Tecnológico de Monterrey

Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Monterrey

Análisis y diseño de algoritmos avanzados
Gpo 604

Optimizing Warehouse Layouts using Advanced Search Techniques

Profesores:

Nezih Nieto Gutiérrez

Equipo

Angel Alexandro Angulo Prudencio	**A00840313**
Sergio Gómez Guerrero	**A01571538**
Hector Aranda Garcia	**A01178284**
Rutilo Alberto	**A01384647**

22 de octubre 2025

Compare deterministic (A*) vs stochastic (SA) results.

En este proyecto, la optimización se abordó en dos niveles distintos:

- Optimización del Layout (el Mapa): Encontrar la mejor configuración de pasillos y zonas de almacenamiento.
- Optimización de Rutas (el Camino): Encontrar el camino más corto para un robot dentro de un layout dado.

Para resolver esto, se emplearon dos paradigmas de algoritmos: A* para las rutas y Simulated Annealing (Estocástico) para el layout.

El algoritmo A*, implementado en pathfinding.py, es un algoritmo determinístico. Esto significa que, dado el mismo mapa, mismo punto de inicio y mismo punto final, garantiza encontrar la ruta óptima y lo hará de la misma manera cada vez.

- Rol: Encontrar la ruta más eficiente para una sola tarea de robot.
- Fortaleza: Es rápido, eficiente y óptimo para su tarea. Utiliza una heurística admisible (Distancia de Manhattan), lo que asegura matemáticamente que la primera ruta que encuentra al objetivo es la mejor posible.
- Limitación: A* es ineficaz para el problema del layout. El número de posibles configuraciones de almacén (intercambiar celdas de pasillo y almacenamiento) es astronómicamente grande. Intentar probar cada layout con A* (un enfoque de "fuerza bruta") sería computacionalmente imposible.

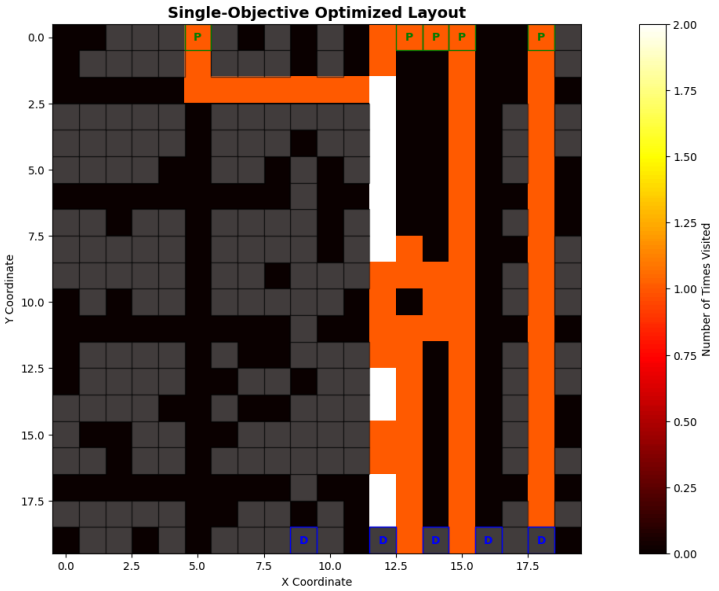


Figura 1. *Single-Objective Optimized Layout.*

Aquí es donde entra el Simulated Annealing (SA), un algoritmo estocástico (aleatorio). En lugar de probar todas las opciones, explora inteligentemente el espacio de soluciones.

- **Rol**: Optimizar el diseño general del almacén (el layout).
- **Fortaleza**: Es excelente para escapar de "óptimos locales" (soluciones que parecen buenas pero no lo son). Lo logra aceptando probabilísticamente soluciones peores (el math.exp(-cost_diff / temp)), especialmente a "temperaturas" altas, permitiéndole explorar más.
- **Resultado**: **La Figura 1** es el resultado de este proceso. Muestra un layout donde el SA determinó una configuración que minimiza el costo total (distancia, congestión y energía). Las rutas (naranja/blanco) son claras y conectan eficientemente los muelles (D) y las estaciones (P).

