Proyectos de Programación

Tercera Entrega

Oriol Farrés, Daniela Cervantes, Guiu Carol y Daniel Martínez

Grado GEI-PROP - Curso 2024-25, Quadrimestre de otoño

Identificador equipo: Grupo 42.3

23 de Diciembre

oriol.farres daniela.cervantes guiu.carol daniel.martinez

Tercera Entrega

Índice general

Capítulo 1

Estudio Estadístico [Simulated Annealing]

1.1. Introducción

El *Quadratic Assignment Problem* (QAP) constituye uno de los problemas de optimización combinatoria más estudiados y, al mismo tiempo, más desafiantes, dado que se clasifica como $\mathcal{NP}$-Hard. Tal como se describe en la Sección anterior, el QAP busca asignar n instalaciones a n ubicaciones de modo que se minimice (o maximice, según el contexto) una función objetivo que combina los denominados *flujos* entre instalaciones con las *distancias* entre ubicaciones. Debido a su complejidad factorial, incluso para instancias relativamente pequeñas, este problema exige el uso de métodos heurísticos y metaheurísticos para poder obtener soluciones de calidad en tiempos de cómputo razonables.

En este documento, nuestro objetivo es aplicar la metaheurística de *Simulated Annealing* (SA) para resolver una instancia del QAP adaptada a la organización de productos en una estantería (véase la formulación en la Sección anterior). Concretamente, deseamos *maximizar* la similitud entre productos adyacentes, lo cual se traduce en una transformación del problema a la forma tradicionalmente usada en el QAP (que tiende a la minimización). El uso de SA se justifica dado que:

- **Escalabilidad:** A diferencia de los métodos exactos (e.g., fuerza bruta), SA permite explorar espacios de búsqueda de mayor tamaño en menor tiempo.
- **Flexibilidad:** SA puede acoplarse a diversas restricciones o adaptaciones particulares del QAP, como es el caso de adyacencias especiales (p.e., *wrap-around*).
- **Calidad aproximada de las soluciones:** Si se sintonizan adecuadamente los parámetros, SA ofrece soluciones *aproximadamente óptimas* con gran probabilidad de éxito.

Parámetros Críticos del SA. Un aspecto esencial para obtener resultados satisfactorios en SA consiste en la elección y ajuste de los siguientes parámetros:

- *Temperatura inicial* (T): control de la probabilidad de aceptar soluciones de peor calidad al comienzo.
- *Factor de enfriamiento* (α): indica la velocidad a la cual decrece T en cada iteración.
- *Temperatura mínima* ($T_{\text{mín}}$): cuando la temperatura es demasiado baja, apenas se aceptan soluciones peores, y puede darse por terminado el proceso.
- *Número máximo de iteraciones* o *criterio de parada*: para limitar el tiempo de ejecución.

Por lo tanto, el estudio no se reduce a *implementar* SA, sino a encontrar un *régimen de temperatura* (T, α, $T_{\text{mín}}$, etc.) que equilibre dos objetivos:

1. *Exploración suficiente:* aceptar soluciones menos prometedoras al principio para escapar de mínimos locales.
2. *Explotación eficiente:* enfriar adecuadamente para converger a soluciones de alta calidad (o bajo coste, según la perspectiva).

A continuación, se presenta de forma esquemática el *pseudocódigo* de SA aplicado al QAP, haciendo hincapié en la parte de aceptación de soluciones y el proceso de enfriamiento:

SIMULATED ANNEALING(QAP):

1. Genera una solución inicial aleatoria (permutación de asignaciones).
2. Calcula el coste total de la solución actual.
3. Mientras no se alcance el criterio de parada:
   a) Genera una solución vecina mediante el intercambio de dos asignaciones aleatorias.
   b) Calcula el coste total de la solución vecina.
   c) Si el coste de la solución vecina es menor que el actual, acepta la solución vecina como la nueva solución actual.
   d) Si el coste es mayor, acepta la solución vecina con una probabilidad $\exp\left(-\frac{\Delta}{T}\right)$, donde Δ es el incremento en el coste y T es la temperatura actual.
   e) Actualiza la mejor solución encontrada si la solución actual es mejor que la mejor solución registrada.
   f) Enfría la temperatura multiplicándola por un factor de enfriamiento α.
4. Retorna la distribución con el coste mínimo encontrado como solución óptima.

En resumen, la motivación de este estudio radica en la **importancia de sintonizar adecuadamente los parámetros de recocido simulado** para adaptarse a la *naturaleza* y *dimensión* de la instancia del QAP que se está resolviendo. Unos parámetros demasiado conservadores pueden resultar en largos tiempos de cómputo con escasa mejora, mientras que parámetros muy agresivos a menudo

derivan en mínimos locales de calidad pobre. Por ello, la Sección de *Resultados y Discusión* presentará un análisis estadístico para determinar cuáles combinaciones de $(T, \alpha, T_{\text{mín}})$ propician soluciones de menor coste en el menor tiempo, ofreciendo de este modo una *guía práctica* para la resolución de instancias del QAP con *Simulated Annealing*.

1.2. Metodología

En esta sección se describe el diseño experimental empleado para estudiar el impacto de distintos parámetros de *Simulated Annealing* (SA) en la calidad de las soluciones obtenidas para varias instancias del QAP. Se explican los parámetros fijos y variables, la naturaleza de los casos de prueba utilizados y los criterios de evaluación.

1.2.1. Diseño del Experimento

El objetivo principal es analizar cómo afectan los valores de α (factor de enfriamiento), la temperatura inicial (Temp), y la temperatura mínima (Temp_{min}) al rendimiento de SA. Para ello, se llevaron a cabo múltiples ejecuciones con:

- **Parámetros fijos**:
  - maxIterations = 10000.
  - maxNoImprovement = 2500.

  Estos límites se eligieron con el fin de dar suficiente "espacio" de exploración a SA, pero manteniendo un tiempo de ejecución manejable.

- **Parámetros variables**:
  - Temperature $\in \{1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000\}$.
  - α (factor de enfriamiento). Se probaron distintos intervalos para α, con variaciones finas cuando se sospechó un valor "crítico" (p.ej., alrededor de 0,998, 0,999, etc.).
  - $\text{Temp}_{\text{min}} \in \{0{,}01, 0{,}009, 0{,}008, 0{,}007, 0{,}006, 0{,}005, 0{,}004, 0{,}003, 0{,}002, 0{,}001\}$.

- **Número de replicaciones (semillas)**: se definieron 10 semillas distintas para el generador de números aleatorios (seed), con el fin de ejecutar el experimento en un entorno reproducible y medir la variabilidad de los resultados.

En cada ejecución de SA, se mantiene fijo un *set* de valores para Temperature, α, Temp_{min}, y las variables maxIterations y maxNoImprovement se fijan en los valores mencionados. Se corre el algoritmo 10 veces (una por cada semilla), generando así medidas estadísticas (coste final, número de pasos, etc.) para cada combinación de parámetros.

1.2.2. Instancias de Prueba

Para examinar el comportamiento de SA en escenarios diversos, se utilizaron distintas matrices **F** (flujo/similitudes) de 25 productos (distribuidos en 5 baldas). Estas matrices difieren en el rango de valores:

- Matrices con valores en $[0, 100]$.
- Matrices con valores en $[0, 20]$.
- Matrices con valores en $[80, 100]$.

El propósito es comprobar si la escala de los flujos altera la sensibilidad de SA a los parámetros, así como verificar la consistencia de los resultados en distintas "intensidades" de flujos.

La matriz de distancias **D** se genera de forma coherente con la estructura de la estantería, contemplando adyacencias directas en filas y columnas. Se añade un enlace de *wrap-around* si procede, de forma que la primera y última posición se consideran adyacentes.

1.2.3. Algoritmo y Criterio de Evaluación

Cada ejecución de SA procede de la siguiente manera:

1. Se inicializa una permutación aleatoria de los productos usando la semilla correspondiente.
2. Se calcula el coste inicial de dicha asignación.
3. Se realizan iteraciones (hasta un tope de maxIterations o hasta que se supere maxNoImprovement pasos sin mejora):
   a) Se genera un vecino mediante el intercambio de dos productos aleatorios.
   b) Se calcula el nuevo coste.
   c) Se decide la aceptación o rechazo según la regla de SA (aceptación incondicional si mejora; en caso contrario, con probabilidad $\exp(-\Delta/T)$).
   d) Se enfría la temperatura multiplicándola por α y se compara con Temp_{min} como criterio de parada adicional.
4. Se registra el coste final, el número de pasos realizados y la mejor solución obtenida.

Para cada combinación de parámetros y cada instancia (**F**), se obtienen resultados agregados (mínimo, máximo, media, mediana) del *Coste* en las 10 replicaciones, permitiendo discernir qué configuración es más efectiva.

1.2.4. Resumen de Parámetros y Número de Ejecuciones

- Temperature: 10 niveles (1000 a 10000 con paso de 1000).

- α: Se ensayan múltiples intervalos. Por ejemplo: $\{0{,}99, 0{,}992, 0{,}994, 0{,}996, 0{,}998\}$ o $\{0{,}998, 0{,}99825, \ldots\}$ dependiendo del estudio puntual.
- Temp_{min}: 10 niveles (desde 0,01 a 0,001 en decrementos de 0,001).
- maxIterations = 10000, maxNoImprovement = 2500 (fijos).
- 10 semillas distintas por cada combinación (replicaciones).

El total de ejecuciones por caso se determina como:

$$\#(\text{Temperature}) \times \#(\alpha) \times \#(\text{Temp}_{\text{min}}) \times \#(\text{semillas}),$$

que puede superar fácilmente varios miles de corridas (dependiendo de cuántos valores α se evalúen). Se han dispuesto *nueve instancias de prueba* —tres para cada rango de valores de **F** $[0,100,\ 0,20,\ 80,100]$—, variando además estructuras y distribuciones de los flujos.

1.3. Resultados

En esta sección se presentan los resultados obtenidos del estudio estadístico llevado a cabo con distintas configuraciones de parámetros y diferentes rangos de coeficientes de similitud en las matrices de afinidad.

Se han considerado en total tres conjuntos de matrices:

- **3 Matrices de 25 productos con coeficientes de similitud en el intervalo [0,100].**
- **3 Matrices de 25 productos con coeficientes de similitud en el intervalo [0,20].**
- **3 Matrices de 25 productos con coeficientes de similitud en el intervalo [20,80].**

Cada uno de estos tres conjuntos consta de **3 matrices**, para un total de 9 casos distintos. En todos ellos se han realizado pruebas variando *4 configuraciones de parámetros* (combinaciones de `temperatures`, `alphas` y `minTemperatures`) con el objetivo de comparar la evolución de los costes medios obtenidos.

1.3.1. Configuraciones de parámetros

Antes de describir los resultados concretos de cada caso, resumimos las configuraciones estudiadas. Se han usado cuatro combinaciones diferentes de los siguientes parámetros:

- **Temperatures** = {1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000}
- **Alphas**:
  - Conjunto 1: {0.9, 0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99}
  - Conjunto 2: {0.99, 0.991, 0.992, 0.993, 0.994, 0.995, 0.996, 0.997, 0.998, 0.999}

- Conjunto 3: {0.998, 0.99825, 0.9985, 0.99875, 0.999, 0.99925, 0.9995, 0.99975}
- Conjunto 4: {0.99, 0.992, 0.994, 0.996, 0.998}

- **MinTemperatures** = {0.01, 0.009, 0.008, 0.007, 0.006, 0.005, 0.004, 0.003, 0.002, 0.001}

1.3.2. 3 Matrices de 25 Productos con coeficientes de similitud en un interval de [0,100]

Este grupo comprende tres casos distintos de matrices de 25×25 con valores en el rango $[0, 100]$. Cada *caso* se asocia a una matriz con una distribución diferente de similitudes:

Caso 1

En este apartado se analiza el rendimiento de *Simulated Annealing* (SA) sobre la primera matriz de 25×25 con valores de similitud en el rango $[0, 100]$. Para esta matriz, se llevaron a cabo múltiples ejecuciones de SA bajo distintas combinaciones de Temperature, α y Temp_{min}. A continuación, se incluyen cuatro tablas de resultados (*mean_cost*, *median_cost* y el número de ejecuciones *count*, siempre con 10 réplicas por configuración (con 10 *randoms* fijados):

Resultados para el Conjunto 1 de Alphas:

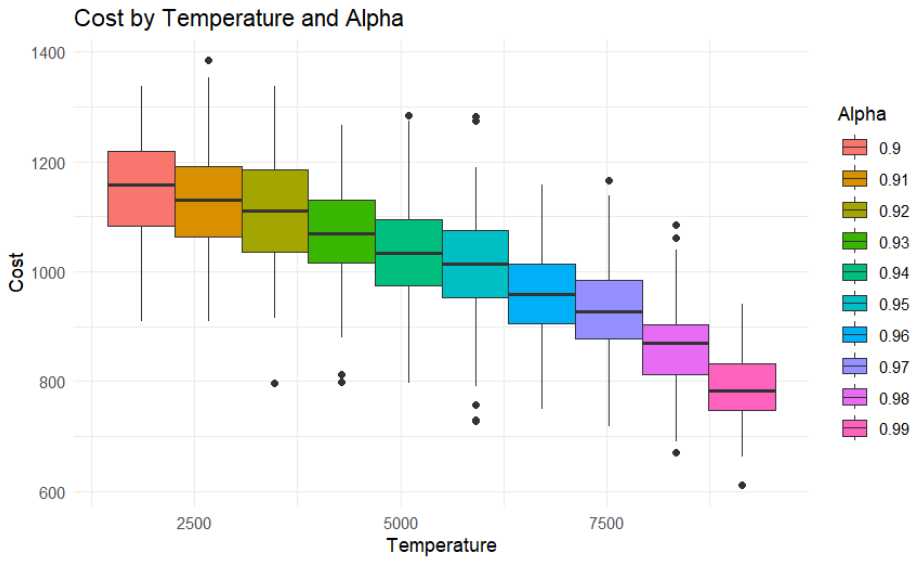

Figura 1.1: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.1: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
8000	0.99	0.001	750	758	10
8000	0.99	0.002	750	758	10
8000	0.99	0.003	752	758	10
8000	0.99	0.004	752	758	10
8000	0.99	0.005	756	758	10
8000	0.99	0.006	759	758	10
6000	0.99	0.001	760	742	10
8000	0.99	0.007	761	770	10
8000	0.99	0.008	761	770	10
6000	0.99	0.002	765	753	10

Cuadro 1.2: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
8000	0.99	0.001	611
8000	0.99	0.002	611
8000	0.99	0.003	611
8000	0.99	0.004	611
8000	0.99	0.005	611

Resultados para el Conjunto 2 de Alphas:

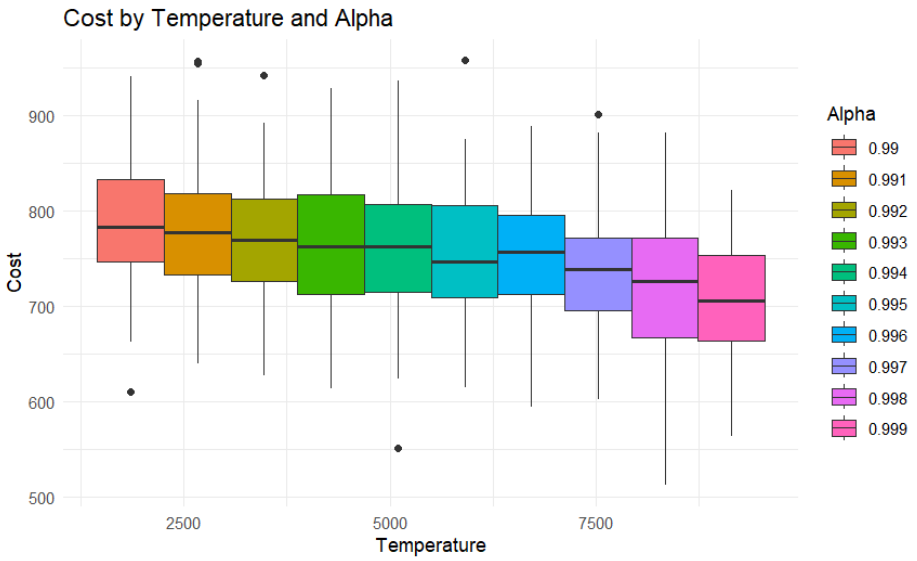

Figura 1.2: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.3: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
8000	0.999	0.001	684	668	10
8000	0.999	0.002	684	668	10
8000	0.999	0.003	684	668	10
8000	0.999	0.004	684	668	10
8000	0.999	0.005	684	668	10
8000	0.999	0.006	684	668	10
8000	0.999	0.007	684	668	10
8000	0.999	0.008	684	668	10
8000	0.999	0.009	684	668	10
8000	0.999	0.010	684	668	10

