Local Search: Warehouse Layout Optimization

New Attempt

- Due Oct 10 by 11:59pm
- Points 100
- Submitting a file upload

Problem Context

Imagine a warehouse where **robots** are responsible for collecting packages from shelves and delivering them to pick-up stations.

The efficiency of this system depends on how we arrange the shelves and stations inside the warehouse. If shelves are poorly placed, robots may have to travel long distances, block each other's paths, or create traffic jams. On the other hand, a well-designed layout reduces travel distance, waiting time, and congestion.

Your task in this assignment is to design and optimize a warehouse layout using **local search methods** (Hill Climbing, Simulated Annealing) and a **Genetic Algorithm** (GA). You will also create **visualizations** that help explain and demonstrate your solution.

Problem Description

- The warehouse is represented as a **grid** (for example, 8×8 cells see figure 1).
- Some cells contain **shelves** (gray squares). These are where items are stored.
- A few cells contain **stations** (colored circles). These are where robots deliver items.
- Robots receive **orders**: each order is a list of items to collect from shelves and deliver to a station.
- The robots move step by step across the grid, taking the shortest path (avoiding shelves).

Your Goal

Design a layout of shelves and stations that **minimizes the overall cost of fulfilling orders**. The cost combines:

1. **Total travel distance** (robots should not walk too far).
2. **Congestion penalty** (too many robots using the same path at the same time is bad).
3. **Fairness across stations** (stations should not be overloaded while others stay idle).

You must **implement and compare three optimization methods**:

- ***Hill Climbing (HC)***
- ***Simulated Annealing (SA)***
- ***Genetic Algorithm (GA)***

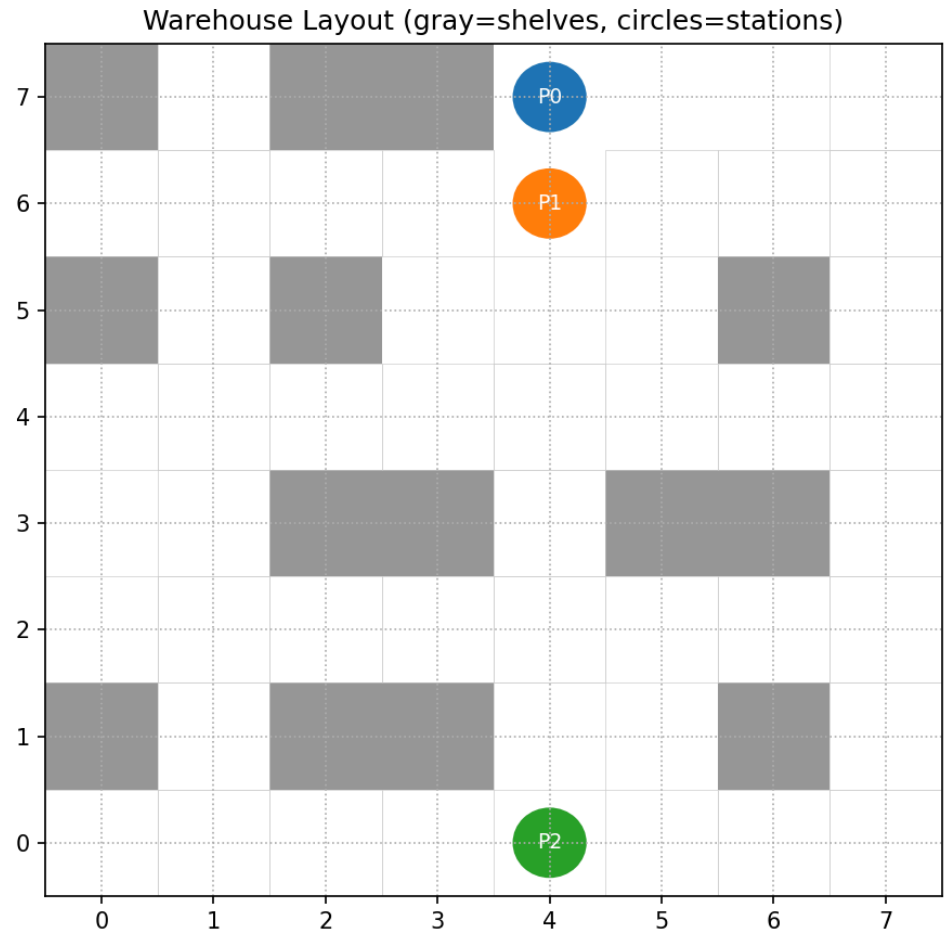

Figure 1: Sample warehouse layout

Environment Details

The simulation environment is already implemented for you in `warehouse_starter/`. You should **read through the code** to understand how it works, since your algorithms will depend on these mechanics. You can download the starter code **warehouse_starter.zip. (https://canvas.umn.edu/courses/518032/files/55624880?wrap=1)** **(https://canvas.umn.edu/courses/518032/files/55624880/download?download_frd=1)**

