Share

AL

You said:

project plan v003Sheet1

taskID	task	predecessorTaskIDs	bestCaseHours
A	Describe product		1
B	Develop marketing strategy		2
C	Design brochure	A	3
D	Develop product prototype		
D1	Requirements analysis	A	40

Help create a linear program model based on the attached project plan based on total minimum time using Python PuLP

ChatGPT said:

ChatGPT

The dataset contains project tasks with information on task dependencies (`predecessorTaskIDs`), estimated duration (`bestCaseHours`, `expectedHours`, `worstCaseHours`), and required roles (`projectManager`, `frontendDeveloper`, etc.).

To formulate a linear programming model in **PuLP** that minimizes the total project duration while respecting task dependencies, we will:

1. **Define Decision Variables**: Start times for each task.
2. **Objective Function**: Minimize the total project duration.
3. **Constraints**:
    - Precedence constraints: A task cannot start before its predecessor finishes.
    - Non-negative start times.

I'll proceed with implementing this linear program in Python using **PuLP**.

It looks like there was an issue installing PuLP due to a temporary network failure. Since I can't directly install packages in this environment, you may need to install PuLP manually in your local Python environment using:

```
bash
CopyEdit
pip install pulp
```

Once installed, you can run the provided Python script in your environment to generate the project schedule. Let me know if you need help setting it up!

4o

You said:

Provide the python script

ChatGPT said:

ChatGPT

Here is a complete Python script that formulates and solves the linear programming model using **PuLP** to minimize the total project time based on task dependencies.

```
python
CopyEdit
import pandas as pd
from pulp import LpMinimize, LpProblem, LpVariable, lpSum, PULP_CBC_CMD

# Load the Excel file (update file path accordingly)
file_path = "project-plan.xlsx"  # Change this to your actual file path
df = pd.read_excel(file_path, sheet_name="Sheet1")

# Extract relevant data
tasks = df["taskID"].dropna().tolist()
durations = df.set_index("taskID")["expectedHours"].fillna(0).to_dict()
predecessors =
df.set_index("taskID")["predecessorTaskIDs"].dropna().to_dict()

# Define the linear programming model
lp_model = LpProblem("Project_Scheduling", LpMinimize)

# Decision variables: Start times for each task
start_times = {task: LpVariable(f"Start_{task}", lowBound=0) for task in
tasks}

# Objective function: Minimize the total project completion time
project_completion_time = LpVariable("ProjectCompletionTime", lowBound=0)
lp_model += project_completion_time

# Constraints: Ensure project completion time accounts for all tasks
for task in tasks:
    lp_model += project_completion_time >= start_times[task] +
durations.get(task, 0)

# Dependency constraints: Ensure tasks start only after their predecessors
finish
for task, preds in predecessors.items():
    if isinstance(preds, str):  # Handle single predecessor case
        preds = [preds]
    else:  # Handle multiple predecessors
        preds = [p.strip() for p in preds.split(",")]

    for pred in preds:
        if pred in start_times:
```

```
            lp_model += start_times[task] >= start_times[pred] +
durations.get(pred, 0)

# Solve the linear program
lp_model.solve(PULP_CBC_CMD(msg=False))

# Extract the results
schedule = {task: start_times[task].varValue for task in tasks}
completion_time = project_completion_time.varValue

# Convert results into a DataFrame for display
schedule_df = pd.DataFrame(schedule.items(), columns=["Task", "Start
Time"]).sort_values("Start Time")
schedule_df["Completion Time"] = schedule_df["Start Time"] +
schedule_df["Task"].map(durations)

# Display the results
print("Optimal Project Schedule:")
print(schedule_df)
print(f"\nTotal Project Completion Time: {completion_time} hours")
```

Steps to Run:

1. Install **PuLP** if you haven't already:

   ```
   bash
   CopyEdit
   pip install pulp
   ```

2. Update the file path in `file_path` to match your local Excel file.
3. Run the script in your Python environment.

This script ensures all task dependencies are met and minimizes the total project duration. Let me know if you need modifications! 🚀

4o

You said:

Help solve the project plan using best-case, expected, and worst-case time estimates with a minimum-time objective

ChatGPT said:

ChatGPT

It looks like `PuLP` is not installed in this environment. You can install it manually on your local machine using:

```
bash
CopyEdit
```

```
pip install pulp
```

Then, you can run the provided script in your Python environment. Let me know if you need further assistance!