Cuadro 1.4: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
5000	0.998	0.001	513
5000	0.998	0.002	513
5000	0.998	0.003	513
5000	0.998	0.004	513
5000	0.998	0.005	513

Resultados para el Conjunto 3 de Alphas:

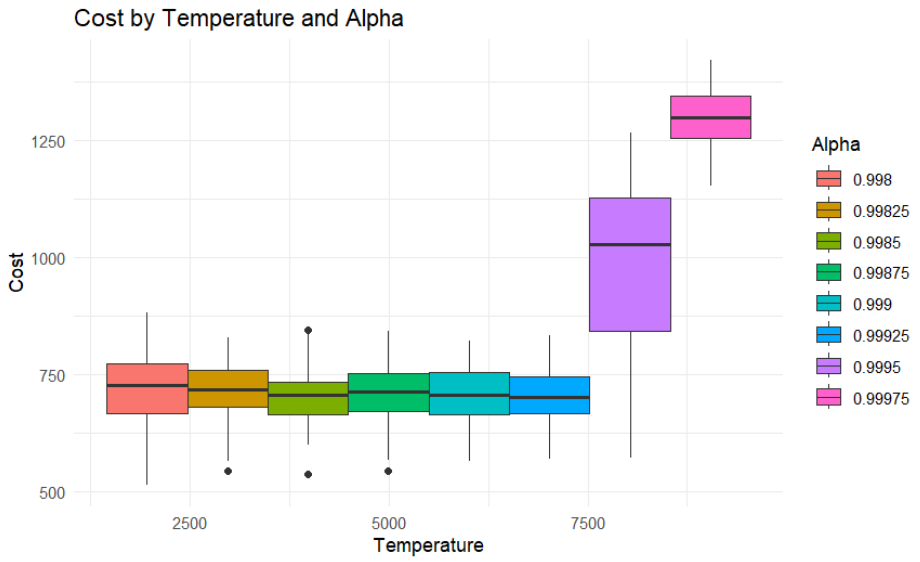

Figura 1.3: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.5: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
7000	0.99825	0.002	665	653	10
7000	0.99825	0.003	665	653	10
7000	0.99825	0.004	665	653	10
7000	0.99825	0.005	665	653	10
7000	0.99825	0.006	665	653	10
7000	0.99825	0.007	665	653	10
7000	0.99825	0.008	665	653	10
7000	0.99825	0.009	665	653	10
7000	0.99825	0.010	665	653	10

Cuadro 1.6: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
5000	0.998	0.001	513
5000	0.998	0.002	513
5000	0.998	0.003	513
5000	0.998	0.004	513
5000	0.998	0.005	513

Resultados para el Conjunto 4 de Alphas:

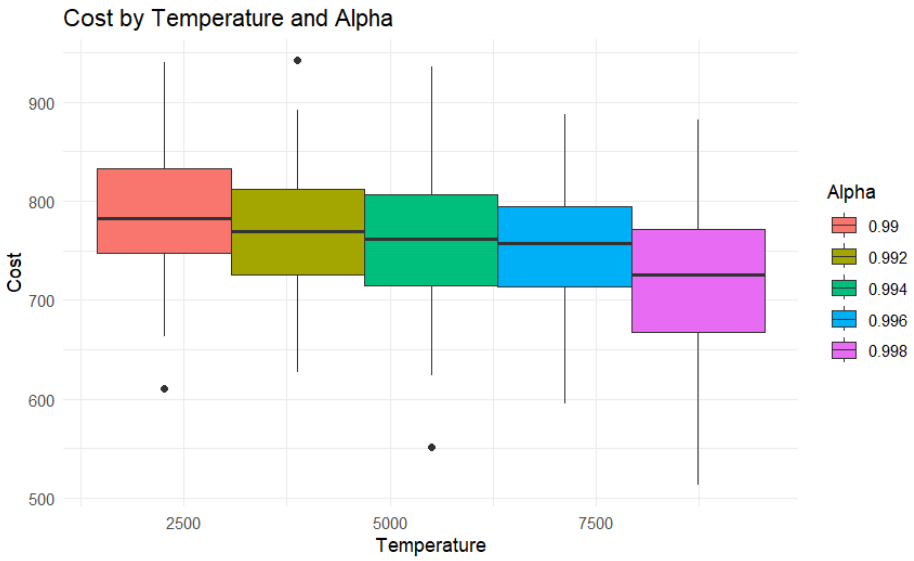

Figura 1.4: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.7: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
5000	0.998	0.001	689	702	10
5000	0.998	0.002	689	702	10
5000	0.998	0.003	689	702	10
5000	0.998	0.004	689	702	10
5000	0.998	0.005	689	702	10
5000	0.998	0.006	689	702	10
5000	0.998	0.007	689	702	10
5000	0.998	0.008	689	702	10
5000	0.998	0.009	689	702	10
5000	0.998	0.010	689	702	10

Cuadro 1.8: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
5000	0.998	0.001	513
5000	0.998	0.002	513
5000	0.998	0.003	513
5000	0.998	0.004	513
5000	0.998	0.005	513

Caso 2

En este apartado se analiza el rendimiento de *Simulated Annealing* (SA) sobre la segunda matriz de 25×25 con valores de similitud en el rango $[0, 100]$. Para esta matriz, se llevaron a cabo múltiples ejecuciones de SA bajo distintas combinaciones de Temperature, α y Temp_{min}. A continuación, se incluyen cuatro tablas de resultados (*mean_cost*, *median_cost* y el número de ejecuciones *count*, siempre con 10 réplicas por configuración (con 10 *randoms* fijados):

Resultados para el Conjunto 1 de Alphas:

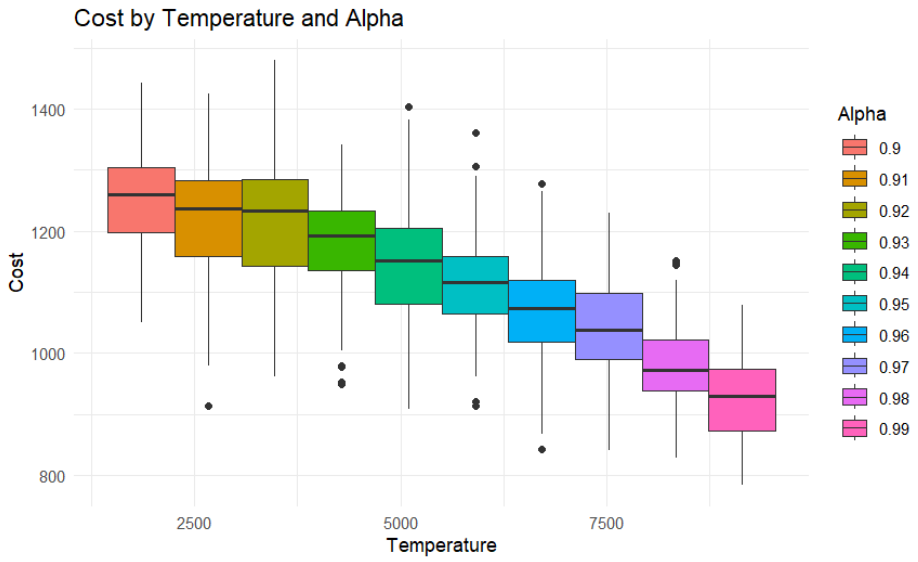

Figura 1.5: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.9: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
4000	0.99	0.001	889	895	10
4000	0.99	0.002	892	895	10
1000	0.99	0.001	897	889	10
5000	0.99	0.001	898	864	10
1000	0.99	0.002	900	889	10
1000	0.99	0.003	900	889	10
1000	0.99	0.004	900	889	10
1000	0.99	0.005	900	889	10
6000	0.99	0.001	901	922	10
6000	0.99	0.002	903	922	10

Cuadro 1.10: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
6000	0.99	0.001	784
6000	0.99	0.002	784
6000	0.99	0.003	784
6000	0.99	0.004	784
6000	0.99	0.005	784

Resultados para el Conjunto 2 de Alphas:

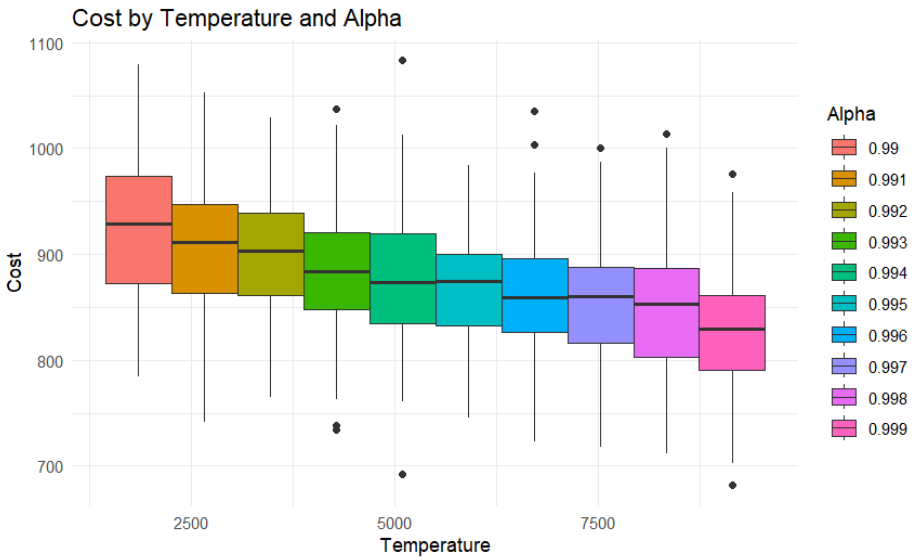

Figura 1.6: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.11: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
2000	0.999	0.001	801	809	10
2000	0.999	0.002	801	809	10
2000	0.999	0.003	801	809	10
2000	0.999	0.004	801	809	10
2000	0.999	0.005	801	809	10
2000	0.999	0.006	801	809	10
2000	0.999	0.007	801	809	10
2000	0.999	0.008	801	809	10
2000	0.999	0.009	801	809	10
2000	0.999	0.01	801	809	10

Cuadro 1.12: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
8000	0.999	0.001	682
8000	0.999	0.002	682
8000	0.999	0.003	682
8000	0.999	0.004	682
8000	0.999	0.005	682

Resultados para el Conjunto 3 de Alphas:

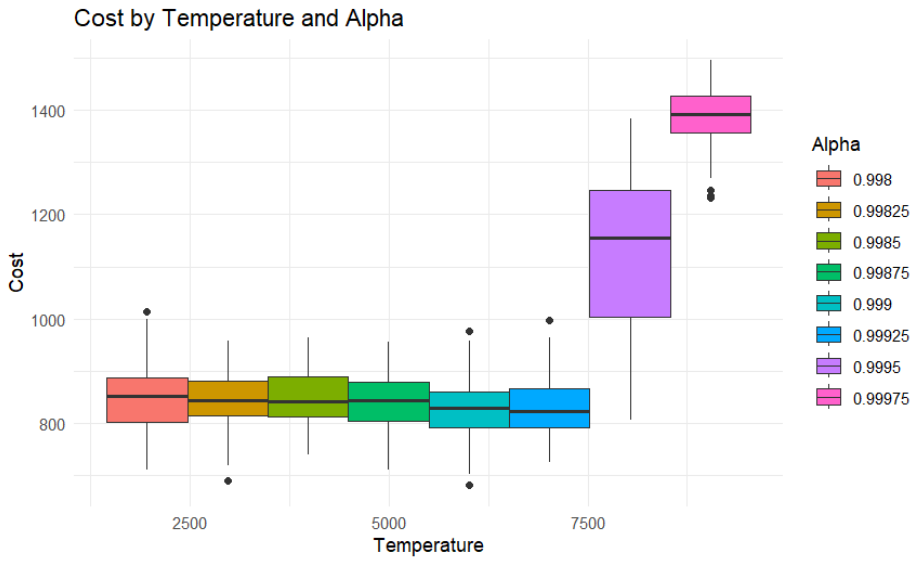

Figura 1.7: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.13: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
1000	0.99925	0.001	794	799	10
1000	0.99925	0.002	794	799	10
1000	0.99925	0.003	794	799	10
1000	0.99925	0.004	794	799	10
1000	0.99925	0.005	794	799	10
1000	0.99925	0.006	794	799	10
1000	0.99925	0.007	794	799	10
1000	0.99925	0.008	794	799	10
1000	0.99925	0.009	794	799	10
1000	0.99925	0.01	794	799	10

Cuadro 1.14: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
8000	0.999	0.001	682
8000	0.999	0.002	682
8000	0.999	0.003	682
8000	0.999	0.004	682
8000	0.999	0.005	682

Resultados para el Conjunto 4 de Alphas:

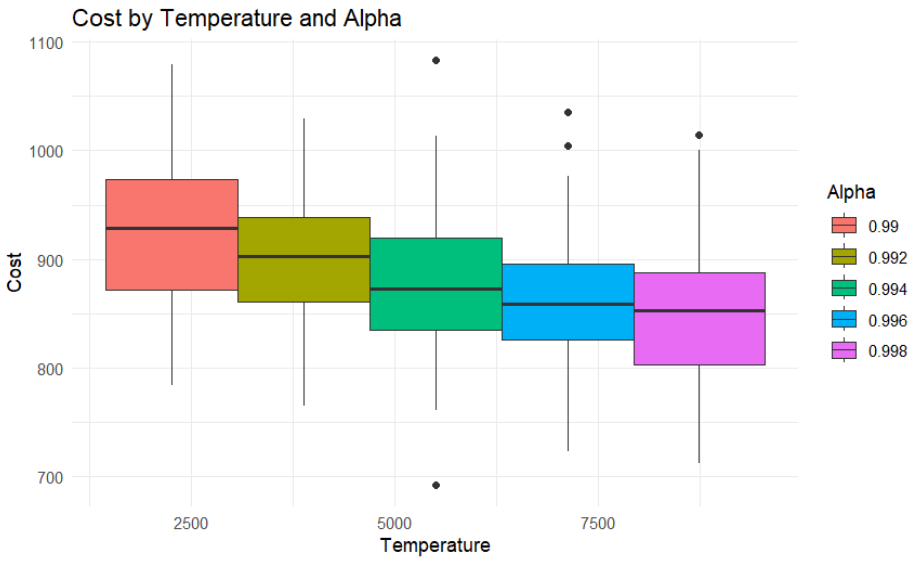

Figura 1.8: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.15: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
4000	0.996	0.001	816	814	10
4000	0.996	0.002	816	814	10
4000	0.996	0.003	823	822	10
4000	0.996	0.004	823	822	10
4000	0.996	0.005	823	822	10
4000	0.996	0.006	823	822	10
4000	0.996	0.007	823	822	10
4000	0.996	0.008	823	822	10
4000	0.996	0.009	823	822	10
4000	0.996	0.01	823	822	10

Cuadro 1.16: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
8000	0.994	0.001	692
8000	0.994	0.002	692
5000	0.998	0.001	712
5000	0.998	0.002	712
5000	0.998	0.003	712

Caso 3

En este apartado se analiza el rendimiento de *Simulated Annealing* (SA) sobre la tercera matriz de 25×25 con valores de similitud en el rango $[0, 100]$. Para esta matriz, se llevaron a cabo múltiples ejecuciones de SA bajo distintas combinaciones de Temperature, α y Temp_{min}. A continuación, se incluyen cuatro tablas de resultados (*mean_cost*, *median_cost* y el número de ejecuciones *count*, siempre con 10 réplicas por configuración (con 10 *randoms* fijados):

Resultados para el Conjunto 1 de Alphas:

