SYSC 4906D
Assignment 4

Ben Mostafa 101192825
Alec Tratnik 101220933

1. Code

```
import math
import random
import matplotlib.pyplot as plt

# Use ggplot style for plotting
plt.style.use("ggplot")

##############################
# GLOBAL PARAMETERS
##############################

alpha = 0.5  # weighting for cost = alpha*delay + (1-alpha)*energy
NUM_STEPS = 120

##############################
# 1. DEFINE 20 TASKS
##############################

PREDEFINED_20 = [
    {
        "Tlocal": 5.0,  "Elocal": 6.0,
        "Tcomm": 1.0,   "Tedge": 1.0,
        "Ecomm": 1.0
    },
    {
        "Tlocal": 6.0,  "Elocal": 5.5,
        "Tcomm": 1.2,   "Tedge": 1.0,
        "Ecomm": 1.0
    },
    {
        "Tlocal": 5.0,  "Elocal": 6.0,
        "Tcomm": 0.8,   "Tedge": 1.0,
        "Ecomm": 1.2
    },
    {
        "Tlocal": 5.5,  "Elocal": 5.0,
        "Tcomm": 1.0,   "Tedge": 0.8,
        "Ecomm": 1.0
    },
    {
        "Tlocal": 6.0,  "Elocal": 7.0,
```

```
        "Tcomm": 1.0,   "Tedge": 1.0,
        "Ecomm": 1.0
    },
    {
        "Tlocal": 10,  "Elocal": 1.1,
        "Tcomm": 0.1,   "Tedge": 0.9,
        "Ecomm": 11.1
    },
    {
        "Tlocal": 11.8,  "Elocal": 1.2,
        "Tcomm": 0.5,   "Tedge": 0.5,
        "Ecomm": 11.3
    },
    {
        "Tlocal": 11.5,  "Elocal": 2.0,
        "Tcomm": 1.0,   "Tedge": 1.0,
        "Ecomm": 13.0
    },
    {
        "Tlocal": 11.2,  "Elocal": 1.5,
        "Tcomm": 1.0,   "Tedge": 1.0,
        "Ecomm": 13.4
    },
    {
        "Tlocal": 11.6,  "Elocal": 2.0,
        "Tcomm": 1.1,   "Tedge": 1.0,
        "Ecomm": 13.1
    },
    {
        "Tlocal": 1.0,  "Elocal": 10.5,
        "Tcomm": 11.5,   "Tedge": 0.5,
        "Ecomm": 1.0
    },
    {
        "Tlocal": 1.0,  "Elocal": 10.2,
        "Tcomm": 11.0,   "Tedge": 0.1,
        "Ecomm": 1.0
    },
    {
        "Tlocal": 1.5,  "Elocal": 10.0,
        "Tcomm": 10.5,   "Tedge": 1.0,
        "Ecomm": 1.5
    },
```

```
    {
        "Tlocal": 1.5,  "Elocal": 11.0,
        "Tcomm": 11.0,   "Tedge": 0.5,
        "Ecomm": 1.2
    },
    {
        "Tlocal": 1.3,  "Elocal": 9.3,
        "Tcomm": 11.2,   "Tedge": 0.1,
        "Ecomm": 1.4
    },
    {
        "Tlocal": 2.0,  "Elocal": 2.0,
        "Tcomm": 2.2,   "Tedge": 9.0,
        "Ecomm": 9.5
    },
    {
        "Tlocal": 2.2,  "Elocal": 2.5,
        "Tcomm": 2.2,   "Tedge": 9.0,
        "Ecomm": 9.8
    },
    {
        "Tlocal": 2.0,  "Elocal": 2.5,
        "Tcomm": 2.0,   "Tedge": 9.0,
        "Ecomm": 9.0
    },
    {
        "Tlocal": 2.5,  "Elocal": 2.5,
        "Tcomm": 2.8,   "Tedge": 9.0,
        "Ecomm": 9.4
    },
    {
        "Tlocal": 2.1,  "Elocal": 2.3,
        "Tcomm": 2.2,   "Tedge": 9.0,
        "Ecomm": 9.3
    }
]

##############################
# 2. Weighted Cost for local/offload
##############################

def cost_local(task):
    return alpha*task["Tlocal"] + (1-alpha)*task["Elocal"]
```

```python
def cost_offload(task):
    return alpha*(task["Tcomm"] + task["Tedge"]) + (1-alpha)*task["Ecomm"]

##############################
# 3. 4-step feasibility for local/offload
##############################

def valid_move_p1(last_moves, new_move):
    """
    - No more than 2 'local' in any 4 consecutive
    - No more than 3 'offload' in any 4 consecutive
    """
    window = (last_moves[-3:] + [new_move])
    if window.count("local") > 2:
        return False
    if window.count("offload") > 3:
        return False
    return True

##############################
# 4. Generate the 120-step scenario
##############################

def generate_scenario_120():
    """
    Creates a list of 120 tasks, each a random pick from PREDEFINED_20.
    """
    tasks = []
    for _ in range(NUM_STEPS):
        task_dict = random.choice(PREDEFINED_20)
        tasks.append(task_dict)
    return tasks

##############################
# 5. Minimax & Random approaches
##############################

def minimax_decision(tasks, last_moves, depth=4):
    """
    Minimax strategy for selecting the best move within a limited lookahead
depth.
    """
```

```
    def minimax(index, moves, is_maximizing):
        if index >= NUM_STEPS or len(moves) >= depth:
            return 0  # Base case: No more moves to evaluate

        task = tasks[index]
        feasible_moves = []
        if valid_move_p1(moves, "local"):
            feasible_moves.append(("local", cost_local(task)))
        if valid_move_p1(moves, "offload"):
            feasible_moves.append(("offload", cost_offload(task)))

        if not feasible_moves:
            return 0  # No valid move

        if is_maximizing:
            return max(minimax(index + 1, moves + [move], not
is_maximizing) + cost for move, cost in feasible_moves)
        else:
            return min(minimax(index + 1, moves + [move], not
is_maximizing) + cost for move, cost in feasible_moves)

    # Apply minimax to choose the best move
    best_move = "local"
    best_cost = float("inf")
    for move, cost in [("local", cost_local(tasks[0])), ("offload",
cost_offload(tasks[0]))]:
