Optimizing Post-Disaster Resource Allocation Using Metaheuristic Techniques:

A Multi-Objective Approach

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Federica Vinciguerra

24 October 2024

Problem Description

Problem Description

- Allocate limited resources (water, food, medical aid) from centers to disaster-affected areas.
- Goals:
  - Minimize transportation distance.
  - Maximize demand satisfaction.
- Areas have different priority levels:
  - High-priority (level 3)
  - Medium-priority (level 2)
  - Low-priority (level 1)
- Constraints:
  - Each center has a limited amount of each resource type.
  - Total resources allocated to an area cannot exceed its demand.

Example: The following setup represents allocations from three different centers to four areas, including random priority levels and a distance matrix.

- **Number of Centers:** 3
- **Number of Areas:** 4
- **Priority Levels:** Randomly assigned
  - Example: [3, 1, 2, 2]
- **Distance Matrix:**

$$\mathbf{D} = \begin{bmatrix} 10 & 15 & 20 & 25 \\ 12 & 18 & 30 & 22 \\ 14 & 10 & 25 & 20 \end{bmatrix}$$

- **Area Demands:**

$$\mathbf{A} = \begin{bmatrix} 25 & 30 & 20 \\ 15 & 10 & 35 \\ 20 & 25 & 30 \\ 10 & 15 & 25 \end{bmatrix}$$

- **Resource Limits:**

$$\mathbf{R} = \begin{bmatrix} 50 & 40 & 60 \\ 55 & 45 & 65 \\ 60 & 50 & 70 \end{bmatrix}$$

Solution Example: A possible chromosome representing resource allocations from centers to areas can be structured as follows:

$$\mathbf{X} = \begin{bmatrix} C_1 & C_2 & C_3 \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 5 & 3 & 2 \\ 0 & 1 & 4 \\ 2 & 0 & 3 \\ 4 & 5 & 1 \end{bmatrix} & \begin{bmatrix} 6 & 2 & 3 \\ 1 & 0 & 2 \\ 0 & 5 & 3 \\ 7 & 4 & 1 \end{bmatrix} & \begin{bmatrix} 2 & 1 & 5 \\ 4 & 0 & 3 \\ 3 & 2 & 1 \\ 0 & 3 & 2 \end{bmatrix} \end{bmatrix} \quad (1)$$

Distance calculation:

$$D = \sum_{c=1}^{n_c} \sum_{a=1}^{n_a} \left(\frac{1}{\text{priority}_a} \cdot \text{dist}_{c,a} \cdot 1(\text{alloc}_{c,a} > 0) \right) \tag{2}$$

Demand Calculation Function:

$$\text{Total demand met} = \sum_{\text{c}=1}^{n_c} \sum_{\text{a}=1}^{n_a} \sum_{\text{r}=1}^{n_r} \text{allocation}_{\text{c,a,r}} \tag{3}$$

where $\text{allocation}_{\text{c,a,r}}$ represents the amount of resource allocated from center c to area a for resource type r.

Variable Definitions:

- c: index for centers (from 1 to n_c)
- a: index for areas (from 1 to n_a)
- r: index for resource types (from 1 to n_r)

Work Environment

Programming Language: Python

- Chosen for its extensive libraries and community support.
- Facilitates rapid development and prototyping.

Libraries used:

- NumPy for numerical computations.
- Matplotlib for result visualization.
- Paretoset to deal with the multiobjective nature of the problem.

Genetic Algorithm

Genetic Algorithm Workflow (1/2)

- **Initialization:**
  - Randomly generate population of allocation plans (chromosomes).
  - High-priority areas (priority = 3) receive resources from multiple centers to maximize coverage.
  - Low-priority areas receive allocations from a single randomly selected center.
  - Allocation respects both resource limits and area demands.
- **Fitness Evaluation:**
  - Objective 1: Minimize total travel distance (distance fitness (2)).
  - Objective 2: Maximize resource allocation to satisfy area demands (demand fitness (3)).

Genetic Algorithm Workflow (2/2)

- **Selection:** Use tournament selection to choose parents for the next generation.
- **Crossover:** Apply either uniform or two-point crossover to generate offspring, with a 50% probability of choosing either method.
- **Mutation:** Random reallocation of resources within constraints, governed by a fixed mutation rate.
- **Repair Function:** Ensure all chromosomes remain feasible, fixing any violations in resource limits or area demands.
- **Elitism:** Retain top-performing chromosomes based on combined fitness scores to maintain solution quality.