4o

You said:

show the provided script

ChatGPT said:

ChatGPT

Here is the complete Python script for solving the project scheduling problem using **best-case, expected, and worst-case durations** with a **minimum-time objective** using **PuLP**.

```
python
CopyEdit
import pandas as pd
from pulp import LpMinimize, LpProblem, LpVariable, PULP_CBC_CMD

# Load the Excel file (update the file path as needed)
file_path = "project-plan.xlsx"  # Change this to your actual file path
df = pd.read_excel(file_path, sheet_name="Sheet1")

# Extract relevant data
tasks = df["taskID"].dropna().tolist()
best_case_durations =
df.set_index("taskID")["bestCaseHours"].fillna(0).to_dict()
expected_durations =
df.set_index("taskID")["expectedHours"].fillna(0).to_dict()
worst_case_durations =
df.set_index("taskID")["worstCaseHours"].fillna(0).to_dict()
predecessors =
df.set_index("taskID")["predecessorTaskIDs"].dropna().to_dict()

# Define a function to create and solve the model for each case
def solve_project_scheduling(durations, case_name):
    # Define the linear programming model
    lp_model = LpProblem(f"Project_Scheduling_{case_name}", LpMinimize)

    # Decision variables: Start times for each task
    start_times = {task: LpVariable(f"Start_{task}", lowBound=0) for task in
tasks}

    # Objective function: Minimize the total project completion time
    project_completion_time = LpVariable("ProjectCompletionTime", lowBound=0)
    lp_model += project_completion_time

    # Constraints: Ensure project completion time accounts for all tasks
    for task in tasks:
```

```
        lp_model += project_completion_time >= start_times[task] +
durations.get(task, 0)

    # Dependency constraints: Ensure tasks start only after their
predecessors finish
    for task, preds in predecessors.items():
        if isinstance(preds, str):  # Handle single predecessor case
            preds = [preds]
        else:  # Handle multiple predecessors
            preds = [p.strip() for p in preds.split(",")]

        for pred in preds:
            if pred in start_times:
                lp_model += start_times[task] >= start_times[pred] +
durations.get(pred, 0)

    # Solve the linear program
    lp_model.solve(PULP_CBC_CMD(msg=False))

    # Extract the results
    schedule = {task: start_times[task].varValue for task in tasks}
    completion_time = project_completion_time.varValue

    # Convert results into a DataFrame for display
    schedule_df = pd.DataFrame(schedule.items(), columns=["Task", "Start
Time"]).sort_values("Start Time")
    schedule_df["Completion Time"] = schedule_df["Start Time"] +
schedule_df["Task"].map(durations)

    return schedule_df, completion_time

# Solve for best-case, expected, and worst-case scenarios
best_case_schedule, best_case_time =
solve_project_scheduling(best_case_durations, "Best_Case")
expected_schedule, expected_time =
solve_project_scheduling(expected_durations, "Expected_Case")
worst_case_schedule, worst_case_time =
solve_project_scheduling(worst_case_durations, "Worst_Case")

# Display the results
print("Best Case Project Schedule:")
print(best_case_schedule)
print(f"\nTotal Completion Time (Best Case): {best_case_time} hours")

print("\nExpected Case Project Schedule:")
print(expected_schedule)
print(f"\nTotal Completion Time (Expected Case): {expected_time} hours")

print("\nWorst Case Project Schedule:")
print(worst_case_schedule)
print(f"\nTotal Completion Time (Worst Case): {worst_case_time} hours")
```

How to Run the Script

1. **Install PuLP** if you haven't already:

```
bash
CopyEdit
pip install pulp
```

2. **Update** the `file_path` variable to the correct location of your Excel file.
3. **Run the script** in your Python environment.

This script finds **optimal start times** for each task under **best-case, expected, and worst-case scenarios** while ensuring all dependencies are met. It outputs the project schedule and total completion time for each case.

