job from the machine used for the previous stage to the currently considered machine.

4. The job-stage operation is then assigned to the machine that yields the earliest start time. The machine's availability is updated, and the job's completion time for that stage is recorded.

This "earliest available time" logic makes it a greedy choice. While simple, it often yields reasonable solutions for resource-constrained scheduling.

3.2 Heuristic Code Explanation (MATLAB)

The heuristic implementation in MATLAB follows the greedy list scheduling logic described above.

- **Initialization:**
  - `jobMachineAssignment = zeros(n_jobs, n_stages);` stores the chosen machine for each job's stage.
  - `jobStartTimes = zeros(n_jobs, n_stages);` stores the start time for each job's stage.
  - `machineAvailable = zeros(n_machines,1);` tracks the time when each machine becomes free. Initially, all machines are available at time 0.
- **Main Loop (Job and Stage Iteration):** The code iterates through each job (`for j = 1:n_jobs`) and then through each stage for that job (`for s = 1:n_stages`). This defines the fixed processing order.
- **Finding the Best Machine for Current Stage:** Inside the stage loop, for the current `(j,s)`:
  - `best_start = inf;` initializes the earliest start time found so far.
  - `best_machine = 1;` stores the machine that offers the earliest start time.
  - `prev_machine` and `prev_finish` track the machine and finish time of the previous stage for the current job, important for transfer times. For the first stage (`s=1`), they are set to zero.
  - **Machine Evaluation Loop (`for m = 1:n_machines`):** For each potential machine `m`:
    - `transfer = 0;`
    - If `s > 1` and `prev_machine > 0`, then `transfer = T(prev_machine,m);` calculates the transfer time.
    - `est = max(machineAvailable(m), prev_finish + transfer);` calculates the earliest start time on machine `m`.
    - If `est < best_start`, update `best_start` and `best_machine`.
- **Assignment and Update:** After evaluating all machines:
  - `jobMachineAssignment(j,s) = best_machine;`
  - `jobStartTimes(j,s) = best_start;`
  - `machineAvailable(best_machine) = best_start + processing_times(j,s,best_machine);`
  - `prev_machine = best_machine;` and `prev_finish = best_start + processing_times(j,s,best_machine)`

Task 3: Comparison Between Heuristic and MILP Solutions

After implementing the heuristic solution, we also implemented the MILP model to solve the scheduling problem exactly. However, since this is an NP-hard problem, running the MILP becomes computationally infeasible for large numbers of jobs, as it takes an impractical amount of time to solve.

Both methods were executed for a small-scale problem instance of (n=6) jobs to establish a baseline comparison:

Heuristic solution: The total makespan obtained was 54.00, with very low computation time (almost instantaneous).

MILP solution: The MILP model found the optimal schedule with a makespan of **28.00**, but required significantly more computation time (259.74 seconds).

Method	Makespan	Computation Time (seconds)
Heuristic	54.00	≈ 0
MILP	**28.00**	**259.74**