        if valid_move_p1(last_moves, move):
            move_cost = minimax(1, last_moves + [move], False) + cost
            if move_cost < best_cost:
                best_cost = move_cost
                best_move = move

    return best_move, best_cost

def run_minimax(tasks):
##############################
# Add your Player code here. The code should return two values: moves,
total_cost
##############################
    last_moves = []
    moves = []
    total_cost = 0.0
```

```python
    for i in range(NUM_STEPS):
        chosen_move, step_cost = minimax_decision(tasks[i:], last_moves)
        moves.append(chosen_move)
        last_moves.append(chosen_move)
        if len(last_moves) > 3:
            last_moves.pop(0)  # Maintain last 3 moves for constraints
        total_cost += step_cost

    return moves, total_cost

def run_random_player(tasks):
    """
    For each task, pick local/offload at random if feasible,
    else default to local with cost=0 if no feasible moves.
    Return (moves, totalCost).
    """
    last3moves = []
    moves = []
    total_cost = 0.0
    for tdict in tasks:
        c_loc = cost_local(tdict)
        c_off= cost_offload(tdict)
        feasible=[]
        if valid_move_p1(last3moves, "local"):
            feasible.append(("local", c_loc))
        if valid_move_p1(last3moves, "offload"):
            feasible.append(("offload", c_off))

        if not feasible:
            chosen = "local"
            step_cost = 0.0
        else:
            chosen, step_cost = random.choice(feasible)
        moves.append(chosen)
        last3moves.append(chosen)
        total_cost += step_cost

    return moves, total_cost

##############################
# 6. Stepwise evaluation
##############################
```

```python
def evaluate_stepwise(tasks, moves):
    """
    Return stepwise cumulative delay, energy, cost arrays for plotting.
    """
    delay_arr = []
    energy_arr= []
    cost_arr  = []
    cum_delay=0.0
    cum_energy=0.0
    cum_cost=0.0

    for i, tdict in enumerate(tasks):
        if moves[i]=="local":
            step_delay = tdict["Tlocal"]
            step_energy= tdict["Elocal"]
        else: # offload
            step_delay= tdict["Tcomm"]+ tdict["Tedge"]
            step_energy= tdict["Ecomm"]
        cum_delay += step_delay
        cum_energy+= step_energy
        step_cost = alpha*step_delay + (1-alpha)*step_energy
        cum_cost += step_cost

        delay_arr.append(cum_delay)
        energy_arr.append(cum_energy)
        cost_arr.append(cum_cost)

    return delay_arr, energy_arr, cost_arr

##############################
# 7. Plot
##############################

def plot_results(dG, eG, cG, dR, eR, cR):
    steps = range(1, NUM_STEPS+1)

    # 1) Delay
    plt.figure(figsize=(8,5))
    plt.plot(steps, dG, label="Minimax")
    plt.plot(steps, dR, label="Random")
    plt.title("Cumulative Delay (120 steps)")
    plt.xlabel("Step")
    plt.ylabel("Delay")
```

```
    plt.legend()
    plt.show()

    # 2) Energy
    plt.figure(figsize=(8,5))
    plt.plot(steps, eG, label="Minimax")
    plt.plot(steps, eR, label="Random")
    plt.title("Cumulative Energy (120 steps)")
    plt.xlabel("Step")
    plt.ylabel("Energy")
    plt.legend()
    plt.show()

    # 3) Cost
    plt.figure(figsize=(8,5))
    plt.plot(steps, cG, label="Minimax")
    plt.plot(steps, cR, label="Random")
    plt.title(f"Cumulative Cost (alpha={alpha})")
    plt.xlabel("Step")
    plt.ylabel("Cost")
    plt.legend()
    plt.show()

##############################
# 8. Print table
##############################

def print_3row_table(tasks, movesG, movesR):
    """
    Row 0 => The index of each task in PREDEFINED_20
    Row 1 => Minimax's local/off decisions
    Row 2 => Random's local/off decisions
    """
    print("\nTABLE (3 rows => tasks, minimax, random):\n")

    # tasks row
    print("Tasks:    ", end="")
    for tdict in tasks:
        idx = PREDEFINED_20.index(tdict)
        print(f"T{idx:2}", end=" ")
    print()

    # Minimax row
```

```python
    print("Minimax:    ", end="")
    for mv in movesG:
        print(f"{mv:6}", end=" ")
    print()

    # Random row
    print("Random:    ", end="")
    for mv in movesR:
        print(f"{mv:6}", end=" ")
    print("\n")

##############################
# MAIN
##############################

def main():
    random.seed(42)

    # 1) Generate scenario of 120 tasks
    tasks_120 = generate_scenario_120()

    # 2) Minimax approach
    movesMinimax, costMinimax = run_minimax(tasks_120)
    print(f"Minimax final cost: {costMinimax:.2f}")

    # 3) Random approach
    movesRandom, costRandom = run_random_player(tasks_120)
    print(f"Random final cost: {costRandom:.2f}")

    # 4) Evaluate stepwise
    dG, eG, cG = evaluate_stepwise(tasks_120, movesMinimax)
    dR, eR, cR = evaluate_stepwise(tasks_120, movesRandom)

    # 5) Print table
    print_3row_table(tasks_120, movesMinimax, movesRandom)

    # 6) Plot
    plot_results(dG, eG, cG, dR, eR, cR)

if __name__=="__main__":
    main()
```