When does each paradigm find “good enough” solutions faster?

Esta pregunta se aplica casi exclusivamente al paradigma estocástico, ya que el paradigma determinista **(A*)** no está diseñado para encontrar soluciones "suficientemente buenas", sino soluciones óptimas a problemas específicos y acotados.

El **SA** es mucho más rápido para encontrar soluciones "suficientemente buenas" en problemas de optimización con espacios de búsqueda masivos, como lo es el diseño del

layout del almacén. La evidencia clave es la gráfica de convergencia generada por el benchmark.py.

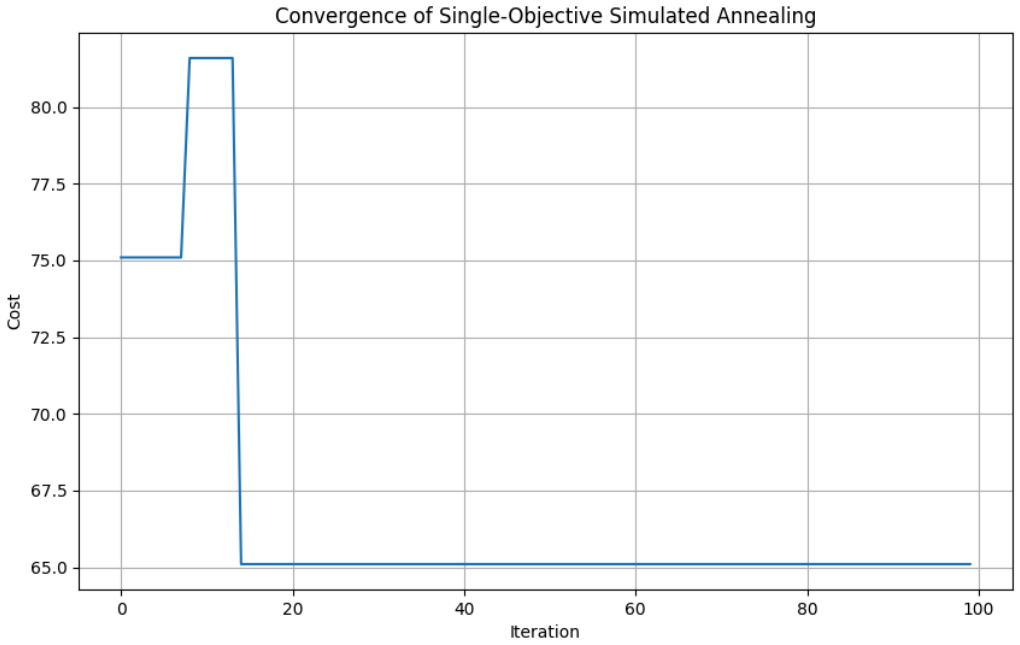


***Figura 2**. Convergence Of Single-Objetive Simulated Annealing.*

Esta gráfica muestra cómo el costo total (una métrica de qué tan "bueno" es el layout) cae drásticamente en las primeras 15-20 iteraciones y luego entra en una larga "meseta" donde solo realiza mejoras mínimas. Esto demuestra que el **SA** encuentra una solución "buena" en una fracción del tiempo total de ejecución.

How does temperature scheduling or heuristic admissibility affect performance?

Ambos son parámetros críticos que controlan el equilibrio entre la calidad de la solución y la velocidad de ejecución.

