Task scheduling optimization problem

Multi-objective optimization

BARBOSA Mathias & THIRÉ Mael

Content

1 Problem & Objective

Problem

- Task assignment problem in a computer
- Fog-cloud computing environment

Objective

- Assigning tasks to a machine located either in the fog or in the cloud while minimizing makespan and cost

2 Mathematical formulation

A Decision variable et constraints

Decision variables

- Xij, where i is the task number and j is the machine number (binary because if task i is assigned to machine j, 1; otherwise, 0).

Constraints

h

- Each task is associated with a single and unique VM. $\longrightarrow$ $\Sigma_j X_{ij} = 1, \forall i$
- Each VM has a capacity constraint (Wi is the workload of task i and Cj is the processing capacity of VM j in 10s.) $\longrightarrow$ $\Sigma_i W_i \cdot X_{ij} \leq C_j, \forall j$

2 Mathematical formulation

B Objective functions

- Minimize Makespan: minimize the total time taken to complete all tasks → $\text{Minimize } \Sigma_i \Sigma_j T_{ij} \cdot X_{ij}$

- Minimize cost: minimize the total cost of VMs → $\text{Minimize } \Sigma_i \Sigma_j (c_{ijp} + c_{ijm} + c_{ijb}) * X_{ij}$

3 Metaheuristic function : Bat Algorithm

Population Initialization

- Generating an initial population of bats, where each bats represents a potential solution according to its position

Parameters Initialization

- Defining parameters such as the number of bats, problem dimensions, and algorithmic parameters:

```
# Parameters
dim = num_tasks  # Dimensionality of the tasks
epochs = 100 # Number of generations
pop_size = 100  # Population size
gam = 0.1  # Pulse rate increasing rate
qmin = 0  # Minimum frequency
qmax = 1  # Maximum frequency
```

3

Metaheuristic function : Bat Algorithm

Search and Solution Update

```python
# Function that returns a solution (2D matrix containing the X_ij) respecting the constraints
def bat_task_assignment(position, workload, capacities):
    # Initialization
    num_tasks = len(workload)
    num_vms = len(capacities)
    solution = np.zeros((num_vms, num_tasks), dtype=int)
    assigned_workloads = np.zeros(num_vms)

    # Sort the tasks in ascending order based on the position of the bat
    sorted_task_indices = np.argsort(position)
    for task_index in sorted_task_indices:
        task_workload = workload[task_index]
        vm_candidates = np.where(np.array(capacities) >= task_workload)[0]
        for vm in vm_candidates:
            # Capacity constraint
            if assigned_workloads[vm] + task_workload <= capacities[vm]:
                solution[vm, task_index] = 1
                assigned_workloads[vm] += task_workload
                break  # Assign task to the first available VM to make sure only one VM has this task
    return solution
```

```python
# Archive part
for bat in population:
    # Get and evaluate a solution depending on position of the bat
    solution = bat_task_assignment(bat.pos, workload, capacity)
    bat_fitness = [makespan(solution), total_cost(solution)]
    bat.set_fitness(bat_fitness)
    bat.set_sol(solution)
    if bat not in archive:
        is_dominated = False
        for archived_bat in archive:
            if bat_fitness[0] < archived_bat.fitness[0] and bat_fitness[1] < archived_bat.fitness[1]:
                # If the new bat dominates an archived bat, remove the archived bat
                archive.remove(archived_bat)
            elif bat_fitness[0] >= archived_bat.fitness[0] and bat_fitness[1] >= archived_bat.fitness[1]:
                # If an archived bat dominates the new bat, it is dominated
                is_dominated = True

        if not is_dominated:
            # If the new bat is not dominated, add it to the archive
            archive.append(bat)
            # Selecting the last best bat as a reference
            best_bat = bat
    else:
      # Checking the dominance in the archive
      for archived_bat in archive:
        if bat.fitness[0]>=archived_bat.fitness[0] and bat.fitness[1]>=archived_bat.fitness[1] and bat.fitness!=archived_bat.fitness:
          archive.remove(bat)
    makespan_cost_epoch.append(bat_fitness)

makespan_costs.append(makespan_cost_epoch)
```