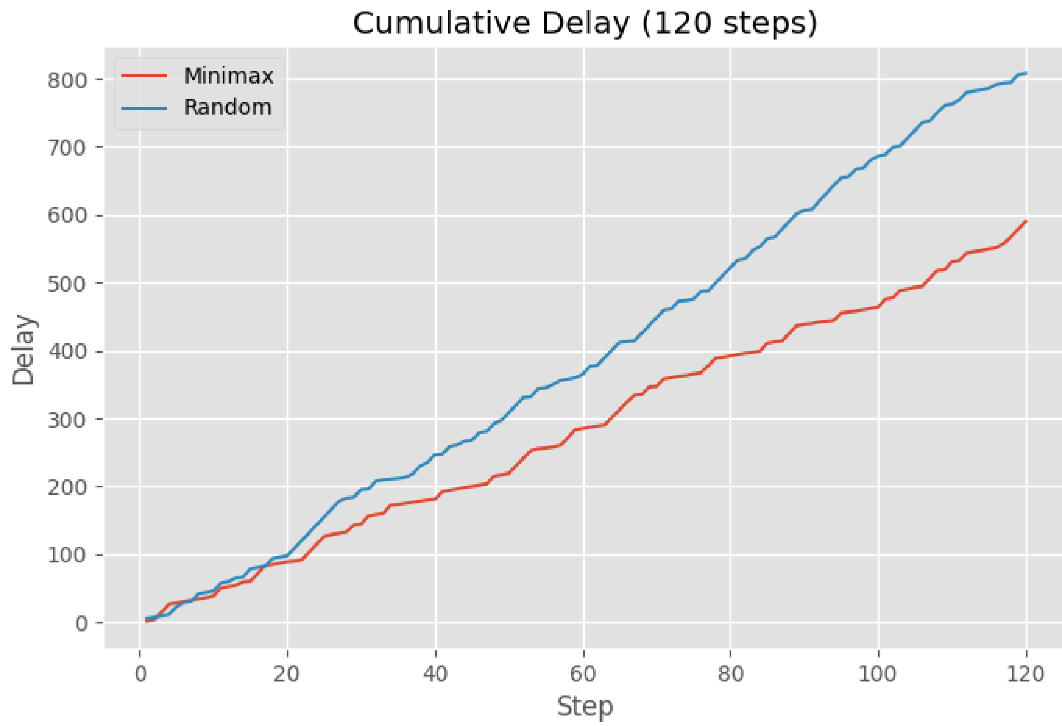
2. Program Output

Scenario 1 (α = 0.5)

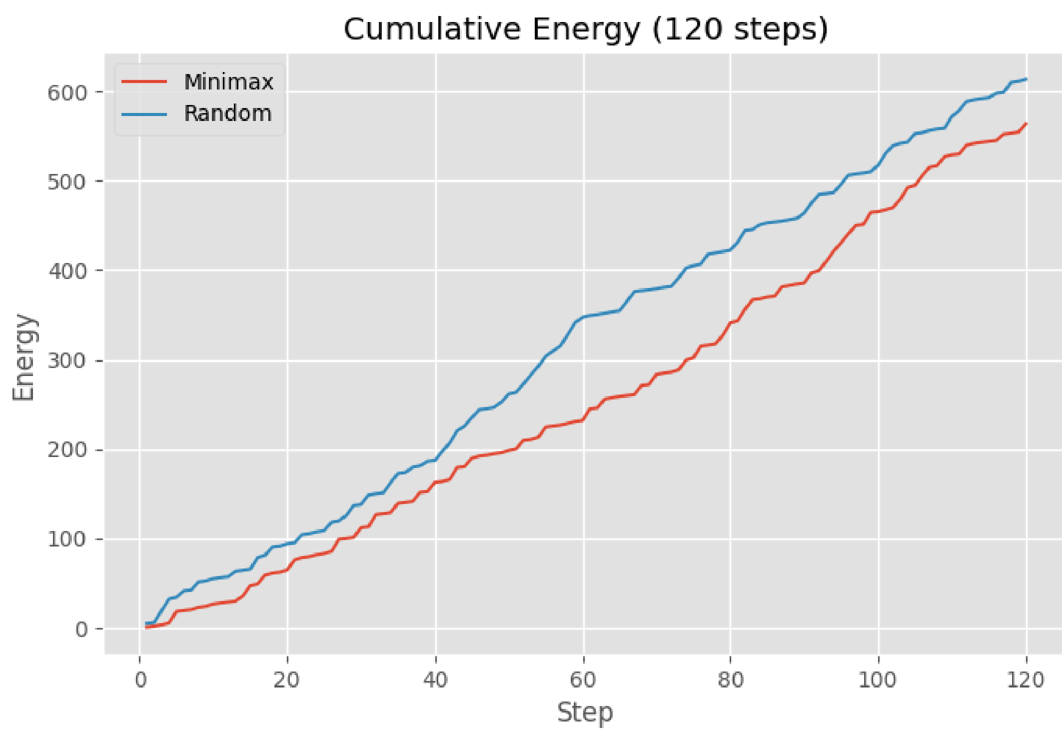
Minimax final cost: 613.30
Random final cost: 710.70