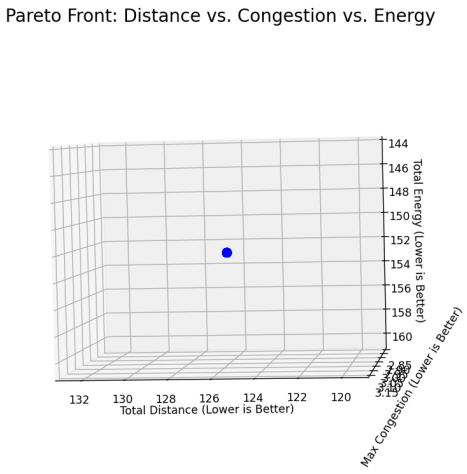
La programación de temperatura en **SA** gestiona el comportamiento de exploración del algoritmo. Una temperatura inicial (temp) alta permite al algoritmo aceptar soluciones peores con más frecuencia (basado en math.exp(-cost_diff / temp)), lo cual es vital para escapar de **óptimos locales** al inicio de la búsqueda. La tasa de enfriamiento (cool_rate) controla la

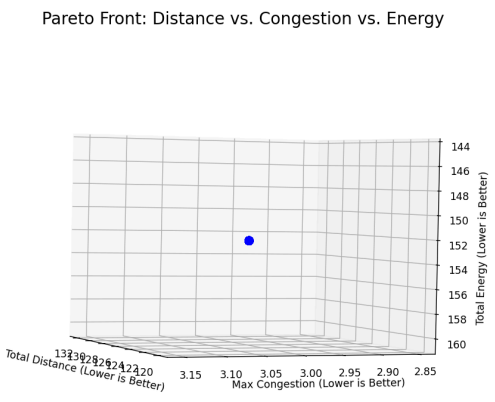
velocidad: un cool_rate cercano a 1 (ej. 0.99) enfría el sistema lentamente, permitiendo una exploración más profunda a costa de más tiempo, mientras que un cool_rate bajo (ej. 0.9) converge rápido, pero arriesgándose a quedar en una solución subóptima.

Por otro lado, la admisibilidad de la heurística es clave para **A***. Nuestra heurística, la Distancia de Manhattan, es admisible porque nunca sobreestima el costo real para llegar al objetivo. Esta admisibilidad es lo que garantiza la optimalidad de **A***. Si usáramos una heurística no admisible, A* podría correr más rápido al explorar menos nodos, pero perderíamos la garantía de encontrar la ruta más corta.

Optimización Multi-Objetivo (MOSA)

Uno de los objetivos clave del proyecto era generar un Frente de Pareto para el escenario multi-objetivo (Distance, Congestion, Energy) utilizando un algoritmo MOSA. Sin embargo, tras la ejecución del benchmark.py, se determinó que no era posible generar una gráfica de Pareto válida.





El análisis de los logs de ejecución reveló que la gran mayoría de los layouts generados aleatoriamente por el optimizador (get_neighbor_layout) resultaban en escenarios de "gridlock" (bloqueo mutuo) o "livelock" (bloqueo por bucles de replanificación) para los robots.

Causa del Bloqueo: Los robots, al intentar seguir rutas óptimas generadas por A*, se encontraban en pasillos estrechos, bloqueándose mutuamente. El RobotController intentaba resolver esto forzando una replanificación (Re-planning path.).

Livelock: Esto creaba un bucle irresoluble donde los robots se bloqueaban, replanificaban, intentaban moverse, y se volvían a bloquear en el siguiente paso, repitiendo el ciclo.

Timeout de Simulación: Inevitablemente, estos bloqueos provocaban que la simulación alcanzara el límite máximo de pasos (max_steps=500) antes de que los robots completaran sus tareas (Simulation timed out...).

Penalización de Costo Infinito: Nuestra función evaluate_layout estaba diseñada correctamente para manejar esto: detectaba que los robots no habían llegado a su destino (all_robots_at_target era False) y asignaba un costo inf para penalizar severamente ese layout.

Resultado Final: El pareto_archive del optimizador se llenaba con miles de estas soluciones inf (Archive Size: 1490). Al ser filtradas antes de graficar, el resultado era una lista vacía, impidiendo la visualización del frente.

Resumen

Mientras que nuestro **A*** y nuestro **SA** para un solo objetivo funcionaron, el problema multi-objetivo falló no por la optimización, sino por el desafío de la simulación y la resolución de conflictos en tiempo real. Una futura implementación requeriría un *RobotController* mucho más avanzado, capaz de gestionar derechos de paso o negociar rutas para evitar el gridlock.