3

Metaheuristic function : Bat Algorithm

Search and Solution Update

- At each iteration, each bat conducts a local search around its current position
- The local search is guided by the quality of the best solution, and each bat updates its position accordingly
- We maintain an archive of the best solutions found, i.e. the Pareto optimal solutions

```
# Moving part
for bat in population:
  if bat not in archive:
    new_pos = [0.00 for _ in range(dim)]
    new_vel = [0.00 for _ in range(dim)]

    # Frequence of the bat
    freq = random.uniform(qmin, qmax)

    # Random pulsation
    pulse_chance = random.uniform(0, 1)
    for d in range(dim):
        # Updating velocity
        new_vel[d] = bat.vel[d] + (bat.pos[d] - best_bat.pos[d]) * freq

        # If pulsation rate not loud enough
        if pulse_chance > bat.pulse_rate:
            # Stays close to the best bat
            new_pos[d] = best_bat.pos[d] + (random.uniform(-1, 1) * 0.1)
        else:
            # Moving toward the best bat and keeps exploring
            new_pos[d] = bat.pos[d] + new_vel[d]

    # Update
    bat.vel = new_vel
    bat.pos = new_pos
    bat.pulse_rate = bat.max_pulse_rate * (1 - exp(-gam * e))
```

Metaheuristic function : Bat Algorithm

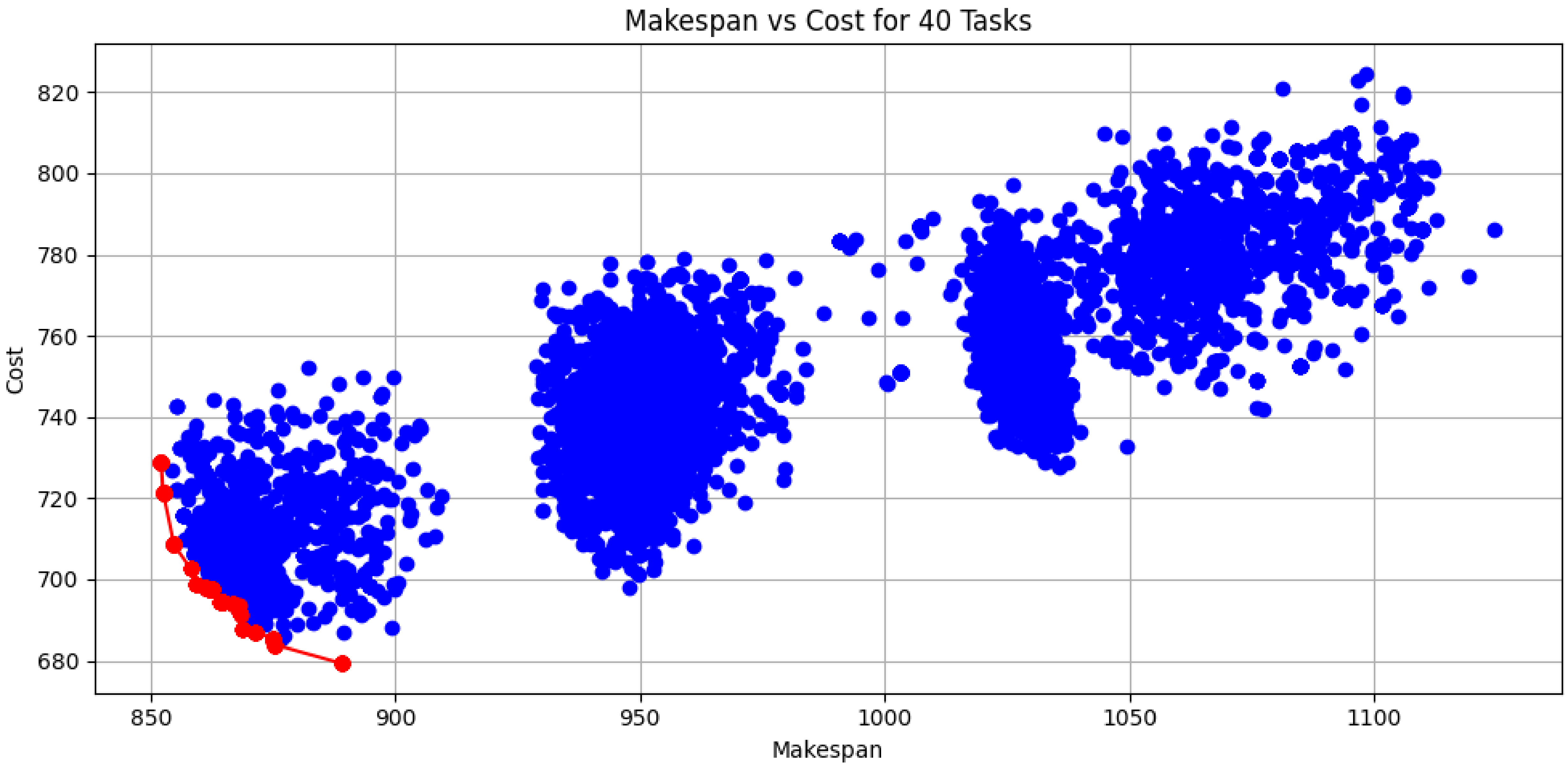
Termination Criterion:

- The algorithm continues iterating until a predefined termination criterion is met, here it is the maximum number of iterations (epochs)