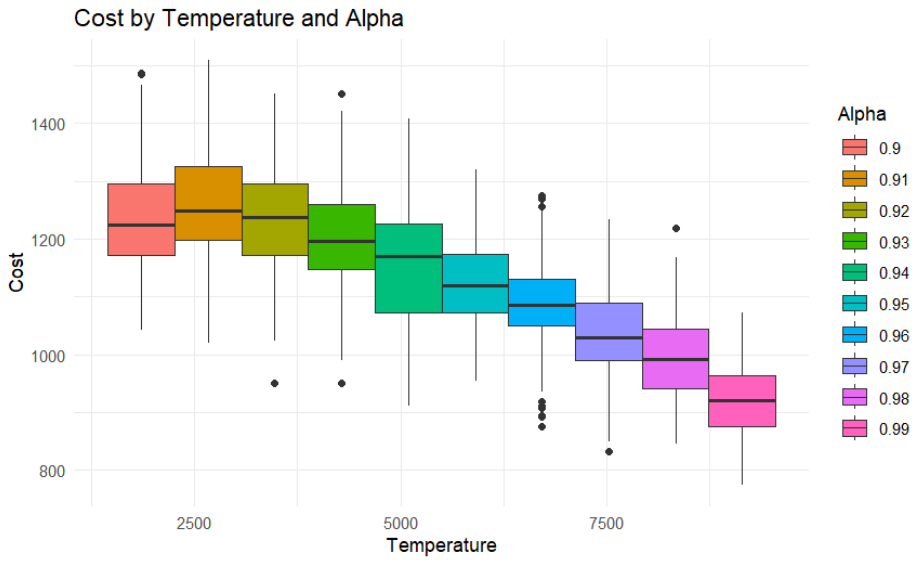

Figura 1.9: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.17: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
4000	0.99	0.001	858	838	10
4000	0.99	0.002	865	838	10
4000	0.99	0.003	874	861	10
4000	0.99	0.004	874	861	10
4000	0.99	0.005	874	861	10
4000	0.99	0.006	878	875	10
4000	0.99	0.007	878	875	10
4000	0.99	0.008	880	875	10
4000	0.99	0.009	884	890	10
4000	0.99	0.01	884	890	10

Cuadro 1.18: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
4000	0.99	0.001	774
4000	0.99	0.002	774
4000	0.99	0.003	774
4000	0.99	0.004	774
4000	0.99	0.005	774

Resultados para el Conjunto 2 de Alphas:

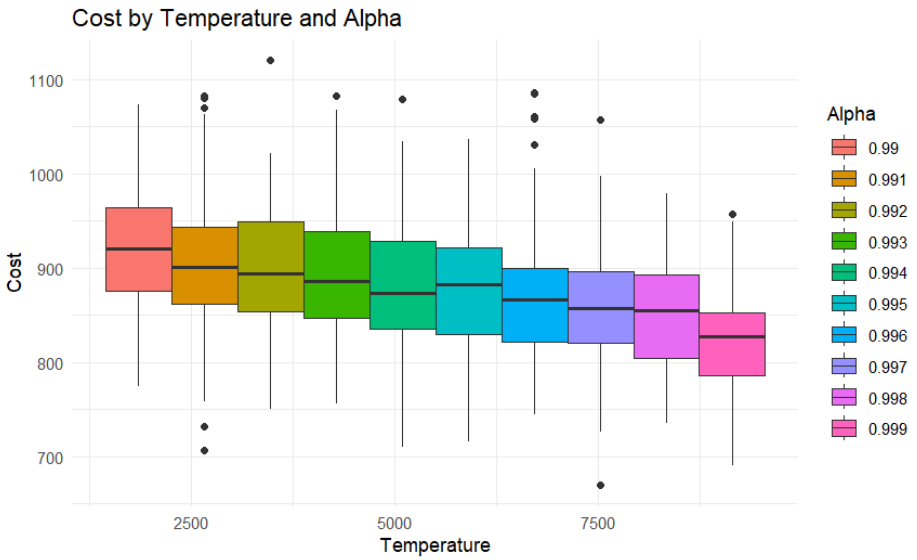

Figura 1.10: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.19: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
5000	0.999	0.001	803	800	10
5000	0.999	0.002	803	800	10
5000	0.999	0.003	803	800	10
5000	0.999	0.004	803	800	10
5000	0.999	0.005	803	800	10
5000	0.999	0.006	803	800	10
5000	0.999	0.007	803	800	10
5000	0.999	0.008	803	800	10
5000	0.999	0.009	803	800	10
5000	0.999	0.01	803	800	10

Cuadro 1.20: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
2000	0.997	0.001	670
2000	0.997	0.002	670
2000	0.997	0.003	670
2000	0.997	0.004	670
2000	0.997	0.005	670

Resultados para el Conjunto 3 de Alphas:

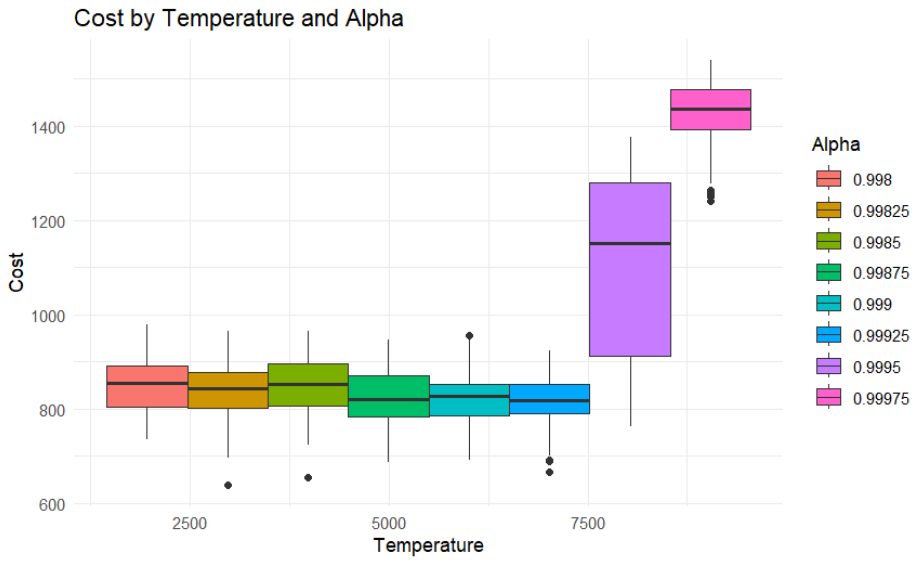

Figura 1.11: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.21: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
1000	0.99925	0.001	781	783	10
1000	0.99925	0.002	781	783	10
1000	0.99925	0.003	781	783	10
1000	0.99925	0.004	781	783	10
1000	0.99925	0.005	781	783	10
1000	0.99925	0.006	781	783	10
1000	0.99925	0.007	781	783	10
1000	0.99925	0.008	781	783	10
1000	0.99925	0.009	781	783	10
1000	0.99925	0.01	781	783	10

Cuadro 1.22: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
3000	0.99825	0.001	639
3000	0.99825	0.002	639
3000	0.99825	0.003	639
3000	0.99825	0.004	639
3000	0.99825	0.005	639

Resultados para el Conjunto 4 de Alphas:

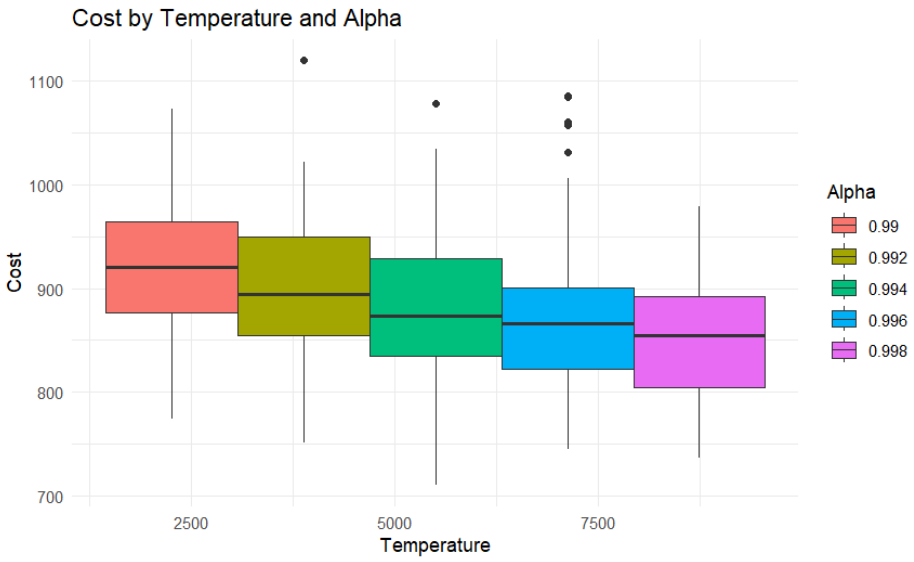

Figura 1.12: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.23: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
10000	0.998	0.001	823	810	10
10000	0.998	0.002	823	810	10
10000	0.998	0.003	823	810	10
10000	0.998	0.004	823	810	10
10000	0.998	0.005	823	810	10
10000	0.998	0.006	823	810	10
10000	0.998	0.007	823	810	10
10000	0.998	0.008	823	810	10
10000	0.998	0.009	823	810	10
10000	0.998	0.01	823	810	10

Cuadro 1.24: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
1000	0.994	0.001	710
1000	0.994	0.002	710
1000	0.994	0.003	710
1000	0.994	0.004	710
1000	0.994	0.005	710

1.3.3. 3 Matrices de 25 Productos con coeficientes de similitud en un interval de [80,100]

Caso 1

Resultados para el Conjunto 1 de Alphas:

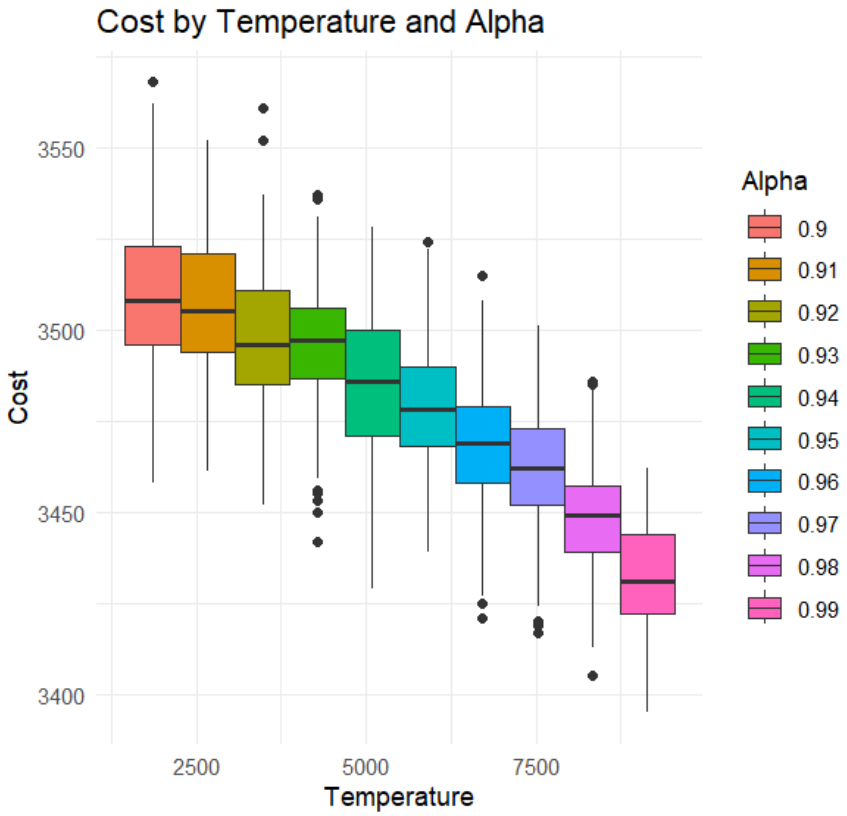

Figura 1.13: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.25: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
4000	0.99	0.001	3424	3427	10
2000	0.99	0.001	3425	3424	10
4000	0.99	0.002	3425	3427	10
2000	0.99	0.002	3425	3425	10
2000	0.99	0.003	3426	3425	10
2000	0.99	0.004	3426	3426	10
2000	0.99	0.005	3426	3426	10
2000	0.99	0.006	3427	3426	10
2000	0.99	0.007	3427	3426	10
2000	0.99	0.008	3427	3426	10

Cuadro 1.26: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
4000	0.99	0.001	3395
4000	0.99	0.002	3395
7000	0.99	0.001	3403
7000	0.99	0.002	3403
7000	0.99	0.003	3403

Resultados para el Conjunto 2 de Alphas:

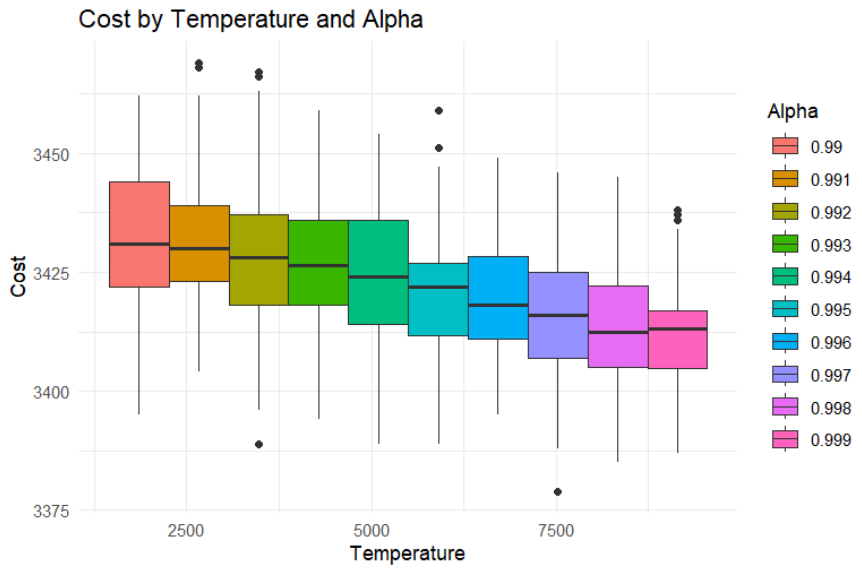

Figura 1.14: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.27: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
4000	0.998	0.001	3405.5	3406.5	10
4000	0.998	0.002	3405.5	3406.5	10
4000	0.998	0.003	3405.5	3406.5	10
4000	0.998	0.004	3405.5	3406.5	10
4000	0.998	0.005	3405.5	3406.5	10
4000	0.998	0.006	3405.5	3406.5	10
4000	0.998	0.007	3405.5	3406.5	10
4000	0.998	0.008	3405.5	3406.5	10
4000	0.998	0.009	3405.5	3406.5	10
4000	0.998	0.01	3405.5	3406.5	10

Cuadro 1.28: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
6000	0.997	0.001	3379
6000	0.997	0.002	3379
6000	0.997	0.003	3379
6000	0.997	0.004	3379
6000	0.997	0.005	3379

Resultados para el Conjunto 3 de Alphas:

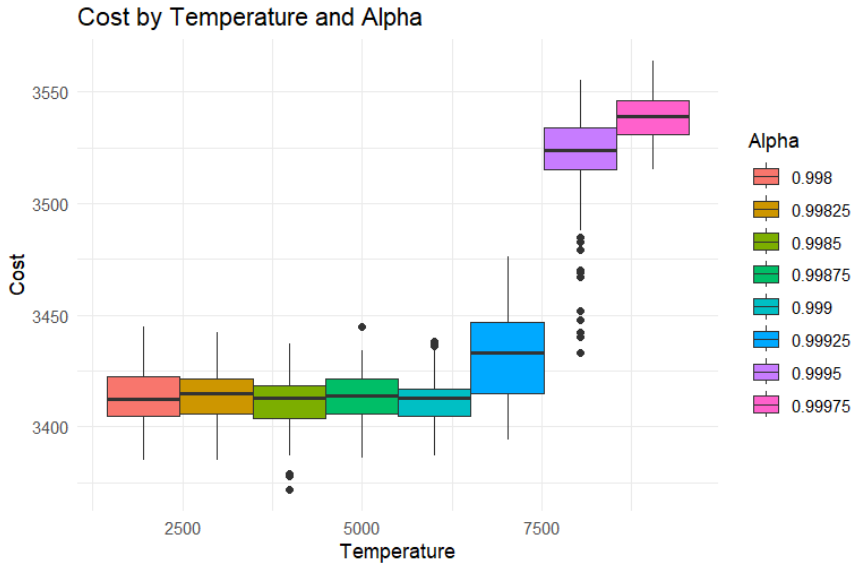

Figura 1.15: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.29: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
7000	0.99825	0.002	665	653	10
7000	0.99825	0.003	665	653	10
7000	0.99825	0.004	665	653	10
7000	0.99825	0.005	665	653	10
7000	0.99825	0.006	665	653	10
7000	0.99825	0.007	665	653	10
7000	0.99825	0.008	665	653	10
7000	0.99825	0.009	665	653	10
7000	0.99825	0.010	665	653	10

Cuadro 1.30: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
1000	0.9985	0.001	3372
1000	0.9985	0.002	3372
1000	0.9985	0.003	3372
1000	0.9985	0.004	3372
1000	0.9985	0.005	3372

Resultados para el Conjunto 4 de Alphas:

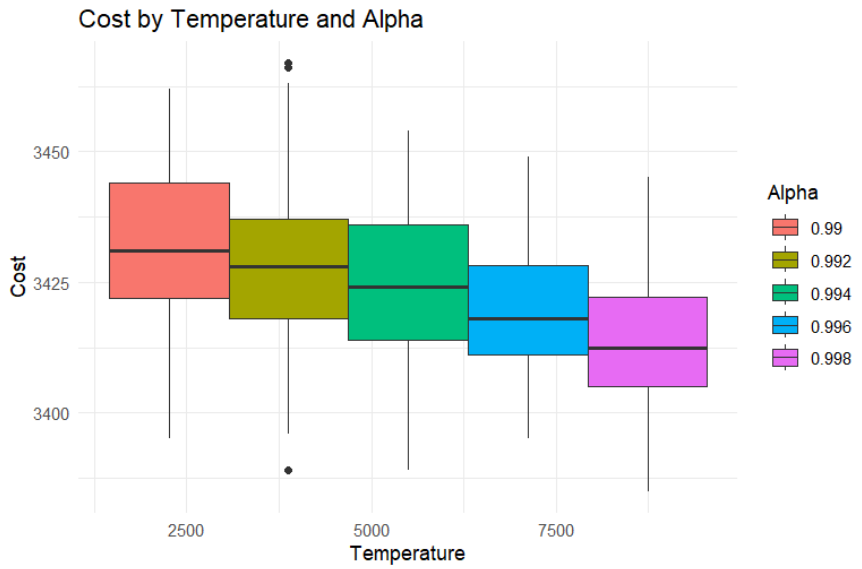

Figura 1.16: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.31: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
5000	0.998	0.001	689	702	10
5000	0.998	0.002	689	702	10
5000	0.998	0.003	689	702	10
5000	0.998	0.004	689	702	10
5000	0.998	0.005	689	702	10
5000	0.998	0.006	689	702	10
5000	0.998	0.007	689	702	10
5000	0.998	0.008	689	702	10
5000	0.998	0.009	689	702	10
5000	0.998	0.010	689	702	10

Cuadro 1.32: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
4000	0.998	0.001	3385
4000	0.998	0.002	3385
4000	0.998	0.003	3385
4000	0.998	0.004	3385
4000	0.998	0.005	3385

Caso 2

Resultados para el Conjunto 1 de Alphas:

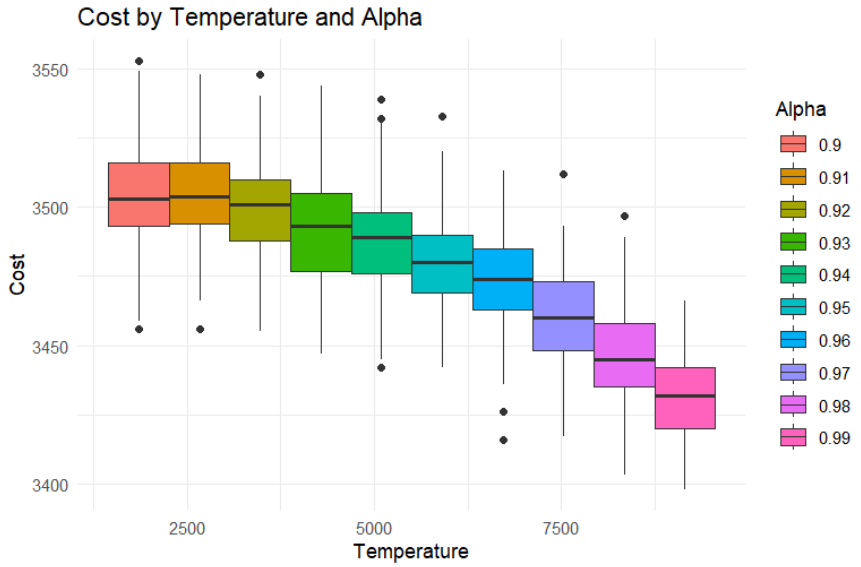

Figura 1.17: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.33: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
4000	0.99	0.001	3420.3	3420	10
4000	0.99	0.002	3422.5	3425.5	10
6000	0.99	0.001	3424	3421	10
4000	0.99	0.003	3424.4	3430	10
3000	0.99	0.001	3425.1	3430	10
9000	0.99	0.001	3425.3	3421	10
10000	0.99	0.001	3425.8	3422.5	10
6000	0.99	0.002	3426.1	3423.5	10
6000	0.99	0.003	3426.5	3423.5	10
3000	0.99	0.002	3426.6	3430.5	10

Cuadro 1.34: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
3000	0.99	0.001	3398
3000	0.99	0.002	3398
3000	0.99	0.003	3398
3000	0.99	0.004	3398
3000	0.99	0.005	3398

Resultados para el Conjunto 2 de Alphas:

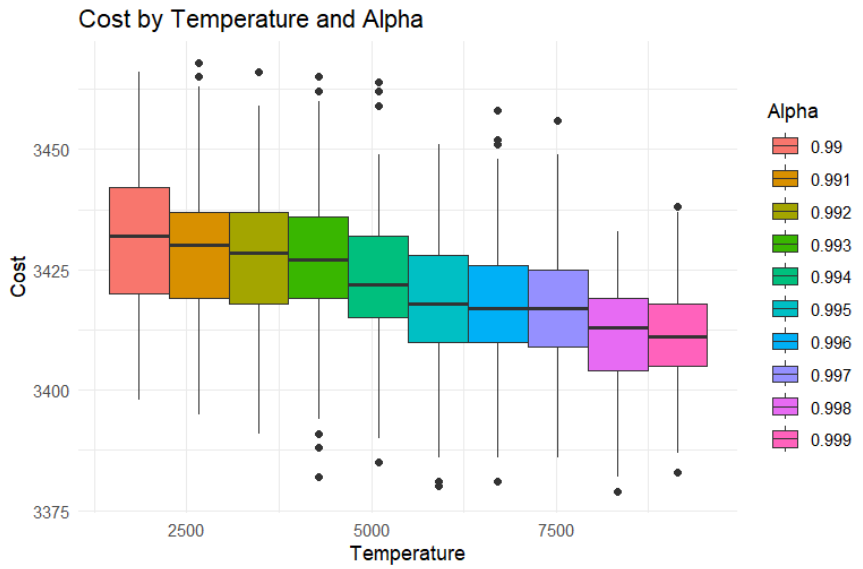

Figura 1.18: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.35: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
7000	0.998	0.001	3405.1	3402	10
7000	0.998	0.002	3405.1	3402	10
7000	0.998	0.003	3405.1	3402	10
7000	0.998	0.004	3405.1	3402	10
7000	0.998	0.005	3405.1	3402	10
7000	0.998	0.006	3405.1	3402	10
7000	0.998	0.007	3405.1	3402	10
7000	0.998	0.008	3405.1	3402	10
7000	0.998	0.009	3405.1	3402	10
7000	0.998	0.01	3405.1	3402	10

Cuadro 1.36: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
2000	0.998	0.001	3379
2000	0.998	0.002	3379
2000	0.998	0.003	3379
2000	0.998	0.004	3379
2000	0.998	0.005	3379

Resultados para el Conjunto 3 de Alphas:

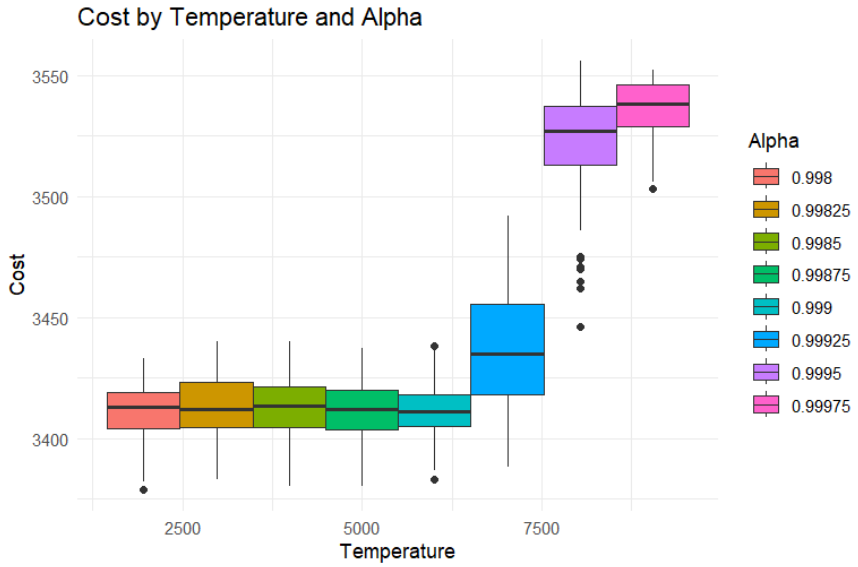

Figura 1.19: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.37: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
7000	0.998	0.001	3405.1	3402	10
7000	0.998	0.002	3405.1	3402	10
7000	0.998	0.003	3405.1	3402	10
7000	0.998	0.004	3405.1	3402	10
7000	0.998	0.005	3405.1	3402	10
7000	0.998	0.006	3405.1	3402	10
7000	0.998	0.007	3405.1	3402	10
7000	0.998	0.008	3405.1	3402	10
7000	0.998	0.009	3405.1	3402	10
7000	0.998	0.01	3405.1	3402	10

Cuadro 1.38: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
2000	0.998	0.001	3379
2000	0.998	0.002	3379
2000	0.998	0.003	3379
2000	0.998	0.004	3379
2000	0.998	0.005	3379

Resultados para el Conjunto 4 de Alphas:

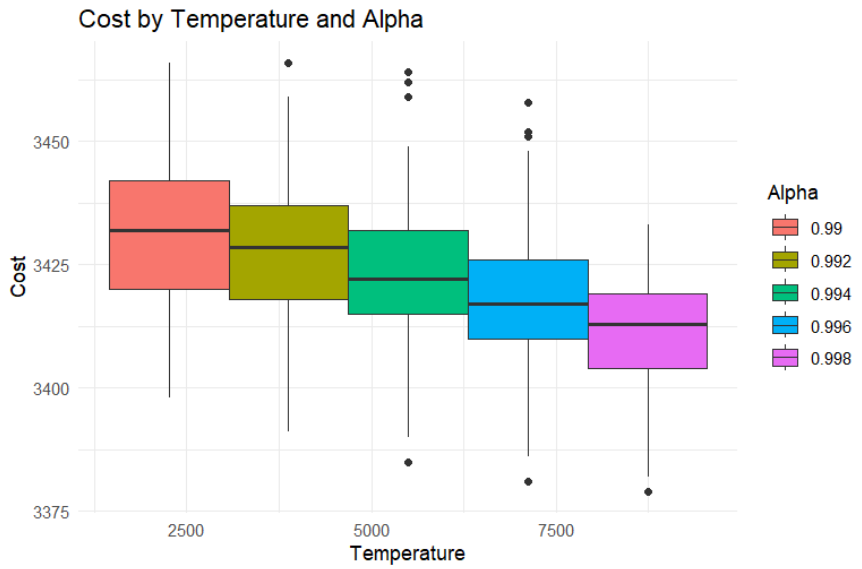

Figura 1.20: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.39: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
7000	0.998	0.001	3405.1	3402	10
7000	0.998	0.002	3405.1	3402	10
7000	0.998	0.003	3405.1	3402	10
7000	0.998	0.004	3405.1	3402	10
7000	0.998	0.005	3405.1	3402	10
7000	0.998	0.006	3405.1	3402	10
7000	0.998	0.007	3405.1	3402	10
7000	0.998	0.008	3405.1	3402	10
7000	0.998	0.009	3405.1	3402	10
7000	0.998	0.01	3405.1	3402	10

Cuadro 1.40: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
2000	0.998	0.001	3379
2000	0.998	0.002	3379
2000	0.998	0.003	3379
2000	0.998	0.004	3379
2000	0.998	0.005	3379

Caso 3

Resultados para el Conjunto 1 de Alphas:

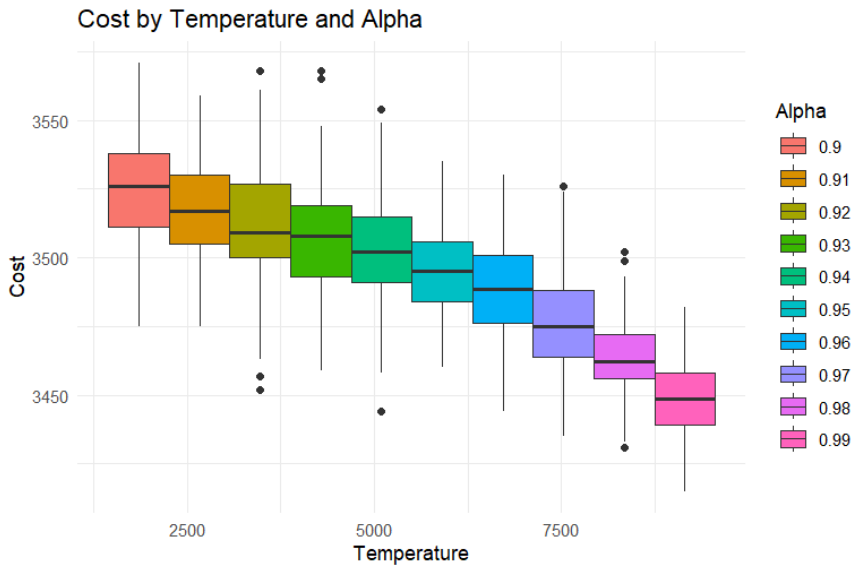

Figura 1.21: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.41: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
7000	0.99	0.001	3440.2	3443.5	10
6000	0.99	0.001	3441.6	3441	10
6000	0.99	0.002	3441.6	3441	10
7000	0.99	0.002	3442.3	3446.5	10
6000	0.99	0.003	3443.1	3441	10
4000	0.99	0.001	3443.2	3442	10
10000	0.99	0.001	3444	3447.5	10
10000	0.99	0.002	3444	3447.5	10
10000	0.99	0.003	3444	3447.5	10
4000	0.99	0.002	3444.7	3443.5	10

Cuadro 1.42: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
7000	0.99	0.001	3415
7000	0.99	0.002	3415
7000	0.99	0.003	3415
7000	0.99	0.004	3415
7000	0.99	0.005	3415

Resultados para el Conjunto 2 de Alphas:

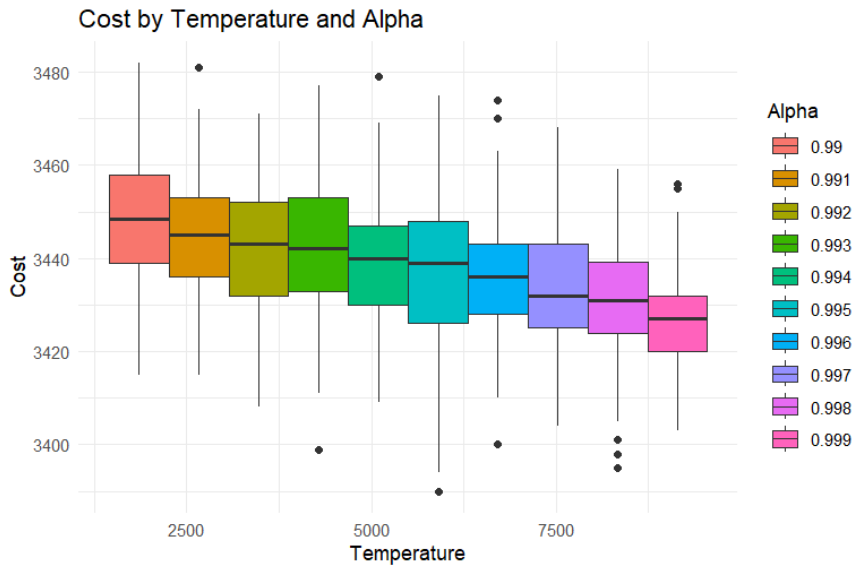

Figura 1.22: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.43: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
1000	0.999	0.001	3420.1	3420	10
1000	0.999	0.002	3420.1	3420	10
1000	0.999	0.003	3420.1	3420	10
1000	0.999	0.004	3420.1	3420	10
1000	0.999	0.005	3420.1	3420	10
1000	0.999	0.006	3420.1	3420	10
1000	0.999	0.007	3420.1	3420	10
1000	0.999	0.008	3420.1	3420	10
1000	0.999	0.009	3420.1	3420	10
1000	0.999	0.01	3420.1	3420	10

Cuadro 1.44: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
3000	0.995	0.001	3390
3000	0.995	0.002	3390
3000	0.995	0.003	3390
3000	0.995	0.004	3390
3000	0.995	0.005	3390

Resultados para el Conjunto 3 de Alphas:

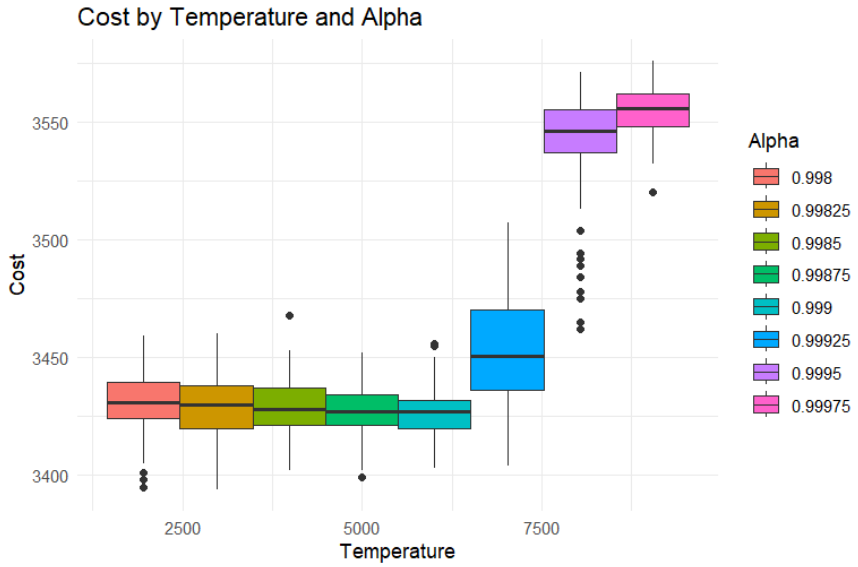

Figura 1.23: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.45: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
1000	0.999	0.001	3420.1	3420	10
1000	0.999	0.002	3420.1	3420	10
1000	0.999	0.003	3420.1	3420	10
1000	0.999	0.004	3420.1	3420	10
1000	0.999	0.005	3420.1	3420	10
1000	0.999	0.006	3420.1	3420	10
1000	0.999	0.007	3420.1	3420	10
1000	0.999	0.008	3420.1	3420	10
1000	0.999	0.009	3420.1	3420	10
1000	0.999	0.01	3420.1	3420	10

Cuadro 1.46: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
5000	0.99825	0.001	3394
5000	0.99825	0.002	3394
5000	0.99825	0.003	3394
5000	0.99825	0.004	3394
5000	0.99825	0.005	3394

Resultados para el Conjunto 4 de Alphas:

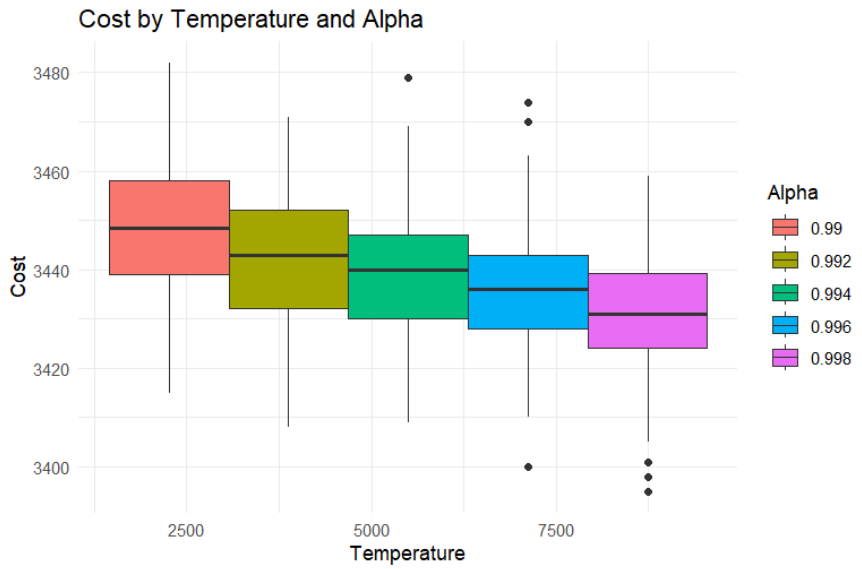

Figura 1.24: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.47: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
2000	0.998	0.001	3422.3	3424	10
2000	0.998	0.002	3422.3	3424	10
2000	0.998	0.003	3422.3	3424	10
2000	0.998	0.004	3422.3	3424	10
2000	0.998	0.005	3422.3	3424	10
2000	0.998	0.006	3422.3	3424	10
2000	0.998	0.007	3422.3	3424	10
2000	0.998	0.008	3422.3	3424	10
2000	0.998	0.009	3422.3	3424	10
2000	0.998	0.01	3422.3	3424	10

Cuadro 1.48: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
2000	0.998	0.001	3395
2000	0.998	0.002	3395
2000	0.998	0.003	3395
2000	0.998	0.004	3395
2000	0.998	0.005	3395

1.3.4. 3 Matrices de 25 Productos con coeficientes de similitud en un interval de [0,20]

Caso 1

Resultados para el Conjunto 1 de Alphas:

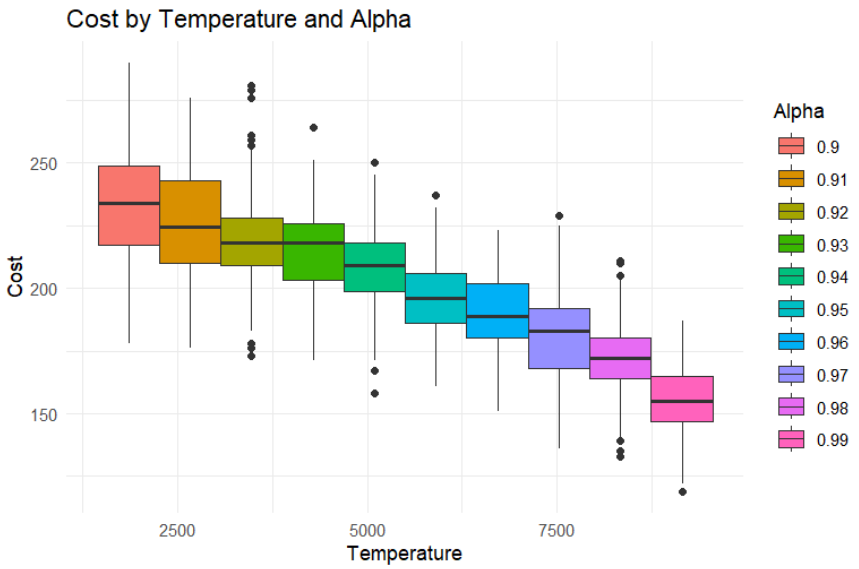

Figura 1.25: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.49: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
7000	0.99	0.001	145.5	146.5	10
7000	0.99	0.002	146.8	147	10
7000	0.99	0.003	147.4	147.5	10
7000	0.99	0.004	147.4	147.5	10
9000	0.99	0.001	147.4	146.5	10
7000	0.99	0.005	147.5	147.5	10
7000	0.99	0.006	147.8	148	10
7000	0.99	0.007	149	148	10
9000	0.99	0.002	150.3	146.5	10
7000	0.99	0.008	150.5	149	10

Cuadro 1.50: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
9000	0.99	0.001	119
5000	0.99	0.001	122
5000	0.99	0.002	122
5000	0.99	0.003	122
5000	0.99	0.004	122

Resultados para el Conjunto 2 de Alphas:

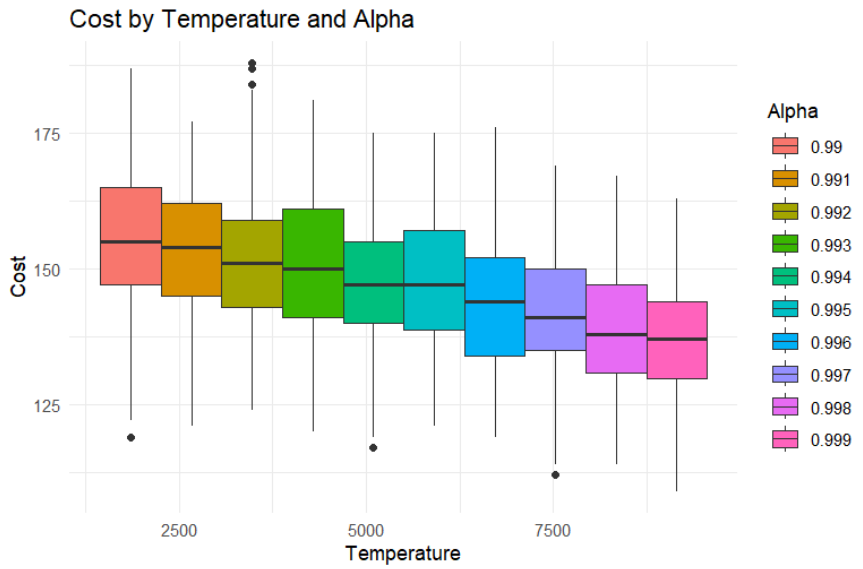

Figura 1.26: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.51: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
9000	0.998	0.001	130.7	125.5	10
9000	0.998	0.002	130.7	125.5	10
9000	0.998	0.003	130.7	125.5	10
9000	0.998	0.004	130.7	125.5	10
9000	0.998	0.005	130.7	125.5	10
9000	0.998	0.006	130.7	125.5	10
9000	0.998	0.007	130.7	125.5	10
9000	0.998	0.008	130.7	125.5	10
9000	0.998	0.009	130.7	125.5	10
9000	0.998	0.01	130.7	125.5	10

Cuadro 1.52: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
2000	0.999	0.001	109
2000	0.999	0.002	109
2000	0.999	0.003	109
2000	0.999	0.004	109
2000	0.999	0.005	109

Resultados para el Conjunto 3 de Alphas:

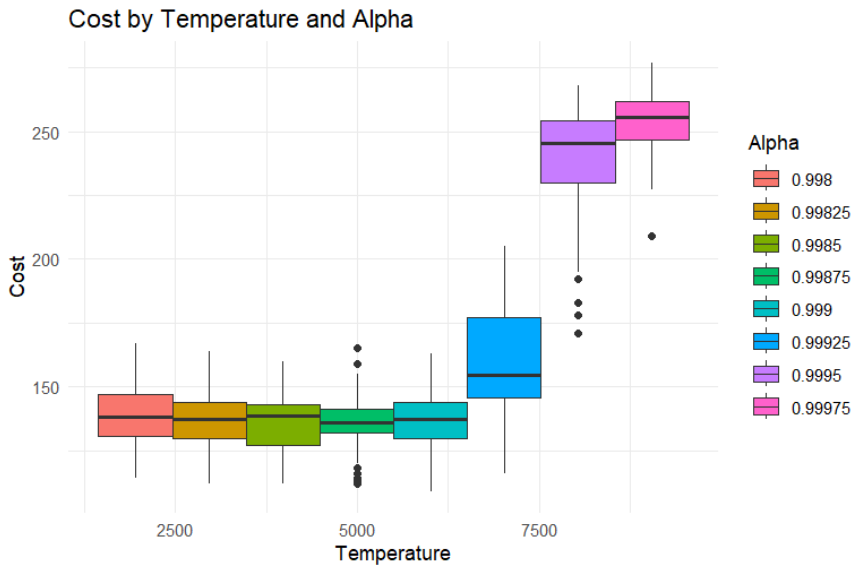

Figura 1.27: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.53: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
10000	0.99875	0.001	130.6	133	10
10000	0.99875	0.002	130.6	133	10
10000	0.99875	0.003	130.6	133	10
10000	0.99875	0.004	130.6	133	10
10000	0.99875	0.005	130.6	133	10
10000	0.99875	0.006	130.6	133	10
10000	0.99875	0.007	130.6	133	10
10000	0.99875	0.008	130.6	133	10
10000	0.99875	0.009	130.6	133	10
10000	0.99875	0.01	130.6	133	10

Cuadro 1.54: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
2000	0.999	0.001	109
2000	0.999	0.002	109
2000	0.999	0.003	109
2000	0.999	0.004	109
2000	0.999	0.005	109

Resultados para el Conjunto 4 de Alphas:

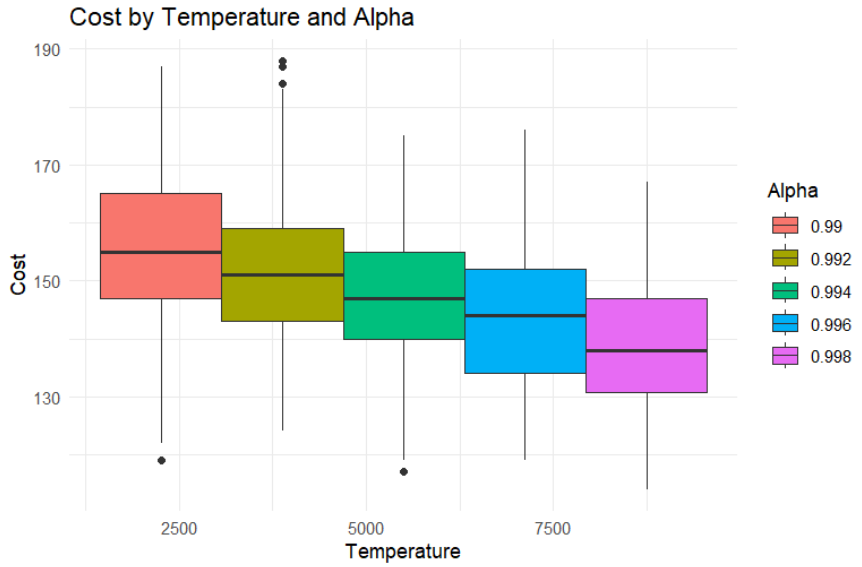

Figura 1.28: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.55: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
9000	0.998	0.001	130.7	125.5	10
9000	0.998	0.002	130.7	125.5	10
9000	0.998	0.003	130.7	125.5	10
9000	0.998	0.004	130.7	125.5	10
9000	0.998	0.005	130.7	125.5	10
9000	0.998	0.006	130.7	125.5	10
9000	0.998	0.007	130.7	125.5	10
9000	0.998	0.008	130.7	125.5	10
9000	0.998	0.009	130.7	125.5	10
9000	0.998	0.01	130.7	125.5	10

Cuadro 1.56: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
4000	0.998	0.001	114
4000	0.998	0.002	114
4000	0.998	0.003	114
4000	0.998	0.004	114
4000	0.998	0.005	114

Caso 2

Resultados para el Conjunto 1 de Alphas:

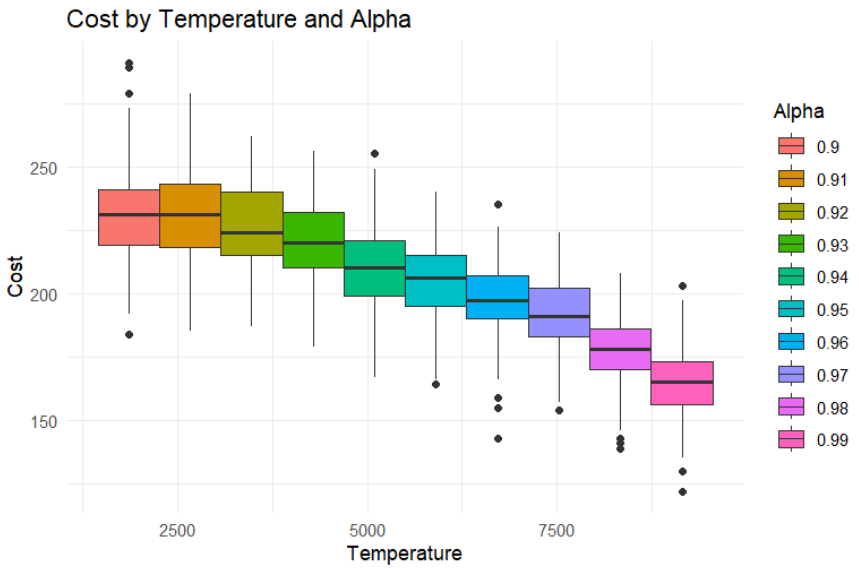

Figura 1.29: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.57: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
1000	0.99	0.001	155.5	155.5	10
10000	0.99	0.001	157.1	158.5	10
9000	0.99	0.001	157.9	157.5	10
9000	0.99	0.002	158.7	159	10
1000	0.99	0.002	159	160	10
2000	0.99	0.001	159	158	10
9000	0.99	0.003	159.5	159.5	10
9000	0.99	0.004	159.7	160.5	10
9000	0.99	0.005	159.7	160.5	10
6000	0.99	0.001	160	162	10

Cuadro 1.58: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
2000	0.99	0.001	122
2000	0.99	0.002	122
2000	0.99	0.003	122
2000	0.99	0.004	122
2000	0.99	0.005	122

Resultados para el Conjunto 2 de Alphas:

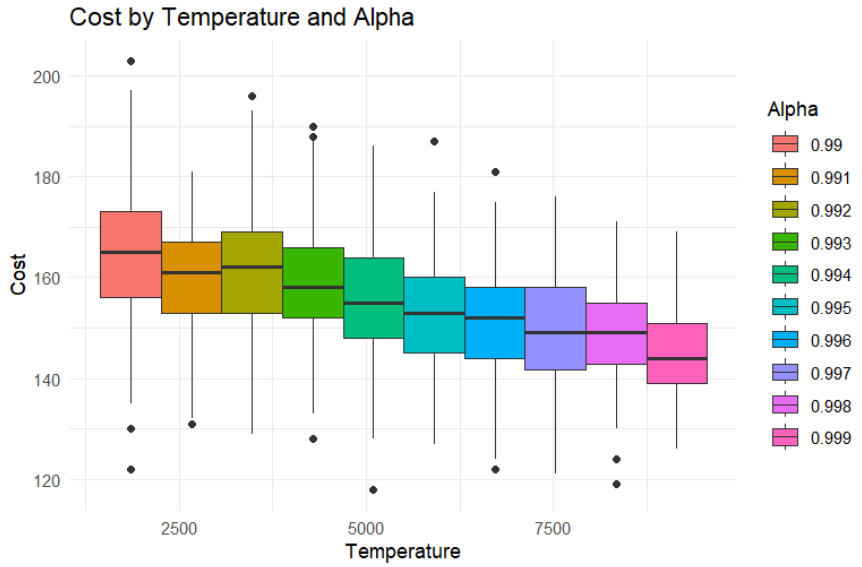

Figura 1.30: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.59: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
5000	0.999	0.001	139.4	139	10
5000	0.999	0.002	139.4	139	10
5000	0.999	0.003	139.4	139	10
5000	0.999	0.004	139.4	139	10
5000	0.999	0.005	139.4	139	10
5000	0.999	0.006	139.4	139	10
5000	0.999	0.007	139.4	139	10
5000	0.999	0.008	139.4	139	10
5000	0.999	0.009	139.4	139	10
5000	0.999	0.01	139.4	139	10

Cuadro 1.60: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
3000	0.994	0.001	118
10000	0.998	0.001	119
10000	0.998	0.002	119
10000	0.998	0.003	119
10000	0.998	0.004	119

Resultados para el Conjunto 3 de Alphas:

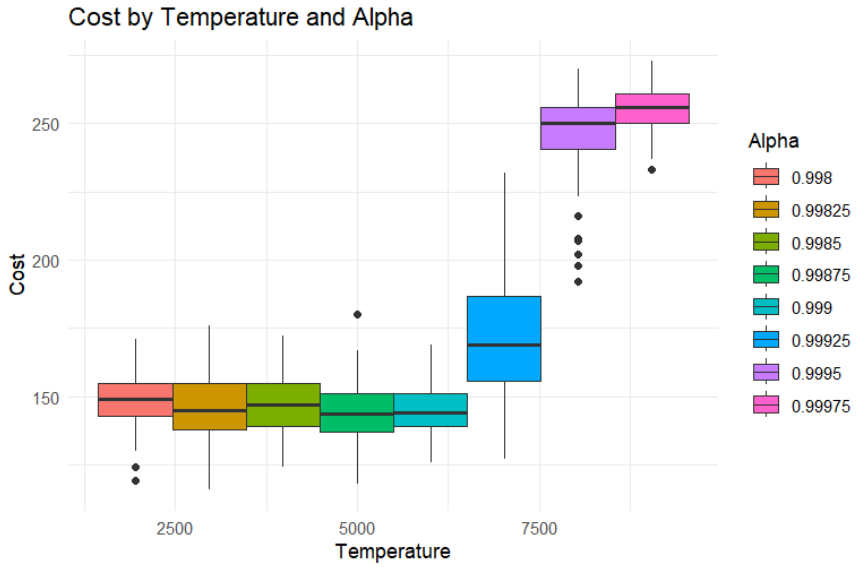

Figura 1.31: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.61: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
9000	0.99825	0.001	138.6	140	10
9000	0.99825	0.002	138.6	140	10
9000	0.99825	0.003	138.6	140	10
9000	0.99825	0.004	138.6	140	10
9000	0.99825	0.005	138.6	140	10
9000	0.99825	0.006	138.6	140	10
9000	0.99825	0.007	138.6	140	10
9000	0.99825	0.008	138.6	140	10
9000	0.99825	0.009	138.6	140	10
9000	0.99825	0.01	138.6	140	10

Cuadro 1.62: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
4000	0.99825	0.001	116
4000	0.99825	0.002	116
4000	0.99825	0.003	116
4000	0.99825	0.004	116
4000	0.99825	0.005	116

Resultados para el Conjunto 4 de Alphas:

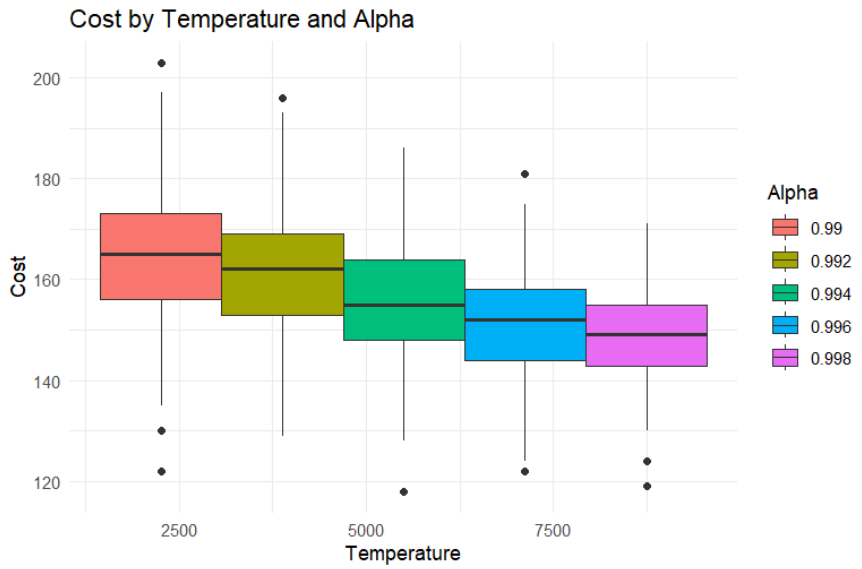

Figura 1.32: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.63: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
10000	0.998	0.001	142.3	148	10
10000	0.998	0.002	142.3	148	10
4000	0.996	0.001	142.7	143.5	10
10000	0.998	0.003	143	148	10
10000	0.998	0.004	143	148	10
10000	0.998	0.005	143	148	10
10000	0.998	0.006	143	148	10
4000	0.996	0.002	143.2	144.5	10
4000	0.996	0.003	143.2	144.5	10
4000	0.996	0.004	143.2	144.5	10

Cuadro 1.64: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
3000	0.994	0.001	118
10000	0.998	0.001	119
10000	0.998	0.002	119
10000	0.998	0.003	119
10000	0.998	0.004	119

Caso 3

Resultados para el Conjunto 1 de Alphas:

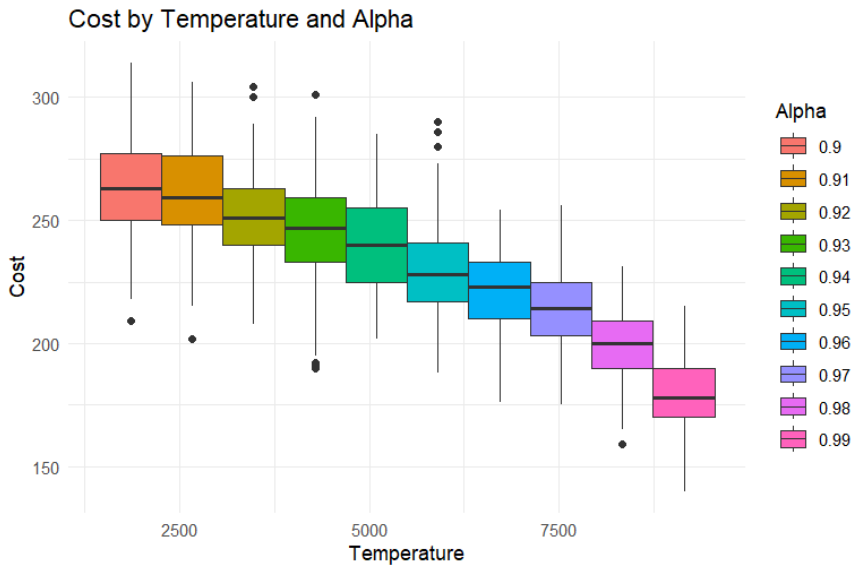

Figura 1.33: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.65: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
5000	0.99	0.001	167.3	166	10
5000	0.99	0.002	167.3	166	10
5000	0.99	0.003	167.3	166	10
5000	0.99	0.004	167.3	166	10
5000	0.99	0.005	167.7	166	10
5000	0.99	0.006	167.7	166	10
5000	0.99	0.007	167.7	166	10
5000	0.99	0.008	167.7	166	10
5000	0.99	0.009	168	167	10
5000	0.99	0.01	168	167	10

Cuadro 1.66: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
8000	0.99	0.001	140
8000	0.99	0.002	140
8000	0.99	0.003	140
8000	0.99	0.004	140
8000	0.99	0.005	140

Resultados para el Conjunto 2 de Alphas:

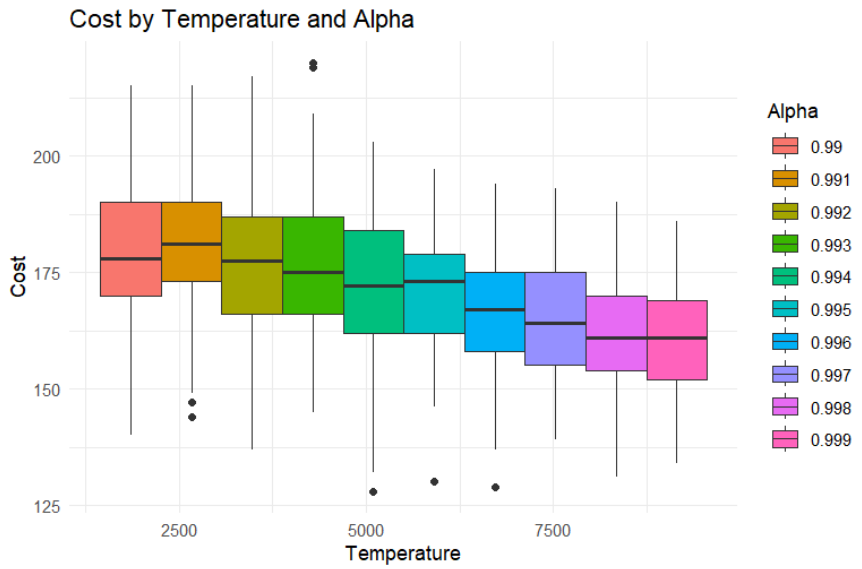

Figura 1.34: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.67: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
1000	0.999	0.001	152.5	149.5	10
1000	0.999	0.002	152.5	149.5	10
1000	0.999	0.003	152.5	149.5	10
1000	0.999	0.004	152.5	149.5	10
1000	0.999	0.005	152.5	149.5	10
1000	0.999	0.006	152.5	149.5	10
1000	0.999	0.007	152.5	149.5	10
1000	0.999	0.008	152.5	149.5	10
1000	0.999	0.009	152.5	149.5	10
1000	0.999	0.01	152.5	149.5	10

Cuadro 1.68: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
1000	0.994	0.001	128
1000	0.994	0.002	128
1000	0.994	0.003	128
1000	0.994	0.004	128
1000	0.994	0.005	128

Resultados para el Conjunto 3 de Alphas:

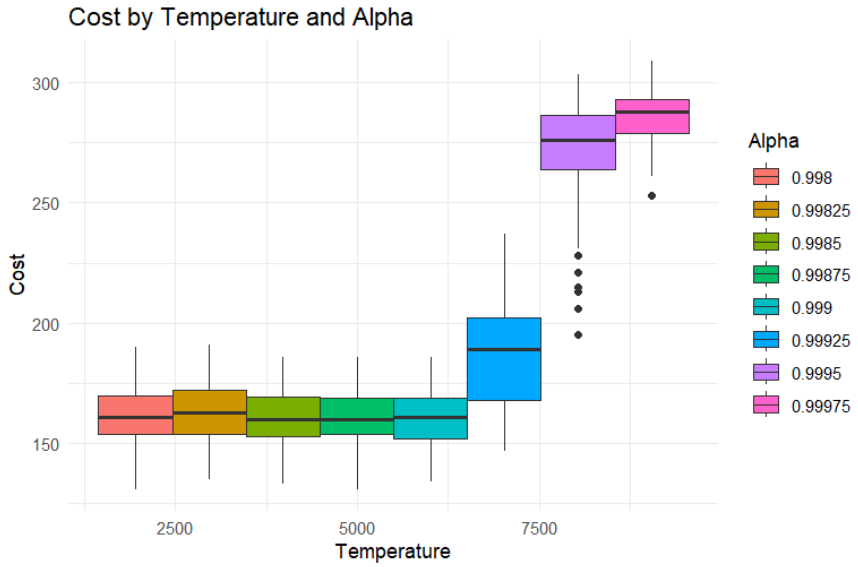

Figura 1.35: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.69: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
9000	0.99875	0.001	152.1	151.5	10
9000	0.99875	0.002	152.1	151.5	10
9000	0.99875	0.003	152.1	151.5	10
9000	0.99875	0.004	152.1	151.5	10
9000	0.99875	0.005	152.1	151.5	10
9000	0.99875	0.006	152.1	151.5	10
9000	0.99875	0.007	152.1	151.5	10
9000	0.99875	0.008	152.1	151.5	10
9000	0.99875	0.009	152.1	151.5	10
9000	0.99875	0.01	152.1	151.5	10

Cuadro 1.70: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
9000	0.99875	0.001	131
9000	0.99875	0.002	131
9000	0.99875	0.003	131
9000	0.99875	0.004	131
9000	0.99875	0.005	131

Resultados para el Conjunto 4 de Alphas:

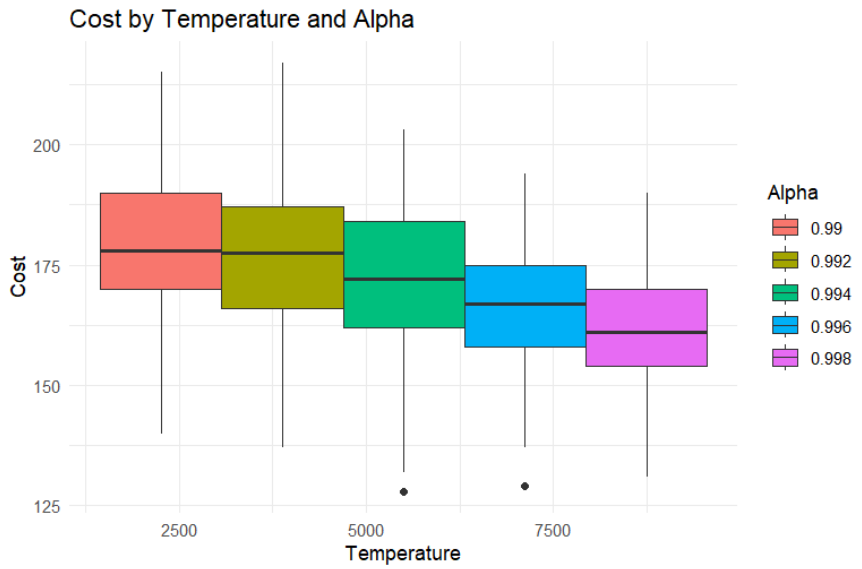

Figura 1.36: Distribución de costes en función de la temperatura inicial y el factor alpha

Cuadro 1.71: Top 10 mejores soluciones por parámetros (ordenados por media de Coste entre los 10 randoms)

Temperature	Alpha	MinTemperature	mean_cost	median_cost	count
10000	0.998	0.001	156.2	157	10
10000	0.998	0.002	157.3	160	10
10000	0.998	0.003	157.3	160	10
10000	0.998	0.004	157.3	160	10
10000	0.998	0.005	157.3	160	10
10000	0.998	0.006	157.3	160	10
10000	0.998	0.007	157.3	160	10
10000	0.998	0.008	157.3	160	10
10000	0.998	0.009	157.3	160	10
10000	0.998	0.01	157.3	160	10

Cuadro 1.72: Top 5 mejores soluciones por parámetros

Temperature	Alpha	MinTemperature	min_cost
1000	0.994	0.001	128
1000	0.994	0.002	128
1000	0.994	0.003	128
1000	0.994	0.004	128
1000	0.994	0.005	128

1.4. Discusión

Los resultados expuestos en la sección anterior permiten identificar patrones claros en la influencia de los diferentes parámetros sobre la calidad de las soluciones y el tiempo de cómputo del algoritmo *Simulated Annealing* (SA). A continuación, se detallan los hallazgos más relevantes:

1.4.1. Importancia del Factor de Enfriamiento (α)

Los experimentos confirman que α (tasa de enfriamiento) es el parámetro más determinante en la efectividad de SA:

- **Rango óptimo elevado.** En la mayoría de instancias analizadas—muy especialmente para las matrices con valores entre 0 y 100—se obtuvieron mejores resultados cuando $\alpha \approx 0{,}998$ en adelante. En particular, destacan $\alpha = 0{,}99825$ o $\alpha = 0{,}99925$ como configuraciones que arrojan costes menores y más consistentes.
- **Búsqueda más exhaustiva.** Al enfriar más lentamente, el algoritmo dispone de un margen mayor para aceptar vecinos de mayor coste en las primeras iteraciones. Esto favorece la exploración de distintas regiones del espacio de soluciones y reduce la probabilidad de quedar atrapado en óptimos locales prematuros.
- **Coste computacional.** El uso de α altos implica un aumento notable en el número de iteraciones efectivas en las que el algoritmo mantiene una probabilidad de aceptación significativa, alargando la ejecución. No obstante, para problemas de tamaño medio, la mejora en la calidad final suele compensar este sobrecoste.