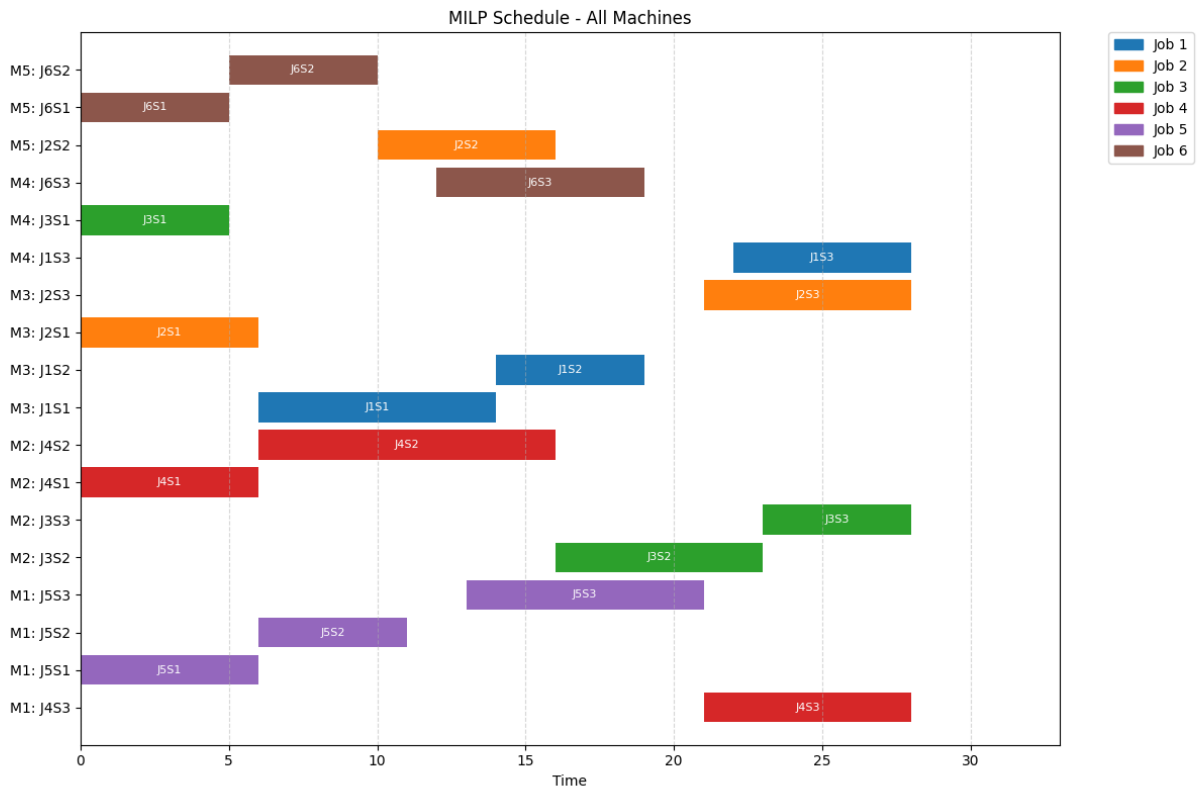

Figure 1: MILP Solution Gantt Chart for n = 6 (Makespan: 28)