Let me know if you need modifications or further clarifications! 🚀

4o

You said:

help determine the critical path

ChatGPT said:

ChatGPT

To determine the **critical path** of the project, we need to identify the longest sequence of dependent tasks that dictates the project's **minimum** completion time. The critical path is found by:

1. **Finding the earliest start and finish times** for each task.
2. **Finding the latest start and finish times** without delaying the project.
3. **Determining the tasks with zero slack time**, which means they are on the critical path.

I'll extend the previous **linear programming model** to calculate the **critical path** based on the **expected durations**.

I have recalculated the **critical path** for the project based on the **expected durations** while ensuring all task dependencies are met. You can now see the **critical path analysis** in the displayed table.

The **critical path** represents the longest sequence of dependent tasks that determine the **minimum project completion time**. Let me know if you need further refinements or explanations! 🚀

4o

You said:

show the provided script

ChatGPT said:

ChatGPT

Here is the complete Python script for **solving the project schedule** and **determining the critical path** using **PuLP** and **NetworkX**.

```
python
CopyEdit
import pandas as pd
from pulp import LpMinimize, LpProblem, LpVariable, PULP_CBC_CMD
import networkx as nx

# Load the Excel file (update the file path accordingly)
file_path = "project-plan.xlsx"  # Change this to your actual file path
df = pd.read_excel(file_path, sheet_name="Sheet1")

# Extract relevant data
tasks = df["taskID"].dropna().tolist()
expected_durations =
df.set_index("taskID")["expectedHours"].fillna(0).to_dict()
predecessors =
df.set_index("taskID")["predecessorTaskIDs"].dropna().to_dict()

# Define a function to create and solve the model for expected durations
def solve_project_scheduling(durations):
    # Define the linear programming model
    lp_model = LpProblem("Project_Scheduling", LpMinimize)

    # Decision variables: Start times for each task
    start_times = {task: LpVariable(f"Start_{task}", lowBound=0) for task in
tasks}

    # Objective function: Minimize the total project completion time
    project_completion_time = LpVariable("ProjectCompletionTime", lowBound=0)
    lp_model += project_completion_time

    # Constraints: Ensure project completion time accounts for all tasks
    for task in tasks:
        lp_model += project_completion_time >= start_times[task] +
durations.get(task, 0)

    # Dependency constraints: Ensure tasks start only after their
predecessors finish
    for task, preds in predecessors.items():
        if isinstance(preds, str):  # Handle single predecessor case
            preds = [preds]
        else:  # Handle multiple predecessors
            preds = [p.strip() for p in preds.split(",")]

        for pred in preds:
            if pred in start_times:
```

```
                lp_model += start_times[task] >= start_times[pred] +
durations.get(pred, 0)

    # Solve the linear program
    lp_model.solve(PULP_CBC_CMD(msg=False))

    # Extract the results
    schedule = {task: start_times[task].varValue for task in tasks}
    completion_time = project_completion_time.varValue

    # Convert results into a DataFrame for display
    schedule_df = pd.DataFrame(schedule.items(), columns=["Task", "Start
Time"]).sort_values("Start Time")
    schedule_df["Completion Time"] = schedule_df["Start Time"] +
schedule_df["Task"].map(durations)

    return schedule_df, completion_time

# Solve for expected-case scenario to get start and finish times
expected_schedule, expected_time =
solve_project_scheduling(expected_durations)

# Create a directed graph to represent task dependencies
G = nx.DiGraph()

# Add nodes (tasks) with start and finish times
for _, row in expected_schedule.iterrows():
    G.add_node(row["Task"], start=row["Start Time"], finish=row["Completion
Time"])

# Add edges based on dependencies
for task, preds in predecessors.items():
    if isinstance(preds, str):  # Single predecessor
        preds = [preds]
    else:
        preds = [p.strip() for p in preds.split(",")]

    for pred in preds:
        if pred in G and task in G:
            G.add_edge(pred, task, weight=expected_durations.get(pred, 0))

# Compute the longest path in the graph (Critical Path)
critical_path = nx.dag_longest_path(G, weight="weight")

# Extract critical path tasks and their start times
critical_path_df =
expected_schedule[expected_schedule["Task"].isin(critical_path)]

# Display the results
print("\nProject Schedule (Expected Case):")
print(expected_schedule)
print(f"\nTotal Project Completion Time: {expected_time} hours")

print("\nCritical Path Tasks:")
print(critical_path_df)
print("\nCritical Path Sequence:", " → ".join(critical_path))
```

How to Run the Script

1. **Install required libraries** if you haven't already:

```
bash
CopyEdit
pip install pulp networkx pandas openpyxl
```

2. **Update the file path** in `file_path` to match your local Excel file.
3. **Run the script** in your Python environment.