TABLE (3 rows => tasks, minimax, random):

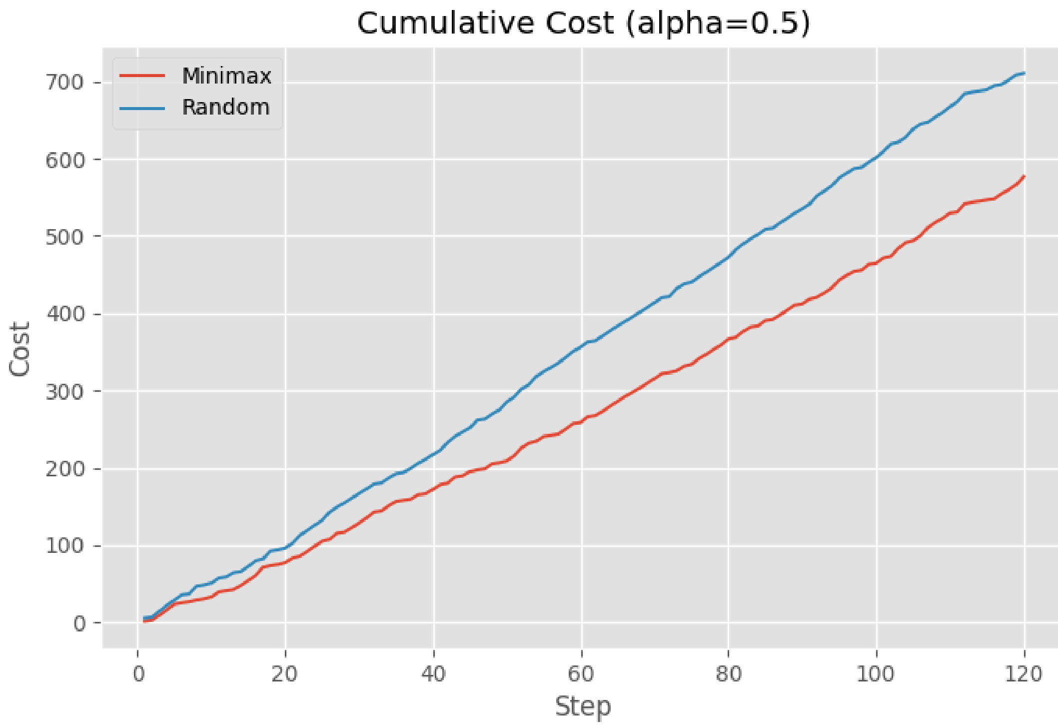
Tasks: T 3 T 0 T 8 T 7 T 7 T 4 T 3 T17 T 2 T18 T13 T 1 T 0 T 2 T 6 T 7 T16 T19 T 0 T17 T 6 T17 T13 T 7 T14 T18 T 8 T 0 T 5 T13 T10 T 8 T 4 T 6 T10 T 3 T 2 T12 T 3 T11 T11 T19 T 8 T 1 T14 T17 T 3 T12 T 2 T17 T 9 T19 T11 T18 T 6 T 2 T 1 T 7 T 9 T 2 T 7 T 3 T12 T 8 T14 T11 T 5 T11 T11 T 6 T 8 T 2 T19 T 5 T17 T 7 T 5 T14 T12 T 8 T17 T 7 T10 T 1 T 7 T 1 T10 T12 T 8 T 2 T 6 T18 T10 T 6 T15 T12 T14 T 4 T 8 T 4 T 7 T17 T17 T 8 T18 T13 T18 T12 T11 T 7 T 4 T16 T15 T 2 T 1 T 3 T 4 T 5 T13 T19
Minimax: offload offload local local offload offload offload local offload local offload offload offload local offload local offload local offload local offload local offload local offload local offload offload local local offload offload offload local local offload offload local offload local offload local offload offload local local offload offload offload local local offload offload local offload offload offload local local offload offload offload local local offload offload local local offload offload local offload local offload local offload local offload local offload local offload local offload local offload local offload local offload offload local local offload offload local local offload offload offload local local offload offload local local offload offload local local offload offload local offload offload offload local local offload offload
Random: local offload offload offload local local offload offload offload local offload offload local offload local offload local offload offload local local offload offload local offload offload local local offload offload local local offload offload local offload local offload local offload local offload offload local local offload offload offload local offload local offload local offload offload local local offload offload local local offload offload local offload local offload offload offload local local offload offload offload local local offload offload offload local offload offload offload local local offload offload offload local local offload offload offload local offload local offload offload local local offload offload local local offload offload local offload offload offload local offload local offload offload local offload offload offload local



Cumulative Delay (120 steps)
Minimax
Random
Delay
Step
0
100
200
300
400
500
600
700
800
0
20
40
60
80
100
120



Cumulative Energy (120 steps)
Minimax
Random
Energy
Step
0
100
200
300
400
500
600
0
20
40
60
80
100
120