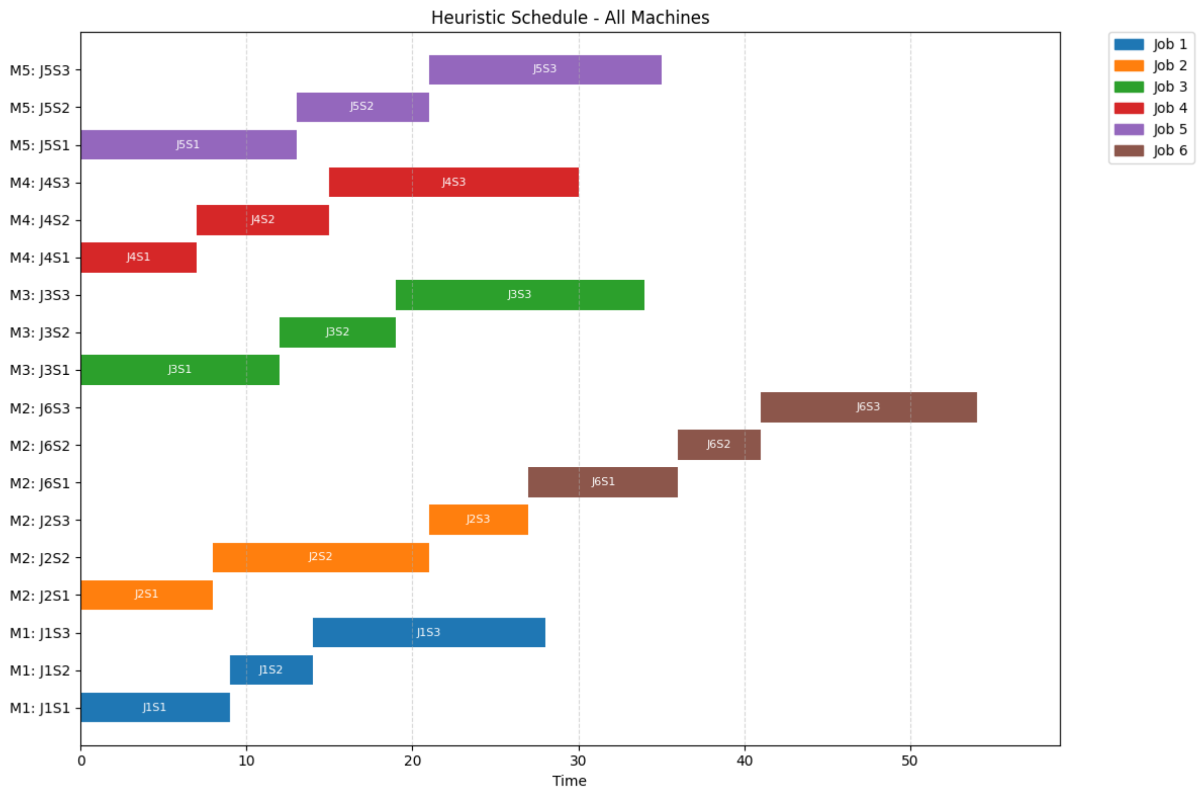

Figure 2: Heuristic Solution Gantt Chart for n = 6 (Makespan: 54)

Comparison Points

- **Makespan:** The MILP method produces an optimal solution with the lowest possible makespan, while the heuristic yields a slightly larger makespan. This means the heuristic trades off some optimality for speed.
- **Computation time:** The heuristic runs almost instantly regardless of problem size. Conversely, the MILP model's computation time grows exponentially, making it impractical beyond small instances like $n = 6$.

Heuristic Solutions for n = 10, 15, and 20

Due to the computational limitations of the MILP model, only the heuristic method was viable for larger problem sizes. The following sections detail the results for $n = 10, 15,$ and 20 jobs.

Heuristic Solution for n = 10

For a problem instance with 10 jobs, the greedy heuristic produced a schedule with a total makespan of 59.00. The detailed schedule is presented below.