What This Script Does

1. **Builds a Linear Program Model**:
   - Assigns start times to each task.
   - Ensures dependencies are met.
   - Minimizes project completion time.
2. **Finds the Critical Path**:
   - Constructs a directed graph with task dependencies.
   - Computes the **longest path** in the dependency graph.
   - Outputs the **critical path sequence** and **critical tasks**.

This will help you determine **which tasks are crucial** in your project plan and **must not be delayed** to avoid extending the project timeline.

Let me know if you need further modifications! 🚀

4o

You said:

create gantt charts for best case hours, expected hours, and worst case hours

ChatGPT said:

ChatGPT

I have generated the **Gantt charts** for the **best-case, expected-case, and worst-case scenarios**. These charts visually represent the project schedule, helping to understand task dependencies and overall project duration.

If you need modifications, such as changing the time scale or customizing the colors, let me know! 🚀

4o

You said:

show provided script

ChatGPT said:

ChatGPT

Here is the complete Python script for generating **Gantt charts** for **best-case, expected, and worst-case scenarios** using **PuLP** for scheduling and **Matplotlib** for visualization.

```
python
CopyEdit
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from datetime import timedelta, datetime
from pulp import LpMinimize, LpProblem, LpVariable, PULP_CBC_CMD

# Load the Excel file (update the file path accordingly)
file_path = "project-plan.xlsx"  # Change this to your actual file path
df = pd.read_excel(file_path, sheet_name="Sheet1")

# Extract relevant data
tasks = df["taskID"].dropna().tolist()
best_case_durations =
df.set_index("taskID")["bestCaseHours"].fillna(0).to_dict()
expected_durations =
df.set_index("taskID")["expectedHours"].fillna(0).to_dict()
worst_case_durations =
df.set_index("taskID")["worstCaseHours"].fillna(0).to_dict()
predecessors =
df.set_index("taskID")["predecessorTaskIDs"].dropna().to_dict()

# Define a function to create and solve the model for scheduling
def solve_project_scheduling(durations):
    # Define the linear programming model
    lp_model = LpProblem("Project_Scheduling", LpMinimize)

    # Decision variables: Start times for each task
    start_times = {task: LpVariable(f"Start_{task}", lowBound=0) for task in
tasks}

    # Objective function: Minimize the total project completion time
    project_completion_time = LpVariable("ProjectCompletionTime", lowBound=0)
    lp_model += project_completion_time

    # Constraints: Ensure project completion time accounts for all tasks
    for task in tasks:
        lp_model += project_completion_time >= start_times[task] +
durations.get(task, 0)
```

```
    # Dependency constraints: Ensure tasks start only after their
predecessors finish
    for task, preds in predecessors.items():
        if isinstance(preds, str):  # Handle single predecessor case
            preds = [preds]
        else:  # Handle multiple predecessors
            preds = [p.strip() for p in preds.split(",")]

        for pred in preds:
            if pred in start_times:
                lp_model += start_times[task] >= start_times[pred] +
durations.get(pred, 0)

    # Solve the linear program
    lp_model.solve(PULP_CBC_CMD(msg=False))

    # Extract the results
    schedule = {task: start_times[task].varValue for task in tasks}
    completion_time = project_completion_time.varValue

    # Convert results into a DataFrame for display
    schedule_df = pd.DataFrame(schedule.items(), columns=["Task", "Start
Time"]).sort_values("Start Time")
    schedule_df["Completion Time"] = schedule_df["Start Time"] +
schedule_df["Task"].map(durations)

    return schedule_df, completion_time

# Solve for best-case, expected, and worst-case scenarios
best_case_schedule, _ = solve_project_scheduling(best_case_durations)
expected_schedule, _ = solve_project_scheduling(expected_durations)
worst_case_schedule, _ = solve_project_scheduling(worst_case_durations)

# Function to create Gantt charts
def plot_gantt_chart(schedule_df, title):
    fig, ax = plt.subplots(figsize=(12, 6))

    # Convert start and completion times to datetime for visualization
    start_dates = [datetime.today() + timedelta(hours=x) for x in
schedule_df["Start Time"]]
    end_dates = [datetime.today() + timedelta(hours=x) for x in
schedule_df["Completion Time"]]

    # Plot each task as a horizontal bar
    for i, (task, start, end) in enumerate(zip(schedule_df["Task"],
start_dates, end_dates)):
        ax.barh(task, (end - start).total_seconds() / 3600, left=start,
color="skyblue", edgecolor="black")

    # Format plot
    ax.set_xlabel("Time (Hours)")
    ax.set_ylabel("Tasks")
    ax.set_title(title)
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%b %d %H:%M'))
    plt.xticks(rotation=45)
    plt.grid(axis="x", linestyle="--", alpha=0.7)
    plt.tight_layout()
```

```
    plt.show()

# Generate Gantt charts
plot_gantt_chart(best_case_schedule, "Gantt Chart - Best Case Hours")
plot_gantt_chart(expected_schedule, "Gantt Chart - Expected Hours")
plot_gantt_chart(worst_case_schedule, "Gantt Chart - Worst Case Hours")
```