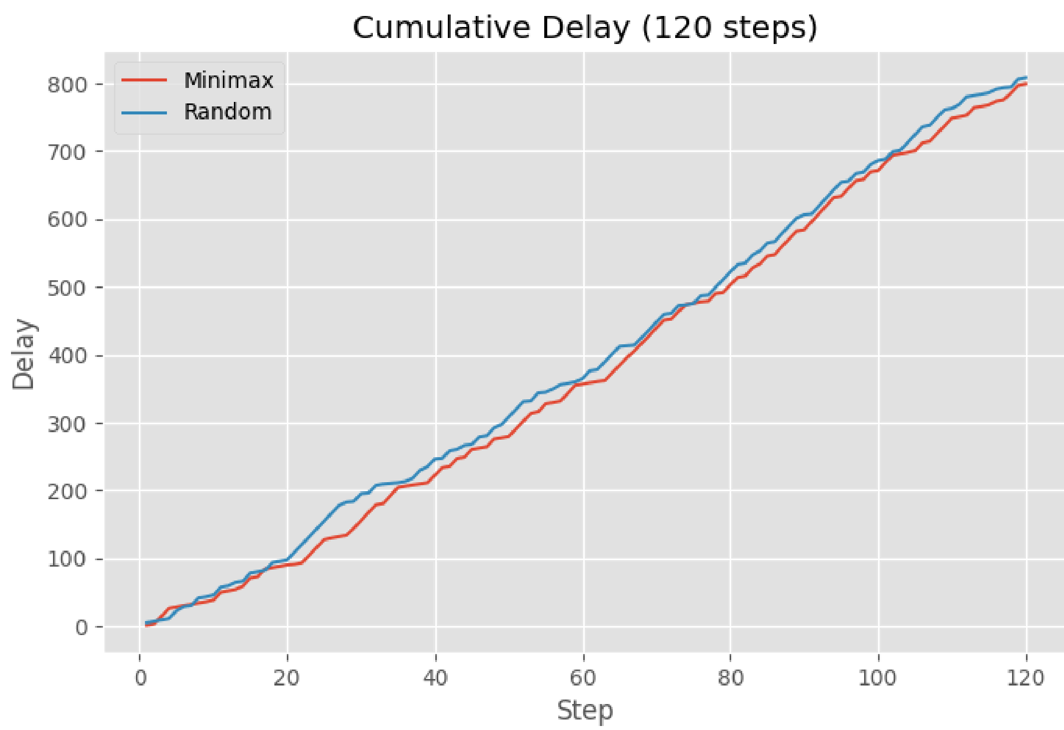
Scenario 2 (α = 0.1)

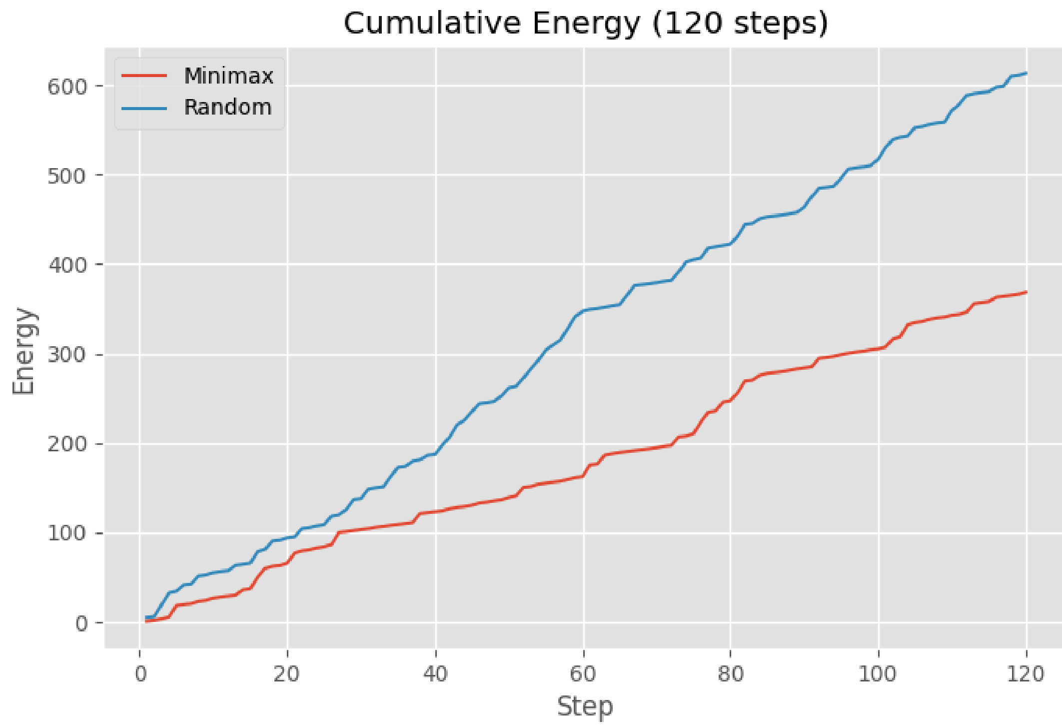
Minimax final cost: 445.28
Random final cost: 632.78