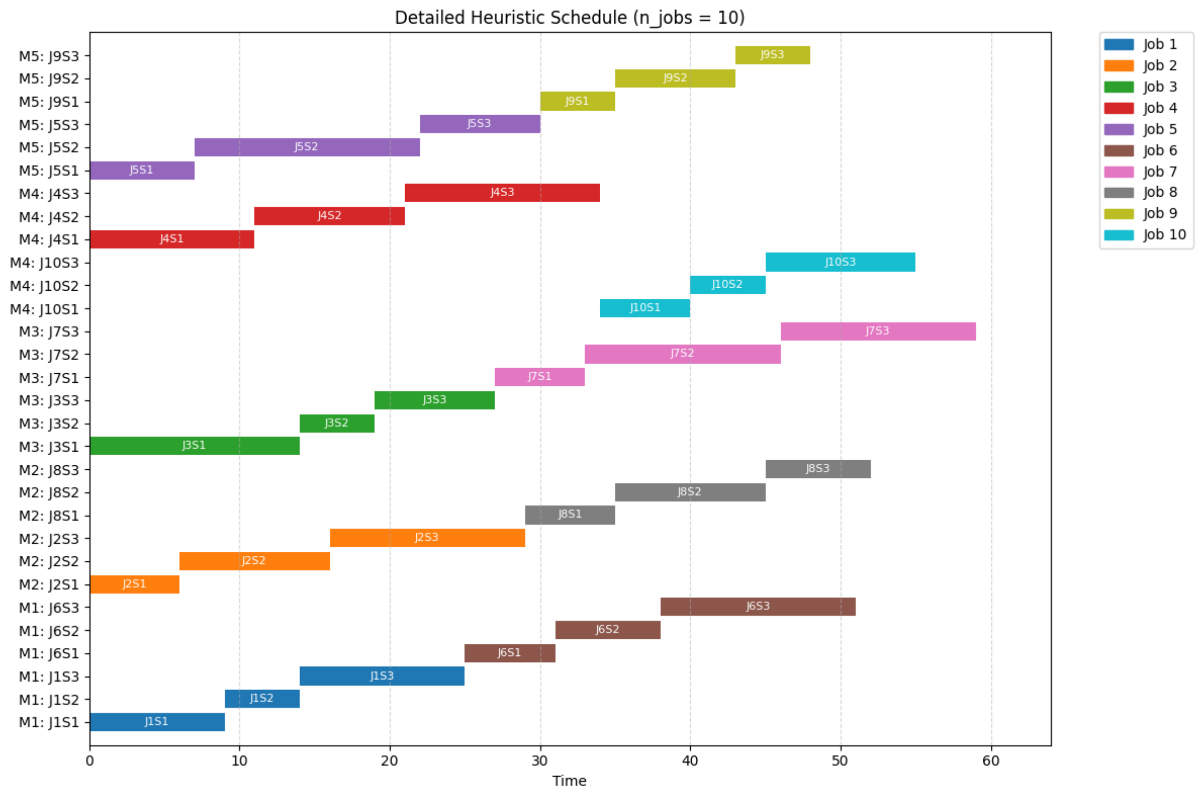

Figure 3: Heuristic Solution Gantt Chart for n = 10 (Makespan: 59)

Heuristic Solution for n = 15

With 15 jobs, the heuristic approach calculated a makespan of 107. The schedule is highlighted below.

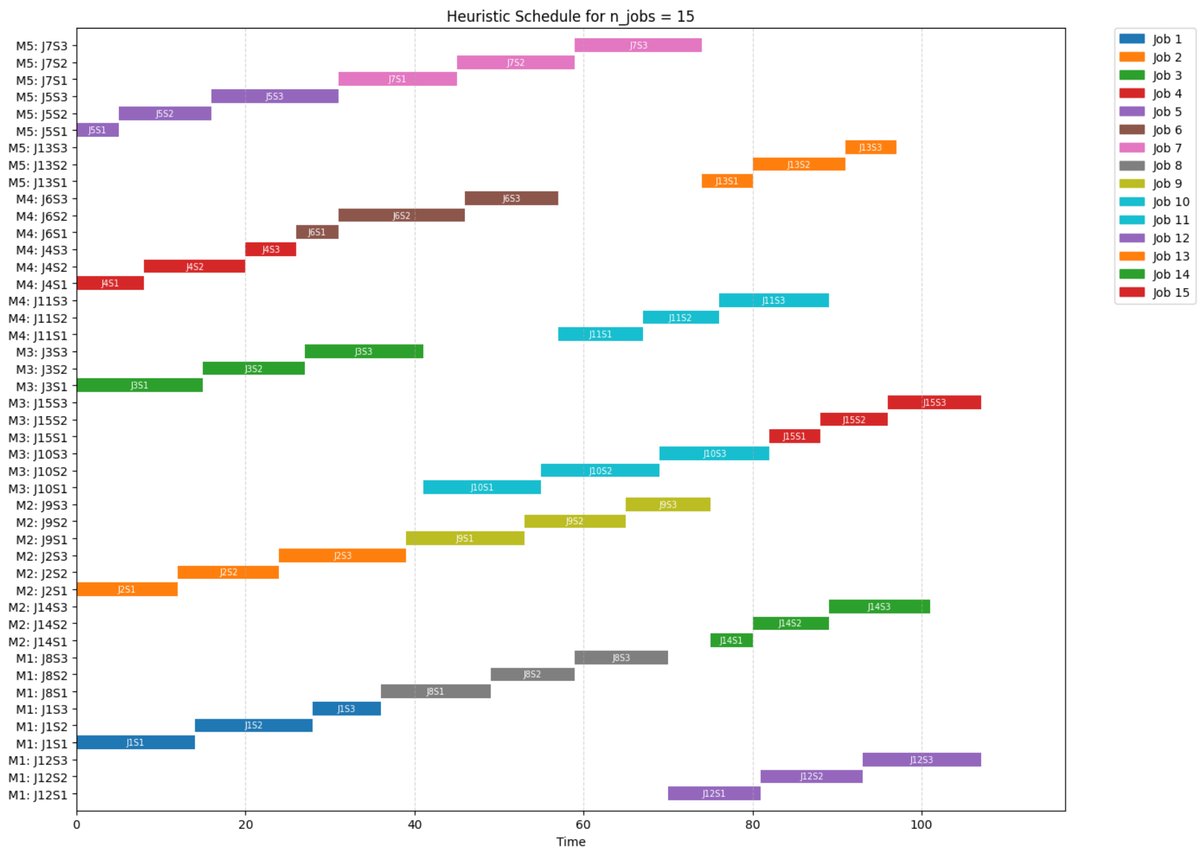

Figure 4: Heuristic Solution Gantt Chart for n = 15 (Makespan: 107)

Heuristic Solution for n = 20

For our largest test case of 20 jobs, the heuristic method determined a makespan of 136.00.