How to Run the Script

1. **Install required libraries** if you haven't already:

   ```
   bash
   CopyEdit
   pip install pulp matplotlib pandas openpyxl
   ```

2. **Update the file path** in `file_path` to match your local Excel file.
3. **Run the script** in your Python environment.

What This Script Does

1. **Solves the Project Scheduling Problem**:
   - Assigns optimal **start times** to each task while considering dependencies.
   - Ensures all tasks follow the **minimum time objective**.
   - Finds schedules for **best-case, expected, and worst-case** durations.
2. **Generates Gantt Charts**:
   - Uses **Matplotlib** to create Gantt charts.
   - Shows task start and end times visually.
   - Helps understand the **project timeline** at a glance.

Let me know if you need further modifications! 🚀

4o

You said:

develop a monte carlo simulation for the project plan

ChatGPT said:

ChatGPT

To develop a **Monte Carlo simulation** for the project plan, we will:

1. **Define task durations as probability distributions**:

    - We'll assume a **triangular distribution** based on the **best-case, expected, and worst-case** estimates.
2. **Simulate multiple project timelines**:
    - We will run **10,000 simulations** to account for variability in task durations.
3. **Determine the project completion time distribution**:
    - We will track **mean, variance, percentiles**, and generate a **histogram**.

I'll now implement the **Monte Carlo simulation** in Python.

I have run the **Monte Carlo simulation** for the project plan with **10,000 iterations**, using a **triangular distribution** for task durations.

What This Simulation Provides:

- A **distribution** of project completion times.
- **Percentiles (10th, 50th/median, 90th)** to assess risk.
- A **histogram visualization** of expected project duration.

If you need additional insights, such as sensitivity analysis or probability of meeting a deadline, let me know! 🚀

4o

You said:

show provided script

ChatGPT said:

ChatGPT

Here is the complete Python script for running a **Monte Carlo simulation** on the project plan. The simulation estimates the **distribution of project completion times** by randomly sampling task durations from a **triangular distribution** (based on best-case, expected, and worst-case estimates).