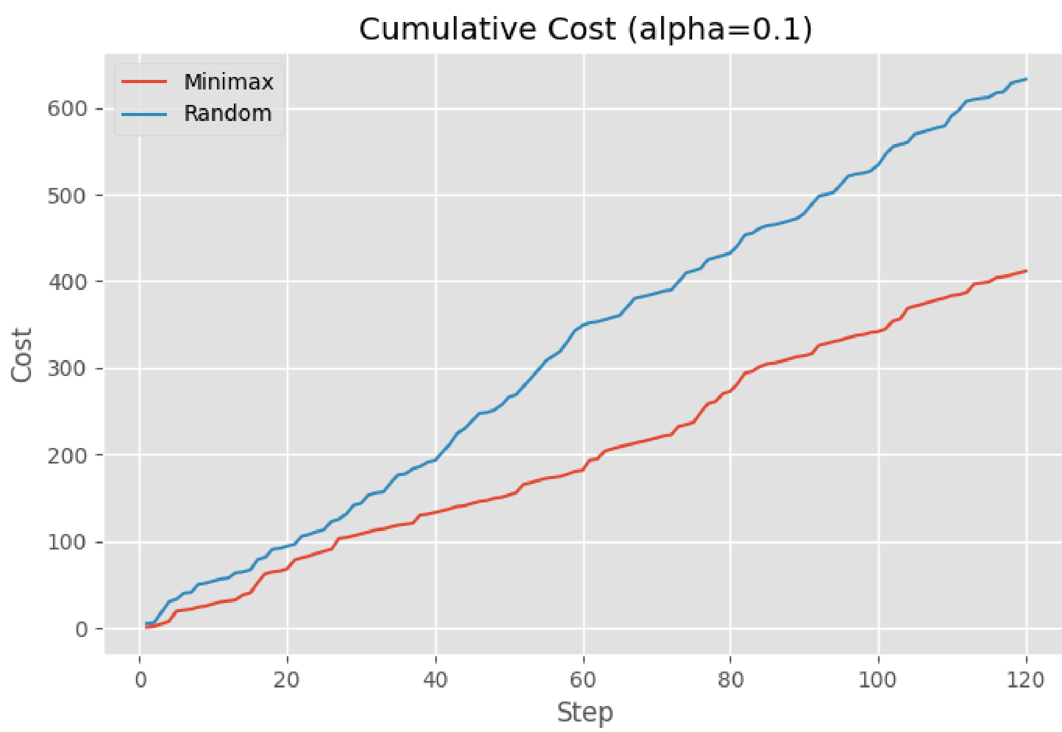
TABLE (3 rows => tasks, minimax, random):

Tasks: T 3 T 0 T 8 T 7 T 7 T 4 T 3 T17 T 2 T18 T13 T 1 T 0 T 2 T 6 T 7 T16 T19 T 0 T17 T 6 T17 T13 T 7 T14 T18 T 8 T 0 T 5 T13 T10 T 8 T 4 T 6 T10 T 3 T 2 T12 T 3 T11 T11 T19 T 8 T 1 T14 T17 T 3 T12 T 2 T17 T 9 T19 T11 T18 T 6 T 2 T 1 T 7 T 9 T 2 T 7 T 3 T12 T 8 T14 T11 T 5 T11 T11 T 6 T 8 T 2 T19 T 5 T17 T 7 T 5 T14 T12 T 8 T17 T 7 T10 T 1 T 7 T 1 T10 T12 T 8 T 2 T 6 T18 T10 T 6 T15 T12 T14 T 4 T 8 T 4 T 7 T17 T17 T 8 T18 T13 T18 T12 T11 T 7 T 4 T16 T15 T 2 T 1 T 3 T 4 T 5 T13 T19
Minimax: offload offload local local offload offload offload local offload local offload offload offload local local offload offload local offload local offload local offload local offload local offload offload local offload offload local offload local offload offload offload local offload offload offload local local offload offload local offload offload offload local local offload offload local local offload offload local local offload offload offload local local offload offload local offload offload local local offload offload local local offload offload offload local local offload offload offload local local offload offload offload local offload local offload offload local local offload offload offload local offload local offload local offload local offload local offload offload local offload local offload offload offload local offload local offload local

Random: local offload offload offload local local offload offload offload local offload offload local offload local offload local offload offload local local offload offload local offload offload local local offload offload local local offload offload local offload local offload local offload local offload offload local local offload offload offload local offload local offload local offload offload local local offload offload local local offload offload local offload local offload offload offload local local offload offload offload local local offload offload offload local offload offload offload local local offload offload offload local local offload offload offload local offload local offload offload local local offload offload local local offload offload local offload offload offload local offload local offload offload local offload offload offload local




Cumulative Energy (120 steps)
Minimax
Random
Energy
Step
0
100
200
300
400
500
600
0
20
40
60
80
100
120