Key points:

- **Grid Layout**
  - The warehouse is a grid (default 8×8).
  - Corridors (arteries) are pre-defined using a simple rule (every odd row + every 3rd column).
  - Stations (`P#`) must be placed on corridor cells.
  - Non-corridor cells become shelves (`S#`) except for a small number left empty.
- **Orders (Zipf Distribution)**
  - Orders are lists of items to pick from shelves.
  - Items are sampled with a **Zipf-like distribution** (few items are very popular, many are rare).
  - Parameters:

    - `orders` (number of orders to simulate),
    - `items_per_order` (number of items in each order),
    - `zipf_alpha` (controls skew, higher means more “head-heavy”).
  - Example: with α=1.2, item `S0` might be requested far more often than `S40`.
- **Robot Tours**
  - Each order is assigned to a station.
  - The robot path is computed as:
    `station → nearest item → next item → … → return to station` (greedy nearest-neighbor).
  - Paths are shortest routes (via BFS), respecting shelves and optional one-way aisle constraints.

What You Will Do

Objective Function

- The **objective function** defines how “good” a layout is.
- Open `warehouse_starter/sim.py` and find the **STUDENT TODO** block. That’s where you will design your scoring function.
- You are free to include distance, congestion, fairness, and any other factors you think are important.
- In your **report**, explain your design choices, including equations or pseudocode.

Optimization Algorithms

Implement the three algorithm mentioned above following inside the `algos/` directory:

- hill.py → hill_climb(...)
- sa.py → simulated_annealing(...)
- ga.py → genetic_algorithm(...)

Each algorithm should:

- Take the current warehouse state.
- Explore new layouts (neighbors, mutations, etc.).
- Return the best layout it finds.

You may add helper functions, modify neighbor definitions, or experiment with state transitions.

Running Experiments

You can run different algorithms directly from the command line. For example:

Baseline random layout

```
python -m warehouse_starter.main --algo none
```

Hill Climbing

```
python -m warehouse_starter.main --algo hc
```

Simulated Annealing

```
python -m warehouse_starter.main --algo sa
```

Genetic Algorithm

```
python -m warehouse_starter.main --algo ga
```

Run all implemented algorithms

```
python -m warehouse_starter.main --algo all
```

Check the provided *README.md* for more details.

Visualization

You do **not** need to write plotting or animation code. The starter includes:

- viz.py to generate:
  - `layout.png` (final shelves + stations),
  - `heat.png` (traffic density),
  - `station_bars.png` (per-station workload).
- `animate.py` to generate an **optional** `traffic.gif` that shows robots moving, with item boxes changing color when picked.

Your job is to **use** these to support your report (e.g., include screenshots/figures and explain what they show). The animation is optional; it's there to help you sanity-check behavior and visually explain congestion.

How to preview the animation (optional)

After you've implemented your objective and an algorithm, you can produce a small GIF:

Any algo works; 'none' just uses a random feasible layout

```
python main.py --algo hc \
  --animate --anim-orders 25 --sec-per-step 0.5 \
  --out-gif out/traffic.gif
```

Notes:

- `--anim-orders` controls how many orders are animated (keep this modest).
- `--sec-per-step` controls playback speed.

Include the GIF in your submission **only if you choose to generate it**.