Por el contrario, valores de $\alpha < 0{,}99$ tienden a "enfriar" con demasiada rapidez, reduciendo excesivamente la probabilidad de aceptación de peores soluciones y acortando la fase de exploración. En consecuencia, el algoritmo puede estancarse antes de alcanzar distribuciones de mayor calidad.

1.4.2. Incidencia Menor de la Temperatura Inicial (Temp)

Los boxplots muestran diferencias menos pronunciadas al variar la temperatura inicial entre 1000 y 10000:

- Temp **demasiado alta** (por encima de 10000) extiende la fase inicial de exploración sin mejoras claras, prolongando el tiempo de cómputo.
- Temp **muy baja** (alrededor de 1000) acorta la fase exploratoria al comienzo. Sin embargo, si α es elevado, la temperatura desciende con la suficiente lentitud como para compensar el arranque bajo.

En la práctica, un intervalo intermedio (3000–8000) combinado con α altos demostró funcionar de forma adecuada, manteniendo un buen equilibrio entre exploración y tiempo de ejecución.

1.4.3. Papel Secundario de la Temperatura Mínima (Temp_{min})

En cuanto a Temp_{min}, la mayoría de ejecuciones alcanzan valores muy bajos tras suficientes iteraciones, sobre todo con α grande. Por tanto, una vez que la temperatura es muy baja (p. ej., $< 0{,}01$), la probabilidad de aceptar soluciones de peor coste tiende a cero. En este escenario, el impacto de elegir 0,01 frente a 0,001 o incluso valores intermedios (0,005) no supone diferencias significativas en el coste final.

1.4.4. Criterios de Parada y su Influencia

Los límites maxIterations y maxNoImprovement se establecieron en valores intermedios (10000 y 2500, respectivamente) para asegurar que hubiese una fase de exploración lo bastante prolongada, sin caer en ejecuciones "infinitas". Efectivamente, *aumentar de forma indefinida* estos parámetros mejora (ligeramente) el coste medio a costa de un *tiempo de cómputo* sustancialmente mayor. Se reafirma así el clásico balance "calidad de solución vs. tiempo de cómputo".

1.4.5. Diferencias entre los Rangos de la Matriz de Flujo

Las pruebas con distintas distribuciones de valores en la matriz de flujo **F** (rango 0–100, 0–20 y 80–100) arrojan ligeras variaciones en el α más adecuado, lo que refleja cómo la *magnitud y dispersión* de costes influye en la dinámica de aceptación:

- **Valores** 0–100**.** Destacan $\alpha \approx 0{,}99825$ y $\alpha \approx 0{,}99925$ con temperaturas iniciales entre 4000 y 8000.

- **Valores** 0–20**.** Se obtienen resultados favorables con $\alpha \approx 0{,}9985$, apuntando a que un enfriamiento relativamente rápido en la fase inicial también puede ser provechoso cuando los costes tienen un rango más reducido.

- **Valores** 80–100**.** Se reporta que la mejor tasa de enfriamiento oscila en torno a $\alpha \approx 0{,}985$. Esto sugiere que, dado el rango de costes más homogéneo y alto, un enfriamiento algo menos lento sirve para distinguir gradualmente soluciones de menor coste sin incurrir en excesivo cómputo.

1.4.6. Implicaciones Prácticas

En síntesis, los experimentos permiten recomendar:

- α **altos** (entre 0,998 y 0,999) para instancias de cierta complejidad, priorizando la calidad sobre el tiempo de ejecución.

- Temperaturas iniciales **intermedias** (3000–8000) para evitar exploraciones iniciales excesivamente largas o excesivamente cortas.

- $\text{Temp}_{\text{min}} \leq 0{,}01$ en la mayoría de instancias, pues no se observan diferencias notables por afinar este umbral.

Como puede verse en los *boxplots* generados (Figura **??**), la dispersión de valores finales se reduce drásticamente al incrementar α. Sin embargo, dicha mejora conlleva tiempos de ejecución superiores. Por consiguiente, la selección de parámetros depende, en gran medida, de las restricciones temporales y del *tamaño de la instancia* que se pretende resolver.

1.5. Conclusión

Los resultados obtenidos corroboran que la correcta sintonización de los parámetros en *Simulated Annealing* (SA) para el QAP puede suponer la diferencia entre alcanzar una solución aceptable o lograr un resultado de calidad competitiva en tiempos razonables. En particular, se desprenden las siguientes reflexiones:

1. **El factor de enfriamiento (α) marca la diferencia principal:** Utilizar valores altos ($\alpha \approx$ 0,998–0,999) prolonga la fase de exploración, facilitando la salida de mínimos locales. Aunque ello conlleva un incremento notable del tiempo de cómputo, se ha demostrado muy efectivo para mejorar de forma consistente la calidad final de las soluciones. Concretamente, en las pruebas con matrices de flujo que abarcan el rango 0–100, se han observado comportamientos destacados en $\alpha = 0{,}99825$ y $\alpha = 0{,}99925$. Para matrices en el rango 0–20, $\alpha = 0{,}9985$ ha arrojado los mejores resultados, mientras que para el rango 80–100 un valor algo inferior ($\alpha \approx 0{,}985$) ha mostrado un rendimiento satisfactorio.

2. **La temperatura inicial (**Temperature**) resulta un parámetro secundario:** Un rango intermedio (entre 3000 y 8000) equilibra la exploración y el coste de ejecución. Ajustar este parámetro a valores extremos (por ejemplo, 1000 o 10000) no perjudica gravemente la solución siempre que α sea suficientemente alto. Sin embargo, fijar una Temp excesivamente alta (por encima de 10000) alarga demasiado la fase inicial, sin grandes beneficios adicionales en la calidad final. De igual forma, partir de 1000 puede acortar la etapa de exploración inicial, aunque con un buen factor de enfriamiento se mitiga ese posible efecto negativo.

3. **La temperatura mínima (**Temp_{min}**) carece de gran relevancia dentro de rangos bajos:** Establecer Temp_{min} en valores entre 0,01 y 0,001 apenas produce diferencias en la calidad final, ya que en ese punto la probabilidad de aceptar soluciones peores es prácticamente nula. Por ello, para la mayoría de casos prácticos, mantener $\text{Temp}_{\text{min}} \leq 0{,}01$ es suficiente.

4. **Los criterios de parada deben ajustarse con cautela:** Incrementar sin límite maxIterations o maxNoImprovement es una estrategia efectiva para reducir el coste de la solución, pero a cambio de alargar significativamente el tiempo de ejecución. Así, conviene hallar un punto de equilibrio entre calidad y velocidad. En este trabajo se han utilizado maxIterations = 10000 y maxNoImprovement = 2500 para garantizar una exploración amplia sin caer en tiempos excesivos.

Valores concretos de configuración recomendada.

Basándonos en los resultados experimentales y considerando un equilibrio razonable entre calidad y tiempo de cómputo, podemos recomendar la siguiente configuración para SA en QAP de tamaño medio:

- $\alpha \approx$ 0,998–0,999 = 0.9985.
- Temperatura inicial (Temp) = 5000, en torno a 3000–8000, evitando valores excesivamente altos.

- Temperatura mínima (Temp_{min}) de 0,01 o incluso menor (0.005–0.001), sin diferencias relevantes en la calidad final.
- Criterios de parada en torno a maxIterations = 10000 y maxNoImprovement = 2500, ajustables según la disponibilidad computacional.

En definitiva, la **prioridad principal es afinar el valor de** α, puesto que su efecto supera al de la temperatura inicial o la mínima a la hora de asegurar una exploración amplia y evitar estancamientos prematuros. Combinando un factor de enfriamiento elevado con límites de iteración razonables, SA demuestra ser una metaheurística fiable y flexible para abordar instancias del QAP de tamaño mediano e, incluso, escalas mayores cuando la optimización exacta resulta inasumible.

Capítulo 2

Juegos de Prueba

2.1. Tests Unitarios

2.1.1. Tests en los Algoritmos Separados

Algoritmos para resolver el TSP

Algoritmo de Fuerza Bruta: El `Algoritmo de Fuerza Bruta` implementado en la clase `driverBruteForceTSP` resuelve instancias del Problema del Viajante (TSP) mediante una exploración exhaustiva de todas las posibles permutaciones de rutas. Para cada conjunto específico de productos (nodos), el algoritmo calcula la distribución lateral mínima utilizando el método `AlgoritmoFactory.obtenerMinDistribucionLateral`. Se han diseñado múltiples casos de prueba con diferentes cantidades de productos (4, 6, 8, 10 y 12) para verificar la correcta ejecución y exactitud del algoritmo. Cada prueba compara el resultado obtenido con un valor mínimo esperado, asegurando así que el algoritmo de fuerza bruta encuentra la solución óptima en cada instancia evaluada.

Algoritmo Held-Karp: El `Algoritmo Held-Karp` implementado en la clase `driverHeldKarpTSP` resuelve instancias del Problema del Viajante (TSP) utilizando el enfoque dinámico de Held-Karp, que es una mejora significativa sobre el método de fuerza bruta. Este algoritmo calcula la distribución lateral mínima de manera exacta para conjuntos de productos (nodos) mediante una optimización que reduce la complejidad computacional en comparación con la fuerza bruta.

Se han diseñado múltiples casos de prueba con diferentes cantidades de productos (4, 6, 8, 10, 12, 15 y 20) para verificar la correcta ejecución y exactitud del algoritmo. Sin embargo, las pruebas se limitan hasta 20 productos debido al alto consumo de memoria y al incremento exponencial en el tiempo de cómputo que conlleva manejar instancias más grandes. A pesar de que el algoritmo Held-Karp es más eficiente que la fuerza bruta y sigue proporcionando soluciones exactas, su demanda de recursos lo hace menos viable para instancias de gran tamaño. Por lo tanto, se ha considerado óptimo hasta 20 productos para mantener un equilibrio entre exactitud y eficiencia computacional.

Cada prueba compara el resultado obtenido con un valor mínimo esperado, asegurando así que el algoritmo de Held-Karp encuentra la solución óptima en cada instancia evaluada dentro de los

límites de tamaño establecidos.

Algoritmo 2-Aproximado basado en el MST: El `Algoritmo 2-Aproximado basado en el MST` implementado en la clase `driverMST2approxTSP` resuelve instancias del Problema del Viajante (TSP) utilizando un enfoque basado en el Árbol de Expansión Mínima (MST). Este método de aproximación es significativamente más eficiente que el método de fuerza bruta, ya que reduce considerablemente el coste computacional al proporcionar una solución que no excede el doble del óptimo.

Se han diseñado múltiples casos de prueba con diferentes cantidades de productos (4, 6, 8, 10, 12, 15, 20, 25 y 50) para verificar la correcta ejecución y exactitud del algoritmo. A diferencia del algoritmo Held-Karp, el algoritmo basado en MST puede manejar instancias de mayor tamaño con un bajo coste computacional, lo que permite su aplicación en problemas con hasta 50 productos sin un incremento significativo en el tiempo de cómputo.

Cada prueba compara el resultado obtenido con un valor mínimo esperado, asegurando así que el algoritmo de aproximación basado en MST encuentra soluciones correctas y eficientes en cada instancia evaluada (no se sabe el coste mínimo para las matrices de 50 productos).

Algoritmo 3/2-Aproximado, Christofides: El **Algoritmo Christofides** implementado en la clase `driverChristofidesTSP` aborda el Problema del Viajante (TSP) proporcionando una solución aproximada con una garantía de $\frac{3}{2}$ veces el coste de la distribución óptima (en determinados grafos).

A diferencia de los métodos exactos como el de Fuerza Bruta o Held-Karp, el algoritmo de Christofides ofrece una solución eficiente en términos de tiempo y memoria, lo que le permite manejar instancias más grandes del problema sin incurrir en un consumo excesivo de recursos computacionales. Sin embargo, para mantener un equilibrio óptimo entre exactitud y eficiencia, las pruebas se limitan hasta 25 productos. A pesar de que el algoritmo es más eficiente que los métodos exactos y sigue siendo una buena solución aproximada dentro de su marco teórico, aumentar el número de productos más allá de este límite puede resultar en un incremento significativo en el tiempo de cómputo y en el uso de memoria, haciéndolo impráctico para conjuntos de datos muy grandes.

Cada prueba realizada compara el resultado obtenido con un valor mínimo esperado, asegurando así que el algoritmo de Christofides proporciona soluciones cercanas al óptimo en cada instancia evaluada. Este enfoque garantiza que, dentro de los límites establecidos, el algoritmo mantiene una alta calidad en las soluciones generadas mientras optimiza el uso de recursos computacionales.

Algoritmos para resolver el QAP

Algoritmo de Fuerza Bruta: El `Algoritmo de Fuerza Bruta` implementado en la clase `driverBruteForceQAP` resuelve instancias del Problema de Asignación Cuadrática (QAP) utilizando un enfoque de fuerza bruta que garantiza la obtención de soluciones exactas. Este algoritmo evalúa todas las posibles permutaciones de asignación para determinar la distribución lateral mínima de manera exhaustiva, lo que asegura la exactitud de los resultados pero con un alto coste computacional.

Se han diseñado múltiples casos de prueba con diferentes cantidades de productos (hasta 10)

para verificar la correcta ejecución y exactitud del algoritmo. Las pruebas incluyen diferentes tipos de estanterías con distintos tamaños en todos los casos, asegurando que el algoritmo puede manejar diversas configuraciones de almacenamiento. Debido al elevado coste computacional, las pruebas se han limitado hasta 10 productos, evitando un incremento exponencial en el tiempo de cómputo que conllevaría manejar instancias de mayor tamaño. Cada prueba compara el resultado obtenido con un valor mínimo esperado, garantizando así que el algoritmo de fuerza bruta encuentra la solución óptima en cada instancia evaluada dentro de los límites de tamaño establecidos.

Algoritmo de Aproximación Hill Climbing (HC): El `Algoritmo de Aproximación Hill Climbing` implementado en la clase `driverHillClimbingQAP` resuelve instancias del Problema de Asignación Cuadrática (QAP) y del Problema del Viajante de Comercio (TSP) utilizando una estrategia heurística que mejora iterativamente una solución inicial mediante movimientos locales. A diferencia del enfoque de fuerza bruta, Hill Climbing no garantiza encontrar la solución óptima global, pero reduce significativamente el tiempo de cómputo, permitiendo manejar instancias de mayor tamaño.

Se han diseñado múltiples casos de prueba con diferentes cantidades de productos (hasta 25) para verificar la correcta ejecución y la calidad de las soluciones encontradas por el algoritmo. Las pruebas incluyen diversas configuraciones de matrices de costos, asegurando que el algoritmo puede adaptarse a distintas estructuras de datos. Cada caso de prueba compara el resultado obtenido con un valor mínimo esperado o soluciones conocidas, garantizando que el algoritmo de Hill Climbing encuentra soluciones eficientes y cercanas al óptimo en todas las instancias evaluadas dentro de los límites de tamaño establecidos.