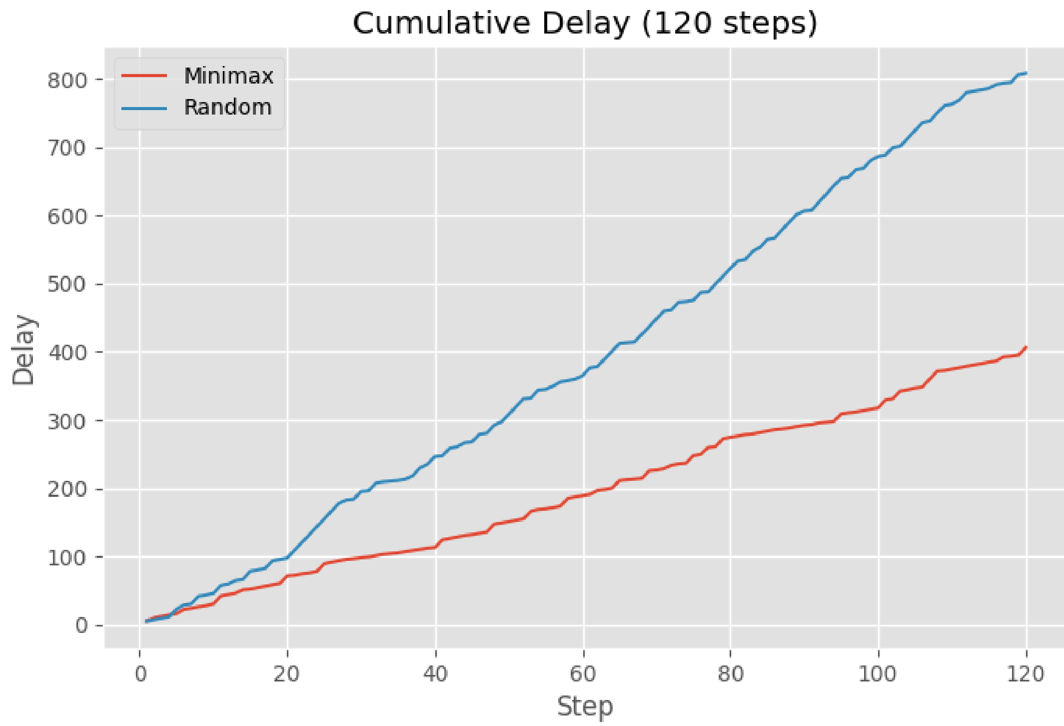

Cumulative Cost (alpha=0.1)
Minimax
Random
Cost
Step
0
100
200
300
400
500
600
0
20
40
60
80
100
120

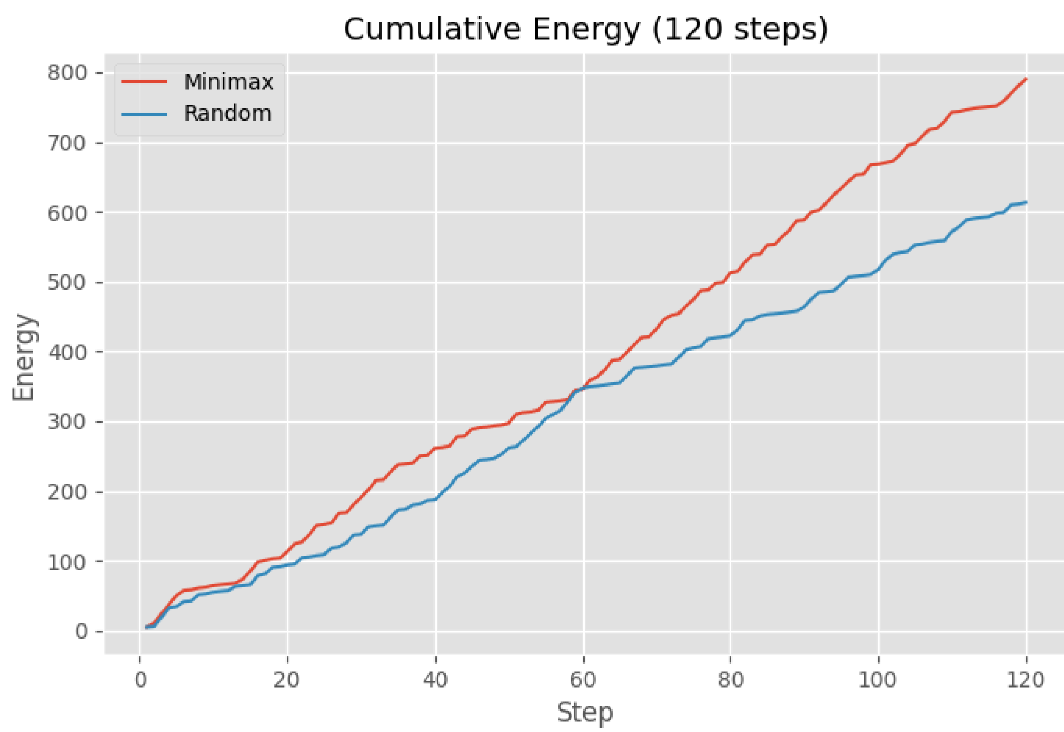
Scenario 3 (α = 0.9)

