**Project: Framework for Solving Scheduling Problems**

This project is divided into five key components, each focusing on a different scheduling concept. The goal is to develop tools and methodologies for solving scheduling problems on single and parallel machines, using dispatching rules, meta-heuristics, and visualization through Gantt charts.

**Components and Steps**

**1. Framework for Single Machine Scheduling with Alternative Objectives**

**Objective**: Create a flexible framework to solve single machine scheduling problems with different objectives, such as minimizing makespan, total completion time, or total tardiness.

**Steps**:

* **Define Objectives**: Begin by listing possible objectives (e.g., makespan, total completion time​, total weighted completion time, maximum lateness, total tardiness, etc.).
* **Input the Job Sequence**: Accept a fixed sequence of jobs with parameters like processing times, due dates, and weights.
* **Calculate Completion Times**: Implement functions to compute the completion time of each job in the sequence.
* **Evaluate Objectives**: Calculate each objective value for the given sequence and allow users to switch between different objectives.

**Deliverable**: A framework that outputs different scheduling metrics for a given job sequence on a single machine. The code needs to provide any objective function value given a problem instance and a job sequence.

**2. Gantt Chart Generator**

**Objective**: Develop a Gantt chart generator to visualize job schedules.

**Steps**:

* **Input Format**: Design the input format for job information (start time, end time, job ID, machine ID).
* **Basic Chart Structure**: Use a plotting library (e.g., Matplotlib in Python) to represent each job as a block on the Gantt chart.
* **Add Labels and Colors**: Include job labels and optionally color-code based on different characteristics (like priority or job type).
* **Interactive Features (optional)**: For advanced students, add features like hover-over job details or filtering by job characteristics.
* **Testing**: Test with several schedules (e.g., different sequences and job durations) to verify the chart’s accuracy.

**Deliverable**: A Gantt chart generator that takes in job scheduling data and produces a visual representation of the schedule. The code needs to generate a gannt chart for any schedule provided in step 1.

**3. Application of Dispatching Rules**

**Objective**: Implement and apply dispatching rules to sequence jobs on a single machine.

**Steps**:

* **Define Common Dispatching Rules**: Introduce rules like:
  + (Weighted) Shortest Processing Time ((W)SPT)
  + Earliest Due Date (EDD)
  + Earliest Release Date (ERD)
  + Longest Processing Time (LPT)
  + etc
* **Implementation**: Write functions to sort jobs according to each rule.
* **Sequence Evaluation**: Apply the rules to different job sets and generate schedules. Evaluate each resulting sequence for different objectives (e.g., makespan, total tardiness) using the deliverable in steps one and two.

**Deliverable**: A report or tool that applies different dispatching rules to a job set and displays the resulting sequences and objective values. The code needs to generate a schedule, an objective function value and a gannt chart given the rule and instance.

**4. Application of Scheduling Rules for Parallel Machines**

**Objective**: Implement scheduling rules for a parallel machine environment. Extend the application in steps 1-3 to a parallel machine environment.

**Steps**:

* **Define the Environment**: Set up m identical machines.
* **Implement Basic Assignment Rules**: Define rules like:
  + Longest Processing Time (LPT)
  + Shortest Processing Time (SPT)
  + Wrap-Around Rule: Assign jobs sequentially to machines, looping back to the first machine when reaching the end of the machine list.

**Deliverable**: A tool that applies scheduling rules in a parallel machine setup and calculates alternative objectives for alternative dispatching rules.

**5. Application of Local Search and Meta-Heuristics for Parallel Machine and/or Flow Shop Problems**

**Objective**: Apply optimization techniques to improve schedules for parallel machines and flow shop setups. **Neighborhood generetor function**

**n-1 kadar swap yapıp neighbor**

**döndürdüğü her**

**For loop**

**Best improving**

**Initial solution üzerinden basla gec**

**Steps**:

* **Define initial solutions**
  + Use the dispatching rules in the previous steps to generate alternative initial solutions
* **Define Neighborhood functions**
  + Define alternative neighborhood functions such as
    - Swap
    - Remove and insert
    - Swap adjacent
* **Define search idea**
  + Move to first improving
  + Move to best improving
* **Define alternative acceptance criteria**
  + Accept if better
  + Threshold acceptance criteria
  + Probabilistic acceptance criteria
* **Define termination criteria**
  + Run time
  + Number of iterations
  + Number of iterations without improvement.

**Deliverable**: Implementations of local search and/or one meta-heuristic for parallel machine and/or flow shop problems, with results demonstrating improvements in scheduling metrics.