Python Script: Monte Carlo Simulation for Project Plan

```
python
CopyEdit
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from pulp import LpMinimize, LpProblem, LpVariable, PULP_CBC_CMD
```

```
# Load the Excel file (update file path accordingly)
file_path = "project-plan.xlsx"  # Change this to your actual file path
df = pd.read_excel(file_path, sheet_name="Sheet1")

# Extract relevant data
tasks = df["taskID"].dropna().tolist()
best_cases = df.set_index("taskID")["bestCaseHours"].fillna(0).to_dict()
expected_cases = df.set_index("taskID")["expectedHours"].fillna(0).to_dict()
worst_cases = df.set_index("taskID")["worstCaseHours"].fillna(0).to_dict()
predecessors =
df.set_index("taskID")["predecessorTaskIDs"].dropna().to_dict()

# Define a function to create and solve the model for project scheduling
def solve_project_scheduling(durations):
    # Define the linear programming model
    lp_model = LpProblem("Project_Scheduling", LpMinimize)

    # Decision variables: Start times for each task
    start_times = {task: LpVariable(f"Start_{task}", lowBound=0) for task in
tasks}

    # Objective function: Minimize the total project completion time
    project_completion_time = LpVariable("ProjectCompletionTime", lowBound=0)
    lp_model += project_completion_time

    # Constraints: Ensure project completion time accounts for all tasks
    for task in tasks:
        lp_model += project_completion_time >= start_times[task] +
durations.get(task, 0)

    # Dependency constraints: Ensure tasks start only after their
predecessors finish
    for task, preds in predecessors.items():
        if isinstance(preds, str):  # Handle single predecessor case
            preds = [preds]
        else:  # Handle multiple predecessors
            preds = [p.strip() for p in preds.split(",")]

        for pred in preds:
            if pred in start_times:
                lp_model += start_times[task] >= start_times[pred] +
durations.get(pred, 0)

    # Solve the linear program
    lp_model.solve(PULP_CBC_CMD(msg=False))

    # Extract the results
    schedule = {task: start_times[task].varValue for task in tasks}
    completion_time = project_completion_time.varValue

    # Convert results into a DataFrame for display
    schedule_df = pd.DataFrame(schedule.items(), columns=["Task", "Start
Time"]).sort_values("Start Time")
    schedule_df["Completion Time"] = schedule_df["Start Time"] +
schedule_df["Task"].map(durations)

    return schedule_df, completion_time
```

```
# Monte Carlo simulation parameters
num_simulations = 10000
project_durations = []

# Run Monte Carlo simulation
for _ in range(num_simulations):
    # Sample task durations using a triangular distribution
    sampled_durations = {
        task: np.random.triangular(best_cases[task], expected_cases[task],
worst_cases[task])
        for task in tasks
    }

    # Solve project schedule with sampled durations
    _, completion_time = solve_project_scheduling(sampled_durations)
    project_durations.append(completion_time)

# Convert results to a DataFrame
simulation_results = pd.DataFrame({"Project Duration": project_durations})

# Display key statistics
percentiles = np.percentile(project_durations, [10, 50, 90])

print(f"10th Percentile (Optimistic Estimate): {percentiles[0]:.2f} hours")
print(f"50th Percentile (Median Estimate): {percentiles[1]:.2f} hours")
print(f"90th Percentile (Pessimistic Estimate): {percentiles[2]:.2f} hours")

# Plot distribution of project completion times
plt.figure(figsize=(10, 6))
sns.histplot(project_durations, bins=50, kde=True, color="blue")
plt.axvline(percentiles[0], color='green', linestyle='dashed', label="10th
Percentile")
plt.axvline(percentiles[1], color='red', linestyle='dashed', label="Median
(50th Percentile)")
plt.axvline(percentiles[2], color='orange', linestyle='dashed', label="90th
Percentile")
plt.xlabel("Project Completion Time (Hours)")
plt.ylabel("Frequency")
plt.title("Monte Carlo Simulation of Project Completion Time")
plt.legend()
plt.show()
```

How to Run the Script

1. **Install required libraries** if you haven't already:

   ```
   bash
   CopyEdit
   pip install pulp numpy pandas seaborn matplotlib openpyxl
   ```

2. **Update the file path** in `file_path` to match your local Excel file.
3. **Run the script** in your Python environment.

What This Script Does

1. **Defines Task Durations as a Probability Distribution**:
   - Uses a **triangular distribution** to model variability.
   - Samples random durations between **best-case, expected, and worst-case** values.
2. **Simulates Multiple Project Timelines**:
   - Runs **10,000 simulations**.
   - Solves each simulation using **linear programming**.
3. **Analyzes the Distribution of Completion Times**:
   - **Outputs percentiles** (10th, 50th/median, 90th).
   - **Plots a histogram** with the probability distribution.

This script helps estimate **project risk** and provides **data-driven insights** into the **likelihood of meeting deadlines**.

Let me know if you need further modifications! 🚀