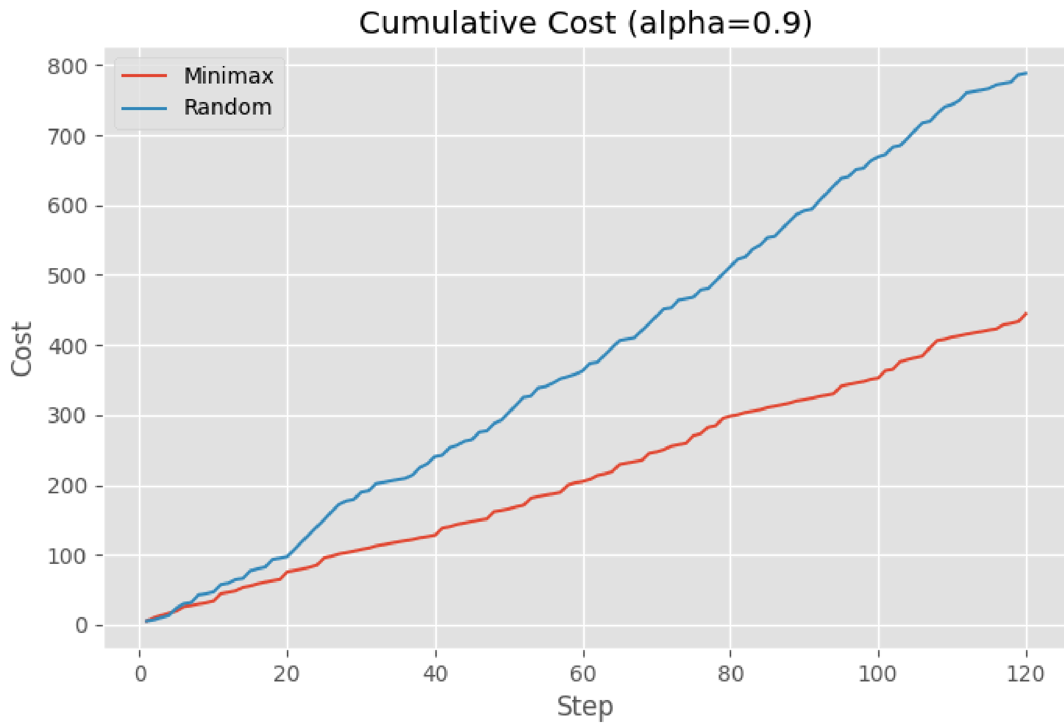
Minimax final cost: 465.33
Random final cost: 788.62

TABLE (3 rows => tasks, minimax, random):

Tasks: T 3 T 0 T 8 T 7 T 7 T 4 T 3 T17 T 2 T18 T13 T 1 T 0 T 2 T 6 T 7 T16 T19 T 0 T17 T 6 T17 T13 T 7 T14 T18 T 8 T 0 T 5 T13 T10 T 8 T 4 T 6 T10 T 3 T 2 T12 T 3 T11 T11 T19 T 8 T 1 T14 T17 T 3 T12 T 2 T17 T 9 T19 T11 T18 T 6 T 2 T 1 T 7 T 9 T 2 T 7 T 3 T12 T 8 T14 T11 T 5 T11 T11 T 6 T 8 T 2 T19 T 5 T17 T 7 T 5 T14 T12 T 8 T17 T 7 T10 T 1 T 7 T 1 T10 T12 T 8 T 2 T 6 T18 T10 T 6 T15 T12 T14 T 4 T 8 T 4 T 7 T17 T17 T 8 T18 T13 T18 T12 T11 T 7 T 4 T16 T15 T 2 T 1 T 3 T 4 T 5 T13 T19
Minimax: local local offload offload offload local offload local offload local offload offload offload local offload offload local local offload offload offload local local offload offload local offload offload offload local local offload offload offload local offload offload local offload local offload local offload offload local local offload offload offload local offload local offload local offload offload offload local offload offload offload local local offload
offload local offload local offload offload offload local local offload offload offload local local offload offload local offload local offload offload offload local local offload offload offload local local offload offload local local offload offload offload local local offload offload local local offload offload local offload offload local local offload offload offload local offload local offload
Random: local offload offload offload local local offload offload offload local offload offload local offload local offload local offload offload local local offload offload local offload offload local local offload offload local local offload offload local offload local offload local offload local offload offload local local offload offload offload local offload local offload local offload offload local local offload offload local local offload offload local offload local offload offload offload local local offload offload offload local local offload offload offload local offload offload offload local local offload offload offload local local offload offload offload local offload local offload offload local local offload offload local local offload offload local offload offload offload local offload local offload offload local offload offload offload local


Cumulative Delay (120 steps)
Minimax
Random
Delay
Step
0
100
200
300
400
500
600
700
800
0
20
40
60
80
100
120


Cumulative Energy (120 steps)
Minimax
Random
Energy
Step
0
100
200
300
400
500
600
700
800
0
20
40
60
80
100
120



3. Discussion

The weighting factor α determines whether the system prioritizes delay or energy consumption when making task allocation decisions. A balanced weighting (α = 0.5) treats both equally, while a higher or lower α shifts the focus toward one over the other.

Scenario 1 (α = 0.5): A Balanced Approach

With equal weighting on delay and energy, Minimax takes a more balanced approach, neither favoring offloading nor local execution too heavily. This results in a final cost of 613.30, which sits between the other two scenarios. Since there is no strong preference for minimizing either delay or energy, the allocation of tasks is more evenly distributed between local execution and offloading. The random strategy, by contrast, results in a higher cost (710.70) due to its lack of an optimization strategy.

Scenario 2 (α = 0.1): Energy is the Priority

Here, energy is weighted at 0.9, while delay is only 0.1. Since energy consumption is the main concern, Minimax offloads more tasks to reduce energy use. This leads to a final cost of 445.28, much lower than the random strategy's 632.78. By strategically offloading tasks, Minimax significantly reduces energy consumption, demonstrating its ability to adapt based on the weighting.

Scenario 3 (α = 0.9): Delay Takes Priority

This is the opposite of Scenario 2, delay is now the priority (0.9), while energy has minimal influence (0.1). To reduce delay, Minimax executes more tasks locally instead of offloading them, avoiding the time lost in sending tasks to external resources. While this improves delay performance, it leads to higher energy consumption, which is why the final cost is 465.33, which is higher than Scenario 2 but still much lower than the random strategy (788.62).

Overall, Minimax proves to be a more efficient decision-making strategy, consistently outperforming random allocation by adjusting its approach based on the given weighting.