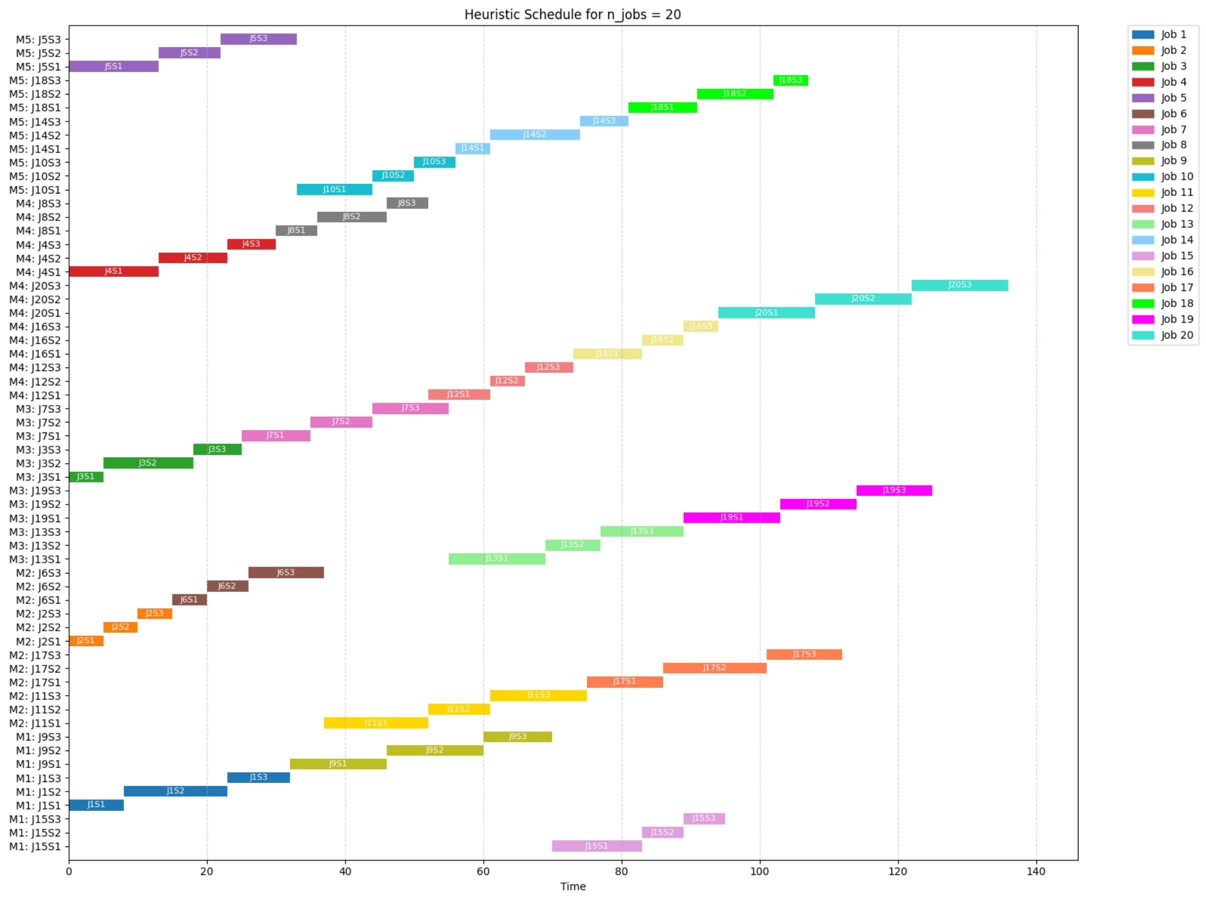

Figure 5: Heuristic Solution Gantt Chart for n = 20 (Makespan: 136)

Summary

In summary, the MILP approach guarantees optimality but suffers from scalability issues for larger problem sizes. The heuristic, while not always optimal, provides quick and feasible solutions for larger n, making it practical for real-world use cases where finding a good solution quickly is more important than finding the perfect solution over a long period. The results for $n = 10, 15,$ and 20 demonstrate the heuristic's ability to handle larger, more complex scheduling scenarios efficiently.