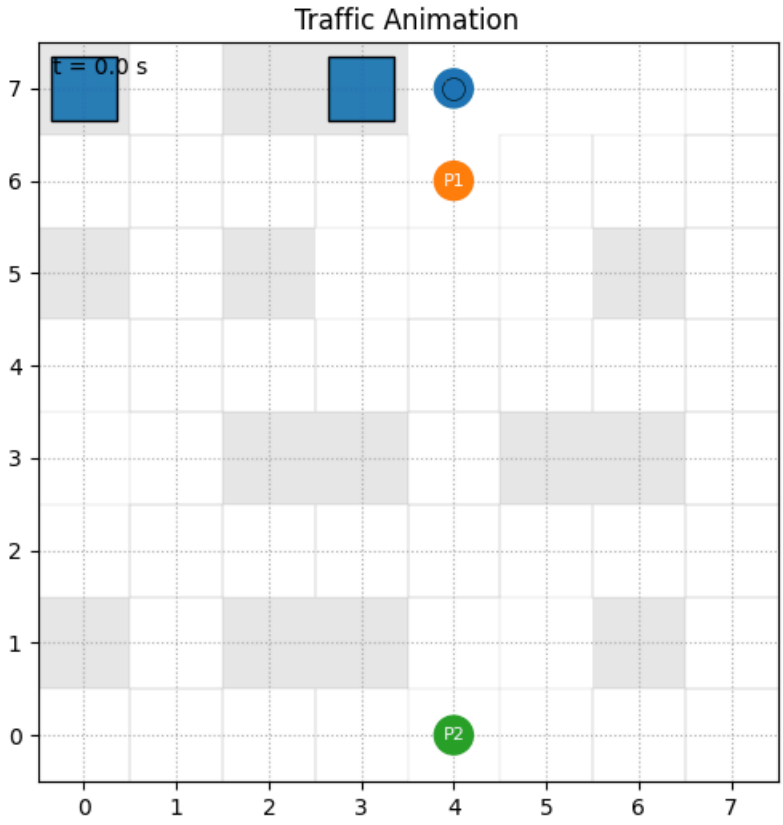

Figure 2: Generated GIF

Testing Notes

- We will run your code with **different random seeds** and **different parameters** than yours.
- Your code must run **without modification** from the warehouse_starter/ directory:
  - If you change the structure or move files, **update the README** with exact run instructions.
- Do **not** hardcode assumptions (e.g., fixed grid size, a specific number of orders, fixed item IDs). Your code should respect the values in config.py and command-line flags.

Write a Report

Submit a short **1–2 page PDF report**. It must include:

- Your **objective function** (equations).
- Description of each algorithm and any modifications you made.
- **Results:** comparison of HC, SA, GA vs. baseline (include screenshots of plots/heatmaps).
- **Discussion of trade-offs:** for example, does one method reduce distance but increase congestion?

Keep it concise but clear.

Deliverables

Submit a single .zip file named:

```
warehouse_layout_<yourlastname>.zip
```

It must contain:

1. Your **code**, including all algorithm implementations in algos/.
2. **Outputs:**
   - summary.json
   - layout.png
   - heat.png
   - station_bars.png
   - (Optional) traffic.gif
3. Your **report (PDF)**.

HW2 - Rubric

Criteria	Ratings					Pts
Environment	**20 pts** **Full Marks**	**18 pts** **Minor mistakes not affecting the assignment too drastically**	**15 pts** **Some mistakes leading that have some impact but understanding is somewhat correct**	**10 pts** **Poor understanding affecting the rest of the assignment drastically**	**0 pts** **No Marks**	20 pts
Objective Function Distance, congestion, fairness, all taken into account	**20 to >18.0 pts** **Full Marks**	**18 to >14.0 pts** **Minor mistakes to one or more of the characteristics**	**14 to >10.0 pts** **Major mistakes to one or more of the characteristics**	**10 to >0 pts** **No Marks**		20 pts
Hill-climbing	**20 to >18.0 pts** **Full Marks**	**18 to >15.0 pts** **Minor mistakes**	**15 to >10.0 pts** **Major mistakes but somewhat reasonable approach**	**10 to >0 pts** **Major mistakes and not a good understanding of how the algorithm works**		20 pts
Simulated Annealing	**10 to >9.0 pts** **Full Marks**	**9 to >8.0 pts** **Minor mistakes**	**8 to >5.0 pts** **Major mistakes but somewhat reasonable approach**	**5 to >0 pts** **Major mistakes and not a good understanding of how the algorithm works**		10 pts
Genetic Algorithm	**10 pts** **Full Marks**	**6 pts** **Minor mistakes (running the assignment/output is not following entirely the instructions)**	**3 pts** **Some mistakes in output either in the expected results for the schedule or the replay summary**	**1 pts** **output is not reporting the expected results both for schedule and replay summary**	**0 pts** **No Marks**	10 pts
Written Summary 1-2 Pages explaining your results.	**10 pts** **Full Marks** 1. Results analysis: compares methods (HC/SA/GA) with evidence (tables/plots) and explains why outcomes differ. 2. Design philosophy: explains objective terms (distance, congestion, fairness), why chosen, and key algorithm choices (representation, neighbors, cooling/selection). 3. Runtime & performance: includes timing (wall-clock or evals/sec) and a clear time–	**7 pts** **Missing small detials** Exactly one required section is missing or shallow.	**5 pts** **Unclear or not fully explained**	**0 pts** **No Marks**		10 pts

Criteria	Ratings				Pts
	performance comparison (e.g., score vs time plot or table). Also reports other metrics beyond score (e.g., J1/J2/J3 breakdown, constraint violations).				
Visualization	**10 pts** **Full Marks** Excellent work requirements met	**7 pts** **Minor mistakes**	**5 pts** **Basic Visualization** Minimum requirements met	**0 pts** **No Marks**	10 pts
Other Creative Approaches Implementation and discussion of other some improvements such as one-way aisles, different robot speeds, etc, as described in writeup	**5 pts** **Full Marks**		**0 pts** **No Marks**		5 pts
					Total Points: 105