Repair Chromosome Example: 1 Center & 3 Areas

1. **Initial Chromosome**:

$$\text{Chromosome} = \begin{bmatrix} 5 & 3 & 4 \\ 6 & 7 & 5 \\ 4 & 2 & 6 \end{bmatrix}$$

2. **Resource Limits**:

$$\text{Resource Limits} = \begin{bmatrix} 10 \\ 15 \\ 12 \end{bmatrix}$$

3. **Area Demands**:

$$\text{Area Demands} = \begin{bmatrix} 3 & 2 & 5 \\ 2 & 4 & 7 \\ 8 & 6 & 10 \end{bmatrix}$$

4. **Repaired Chromosome**:

$$\text{Repaired Chromosome} = \begin{bmatrix} 0 & 2 & 1 \\ 6 & 7 & 5 \\ 4 & 2 & 6 \end{bmatrix}$$

Pareto Front Solutions

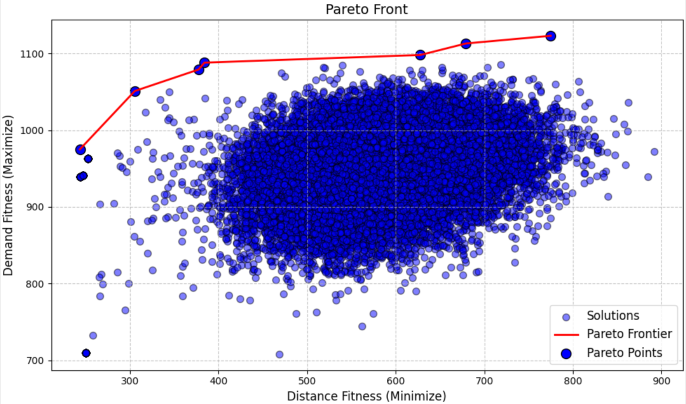


Figure: Pareto front solutions obtained with the following parameters:

$$\begin{aligned}\text{Elitism} &= 0.05 \times \text{Population Size},\\ \text{Mutation Rate} &= 0.3,\\ \text{Crossover Type} &= \text{Mixed between uniform and two points.}\end{aligned}$$

Simulated Annealing

Key Parameters of Simulated Annealing

Parameter	Value	Description
Initial Temperature T_0	1000	Controls the acceptance of worse solutions.
Cooling Rate Ω	0.995	Defines how fast the temperature decreases.
Min Temperature	1×10^{-5}	Stops the process when this temperature is reached.
Max Iterations	10000	The total number of iterations allowed.
Neighbors per Iteration	50	Number of neighboring solutions explored per iteration.
No Improvement Count	1000	Stops if no improvement occurs after these many iterations.

Simulated Annealing Workflow

1. Initialize the current and best solution with the initial solution provided by the previous GA.
2. For each iteration:
   - Generate neighbors by exploring small changes.
   - Calculate fitness (distance (2) and demand (3)).
   - Accept the neighbor if it’s better or probabilistically if it’s worse.
   - Decrease the temperature according to the cooling schedule.
3. Stop when the temperature reaches the minimum threshold or after a fixed number of iterations.

Neighborhood Generation Function

- **Method**:
  - Randomly select a center, area, and resource type.
  - Increment by 1 the resource allocation if it is strictly lower than area demands and resource limits.
  - Decrement by 1 otherwise
- This method ensures that the solution stays within feasible bounds (no repair function is needed) and allows to explore the neighborhood of the initial solution with very small changes.

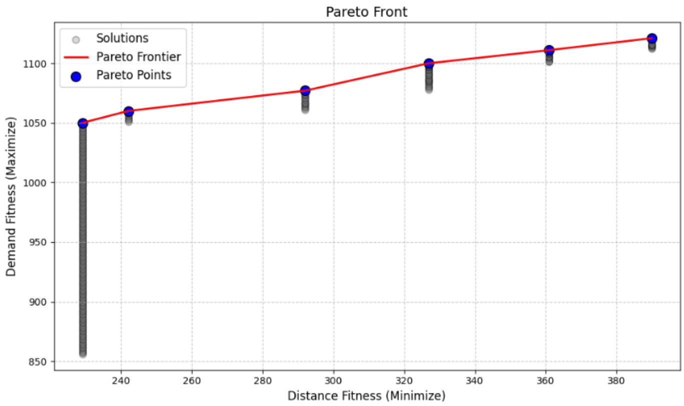
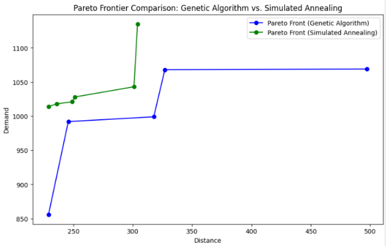


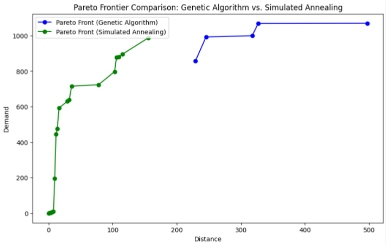
Figure: Pareto fronter of the Simulated Annealing Process

Conclusion

Pareto Frontier Comparisons



(a) Genetic Algorithm vs. Simulated Annealing with Well-Initialized Starting Point



(b) Genetic Algorithm vs. Simulated Annealing with Random Initialization

Comparison Table

Simulated Annealing (SA)	Genetic Algorithm (GA)
Lower computational complexity.	Higher computational cost due to population-based operations.
Performance highly dependent on the initial solution, affecting exploration.	Independent of the initial solution, allowing for broader exploration of the solution space.