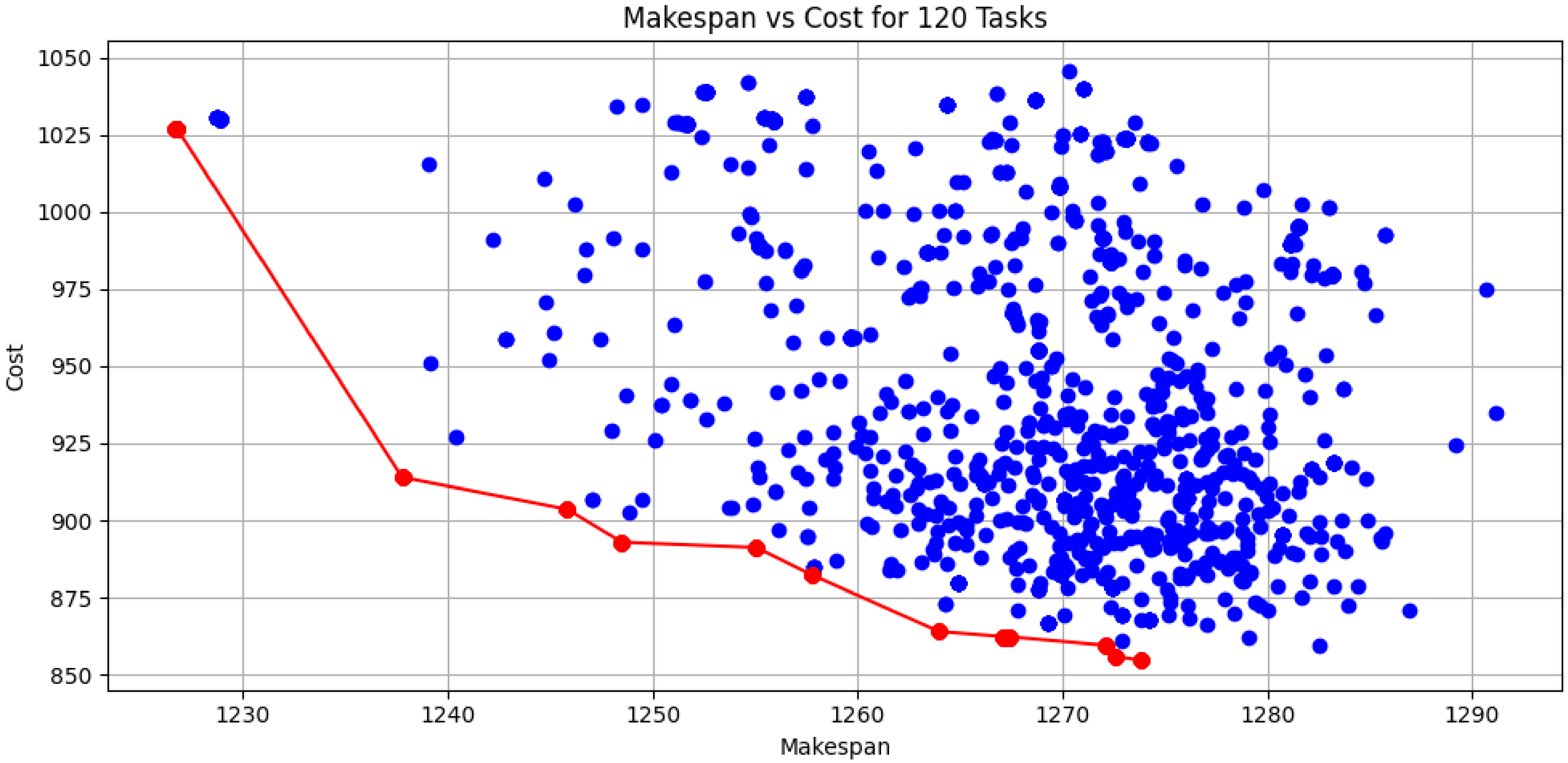
4 Performances and comparison

- O(enm): e the number of epochs, n the number of tasks and m the number of bats.
- e and m fixed leads to an execution time linear in n.
- O(ln(n)) possible?
- 30 seconds: solve a task scheduling problem with 40 tasks in 100 epochs with a population of 100 bats.

4 Performances and comparison



4 Performances and comparison



4 Performances and comparison

- Comparison with the solution of the group of Justine and David
- Task scheduling problem with a genetic algorithm but with different constraints
- Memory capacity constraint rather than a CPU capacity and a different calculation for the makespan
- Not completely comparable but same execution time
- Both performant and solutions graphically sufficient

5 Conclusion

- A good adaptation of the BAT algorithm to make it multi-objective in our problem
- Convincing results

THANK YOU!