Algoritmo de Simulated Annealing (SA): El `Algoritmo de Simulated Annealing` implementado en la clase `driverSimulatedAnnealingQAP` resuelve instancias del Problema de Asignación Cuadrática (QAP) y del Problema del Viajante de Comercio (TSP) utilizando una técnica probabilística inspirada en el proceso de recocido térmico. Este método permite escapar de óptimos locales aceptando, con cierta probabilidad, soluciones peores durante la búsqueda, lo que aumenta las posibilidades de encontrar una solución óptima global en comparación con métodos deterministas como Hill Climbing.

Se han diseñado múltiples casos de prueba con diferentes cantidades de productos (hasta 25) para verificar la correcta ejecución y la calidad de las soluciones encontradas por el algoritmo. Las pruebas incluyen diversas configuraciones de matrices de costos, asegurando que el algoritmo puede adaptarse a distintas estructuras de datos. Además, los parámetros del algoritmo, tales como la temperatura inicial, la tasa de enfriamiento y la temperatura mínima, han sido optimizados a través de un estudio estadístico para garantizar un rendimiento óptimo en las diferentes instancias evaluadas. Cada caso de prueba compara el resultado obtenido con un valor mínimo esperado o soluciones conocidas, asegurando que el algoritmo de Simulated Annealing encuentra soluciones eficientes y cercanas al óptimo en todas las instancias evaluadas dentro de los límites de tamaño establecidos.

2.1.2. Tests en la Ejecución de los Algoritmos Simultáneamente

Algoritmos para resolver el TSP

Algoritmos para resolver el QAP

2.1.3. Tests para realizar el Estudio Estadístico

Para optimizar los parámetros del **Algoritmo de Simulated Annealing (SA)**, se ha llevado a cabo un estudio estadístico que evalúa el rendimiento del algoritmo bajo diversas configuraciones de matrices de costos. Este estudio tiene como objetivo identificar los valores óptimos para los parámetros clave del SA, tales como la temperatura inicial, la tasa de enfriamiento y la temperatura mínima, asegurando así que el algoritmo proporcione soluciones cercanas al óptimo global de manera eficiente.

Diseño de los Casos de Prueba

Se han diseñado tres tipos principales de casos de prueba, cada uno utilizando matrices de costos con diferentes características:

1. **Matrices con Valores entre 0 y 100:** Estas matrices contienen valores generados de manera aleatoria, simulando escenarios sin patrones específicos. Este tipo de matrices permite evaluar la robustez del algoritmo en situaciones impredecibles.

2. **Matrices con Valores entre 0 y 20:** En este caso, las matrices presentan una distribución de costos bajos, lo que permite observar cómo el algoritmo maneja asignaciones con costos mínimos. Este tipo de matrices es útil para analizar la capacidad del SA para encontrar soluciones eficientes cuando los costos son relativamente bajos.

3. **Matrices con Valores entre 80 y 100:** Estas matrices contienen valores altos, lo que ayuda a evaluar la eficacia del algoritmo en situaciones donde los costos son elevados. Este escenario es relevante para casos donde las asignaciones con altos costos son más comunes.

Para cada tipo de matriz, se han ejecutado tres casos de prueba distintos, totalizando nueve casos de prueba. Esta diversidad permite analizar cómo los diferentes parámetros del SA afectan la calidad de las soluciones obtenidas en distintos entornos de costos.

Optimización de Parámetros

Los parámetros del SA, incluyendo la temperatura inicial, la tasa de enfriamiento y la temperatura mínima, han sido ajustados mediante un enfoque estadístico. Se han realizado múltiples ejecuciones del algoritmo con diferentes combinaciones de parámetros para determinar cuáles proporcionan las mejores aproximaciones en cada tipo de matriz de costos. El análisis estadístico ha permitido identificar las configuraciones que maximizan la probabilidad de encontrar soluciones cercanas al óptimo global mientras se mantiene una eficiencia computacional adecuada.

2.2. Tests de Integración

2.3. Tests para Demostrar el Correcto Funcionamiento de los Algoritmos

En esta sección se describen los distintos escenarios de prueba diseñados para verificar que todos los algoritmos de TSP (4 algoritmos) y QAP (3 algoritmos) funcionan correctamente. Se evalúan situaciones que abarcan distintos tamaños de matrices, con diferentes coeficientes de similitud y diferentes configuraciones de productos (desde casos muy pequeños de 3 o 4 productos, hasta casos medianos de 8 y 10). Además, se contemplan condiciones especiales, como coeficientes de similitud puntuales nulos, muy altos o con restricciones que exigen que ciertos productos estén adyacentes (la restricción wrap-around, exigida para el TSP).

A continuación, se describen en detalle los distintos test que se realizan y el razonamiento detrás de ellos:

2.3.1. Test 1: Tres Ciudades

- **Contexto**:
  - Matriz de coeficiente de similitud de 3×3 que refleja un escenario básico.
  - Se evalúan dos configuraciones de baldas:
    1. 1 fila por 3 columnas.
    2. 3 filas por 1 columna.
- **Objetivo de la prueba**:
  - Verificar que todos los algoritmos de TSP (BRUTEFORCE, HELDKARP, MST2APPROXIMATION y CHRISTOFIDES) encuentran la misma distribución mínima.
  - De igual manera, comprobar que los tres algoritmos de QAP (BRUTEFORCE, HILL-CLIMBING y SIMULATED_ANNEALING) devuelven resultados coherentes y correctos.
  - Según el comentario en el `init()`, la solución mínima esperada es de 100.
- **Conclusión**:
  - Todos los algoritmos deben devolver la misma solución que respalde ese mínimo de 100.
  - Se confirma que, con este caso sencillo, la lógica de cada algoritmo opera correctamente tanto para TSP como para QAP.

2.3.2. Test 2: Cuatro Ciudades, coeficiente de similitud Cero Excepto en un Par

- **Contexto**:

- Matriz de coeficiente de similitud 4×4 donde prácticamente todas las interacciones tienen coste 0, salvo un par de productos que tienen un coste de 1.
- Nuevamente, se evalúan:
  1. 1 fila por 4 columnas.
  2. 4 filas por 1 columna.
  3. 2 filas por 2 columnas.

- **Objetivo de la prueba**:
  - Comprobar que, si la coeficiente de similitud es 0 para todos menos para un par de productos (coste 1), los algoritmos deben encontrar distribuciones que *eviten* colocar esos dos productos adyacentes.
- **Conclusión**:
  - Todos los algoritmos, tanto TSP como QAP, han de ser capaces de encontrar configuraciones que eviten poner juntos los dos productos con coeficiente de similitud 1, obteniendo así un coste de 0.
  - Se demuestra, por ende, que todos los algoritmos manejan correctamente el caso donde casi todas las coeficientes de similitud son nulas y solo existe un par "problemático".

2.3.3. Test 3: Cuatro Ciudades, coeficiente de similitud 100 Excepto un Par con 99

- **Contexto**:
  - Matriz de coeficiente de similitud 4×4 en la que todas las interacciones tienen coste 100, excepto un par de productos que tienen coste 99.
  - Se vuelve a probar en 3 configuraciones de baldas: 1×4, 4×1 y 2×2.
- **Objetivo de la prueba**:
  - Se busca comprobar que la solución mínima (indicado en el comentario como 399) se logra colocando *juntos* los productos con coeficiente de similitud 99.
  - Es decir, aquellos productos con menor coste de interacción deben estar adyacentes para reducir el coste total.
- **Conclusión**:
  - Todos los algoritmos deben priorizar la adyacencia de la pareja con coeficiente de similitud 99, de modo que su "peso" repercuta favorablemente en el coste global.
  - Se confirma así que todas las variantes de TSP y QAP encuentran configuraciones equivalentes o cercanas (en el caso de heurísticas) a la mejor solución.

2.3.4. Test 4: Ocho Ciudades, coeficiente de similitud 100 Excepto Tres Pares con 99

- **Contexto**:
  - Matriz de coeficiente de similitud 8×8 con un coste de 100 en la mayoría de interacciones, salvo tres pares de productos que tienen coste 99.
  - Se realizan pruebas en configuraciones 1×8, 8×1 y 2×4.
- **Objetivo de la prueba**:
  - Verificar que la solución ideal debe agrupar esos tres pares *adyacentes*, dado que cada uno aporta un ligero ahorro respecto a los 100 de coste general.
  - El test confirma que los algoritmos reconocen que el coste mínimo se obtiene cuando esos pares de productos se colocan uno al lado del otro.
- **Conclusión**:
  - Tanto las aproximaciones de TSP como las heurísticas de QAP deben detectar de forma consistente la solución donde dichos pares comparten posiciones contiguas.
  - Se demuestra el correcto funcionamiento en un escenario un poco más grande (ocho productos).

2.3.5. Test 5: Ocho Ciudades, coeficiente de similitud 100 Excepto Tres *Clusters*

- **Contexto**:
  - Matriz 8×8 donde se distinguen tres *clusters*:
    1. Productos {1, 2, 3}.
    2. Productos {4, 5, 6}.
    3. Productos {7, 8}.
  - Las coeficientes de similitud dentro de cada cluster son ligeramente menores (por ejemplo, costes 80) que el resto (costes 100), lo que fomenta que cada cluster permanezca unido.
- **Objetivo de la prueba**:
  - Comprobar que las soluciones de TSP y QAP sitúan a cada cluster "junto", minimizando las distancias/costes internos entre los productos que pertenecen a la misma agrupación.
  - Se vuelven a usar configuraciones 1×8, 8×1 y 2×4.
- **Conclusión**:
  - Todos los algoritmos son capaces de agrupar los productos de cada cluster y, así, alcanzar la configuración más ventajosa.

- Se reafirma que incluso con matrices más complejas y patrones de costes, los algoritmos cumplen su cometido.

2.3.6. Test 6: Ocho Ciudades, Existen Distribuciones Únicas

- **Contexto**:
  - Matriz 8×8 construida de tal manera que *solo* existe una distribución óptima.
  - Se sigue probando en configuraciones 1×8, 8×1 y 2×4.
- **Objetivo de la prueba**:
  - Constatar que, cuando la solución mínima es *única*, cada algoritmo es capaz de encontrar exactamente esa misma disposición de los productos.
- **Conclusión**:
  - Aunque pudiera haber empates en algunos escenarios, aquí se ha diseñado un caso de "máxima restricción", donde no hay ambigüedad: la salida correcta es única.
  - Todos los algoritmos demuestran que pueden converger en dicha solución unívoca.

2.3.7. Test 7: Diez Ciudades, Distribución Única

- **Contexto**:
  - Matriz 10×10 en la que, de modo similar al test anterior, se fuerza a que la distribución óptima sea únicamente una.
  - Se ensayan de nuevo tres configuraciones de baldas: 1×10, 10×1 y 2×5.
- **Objetivo de la prueba**:
  - Probar el rendimiento de todos los algoritmos a una escala un poco mayor (10 productos).
  - Constatar que, si la solución está muy condicionada por las coeficientes de similitud específicas, todos los métodos (exactos y heurísticos) pueden encontrar la misma única distribución mínima.
- **Conclusión**:
  - El correcto funcionamiento se mantiene incluso con un número de productos más elevado.
  - Se cierra así la validación de que la implementación de cada algoritmo (tanto de TSP como de QAP) abarca casos grandes y configuraciones que exigen solucioneś óptimas o muy cercanas a las óptimas.

2.3.8. Conclusión General

Los distintos tests descritos validan la corrección de los algoritmos TSP y QAP implementados:

- **Correctos**, porque todos los métodos son capaces de encontrar o aproximarse a la misma solución mínima (o soluciones equivalentes) en escenarios que varían desde 3 a 10 productos.
- **Robustos**, porque se han probado distintas configuraciones de baldas (filas y columnas), distintas combinaciones de coeficientes de similitud (valores nulos, altos y medios) y la necesidad de agrupar productos estratégicamente.

De esta forma, queda demostrada la validez[1] de todos los algoritmos para resolver tanto problemas TSP (rutas mínimas que pasan por todas las "ciudades") como QAP (distribución óptima de productos en baldas), confirmando que cada uno de ellos funciona según lo esperado y alineado con los resultados teóricos deseados.

[1] Almenos, así hemos quedado que demostraríamos la corrección de los algoritmos del QAP, los cuales no tiene dataSets solucionados como el TSP.

Capítulo 3

Programas útiles

3.1. Programas en R

3.1.1. Ranking Mejores Distribuciones del Simulated Annealing por Parámetros (Basado en Coste Medio)

El script de R `Ranking Mejores Distribuciones del Simulated Annealing` está diseñado para identificar y clasificar las mejores configuraciones de parámetros (`Temperature`, `Alpha`, y `MinTemperature`) basándose en el coste medio obtenido al utilizar el algoritmo de Simulated Annealing. Este enfoque es crucial para minimizar la influencia de la aleatoriedad en los resultados, permitiendo así una evaluación más consistente y fiable de las configuraciones de parámetros que generan distribuciones óptimas.

El script realiza las siguientes acciones principales:

1. Carga los datos desde un archivo CSV, omitiendo las líneas que comienzan con "Seed"para evitar información irrelevante.
2. Realiza una inspección preliminar de los datos utilizando funciones como `head` y `summary` para asegurar que los datos se han cargado correctamente.
3. Agrupa los datos por los parámetros `Temperature`, `Alpha`, y `MinTemperature`, y calcula el coste medio (`mean_cost`), el coste mediano (`median_cost`) y el conteo de observaciones (`count`) para cada grupo.
4. Ordena los grupos resultantes en orden ascendente basado en el `mean_cost`, identificando así las configuraciones con los costes medios más bajos.
5. Imprime las diez mejores combinaciones de parámetros que han generado los costes medios más bajos, facilitando la selección de configuraciones óptimas para futuras ejecuciones del algoritmo.

Con este enfoque, el script proporciona una metodología sistemática para evaluar y seleccionar las mejores configuraciones de parámetros, mejorando así la eficiencia y efectividad del algoritmo de Simulated Annealing en la búsqueda de soluciones óptimas.

A continuación, se presenta un breve pseudocódigo que describe la lógica del programa:

```
Inicio
    Cargar librerías necesarias (tidyverse)

    Definir file_path como la ruta al archivo CSV

    Leer el archivo CSV en data_raw, omitiendo líneas que comienzan con "Seed"

    Mostrar las primeras filas de data_raw
    Mostrar resumen estadístico de data_raw

    Agrupar data_raw por Temperature, Alpha, y MinTemperature
        Calcular mean_cost como el coste medio en cada grupo
        Calcular median_cost como el coste mediano en cada grupo
        Calcular count como el número de observaciones en cada grupo
        Desagrupar los datos agrupados

    Ordenar df_summary_sorted por mean_cost de forma ascendente

    Imprimir las primeras 10 filas de df_summary_sorted
Fin
```

Listing 3.1: Pseudocódigo del Ranking de Mejores Distribuciones Basado en Coste Medio

3.1.2. Ranking Mejores Distribuciones del Simulated Annealing por Parámetros (Basado en Coste Único)

El script de R `Ranking Mejores Distribuciones del Simulated Annealing` está diseñado para identificar y clasificar las mejores configuraciones de parámetros (`Temperature`, `Alpha`, y `MinTemperature`) que resultan en el coste mínimo al utilizar el algoritmo de Simulated Annealing. Este análisis es crucial para optimizar el rendimiento del algoritmo, permitiendo seleccionar las combinaciones de parámetros que generan las distribuciones más eficientes y con menores costes.

El script realiza las siguientes acciones principales:

1. Carga los datos desde un archivo CSV, omitiendo las líneas que comienzan con "Seed"para evitar información irrelevante.

2. Agrupa los datos por los parámetros `Temperature`, `Alpha`, y `MinTemperature`, y calcula el coste mínimo (`min_cost`) para cada grupo.

3. Ordena los grupos resultantes en orden ascendente basado en el `min_cost`, identificando así las configuraciones con los costes más bajos.

4. Imprime las cinco mejores combinaciones de parámetros que han generado los costes mínimos, facilitando la selección de configuraciones óptimas para futuras ejecuciones del algoritmo.

A continuación, se presenta un breve pseudocódigo que describe la lógica del programa:

```
Inicio
    Cargar librerías necesarias (tidyverse)

    Definir file_path como la ruta al archivo CSV

    Leer el archivo CSV en data_raw, omitiendo líneas que comienzan con "Seed"

    Agrupar data_raw por Temperature, Alpha, y MinTemperature
        Calcular min_cost como el coste mínimo en cada grupo
        Desagrupar los datos agrupados

    Ordenar df_min_cost_sorted por min_cost de forma ascendente

    Imprimir las primeras 5 filas de df_min_cost_sorted
Fin
```

Listing 3.2: Pseudocódigo del Ranking de Mejores Distribuciones

3.1.3. Plot de las Mejores Distribuciones, Comparando Coste con los Parámetros Variables

El script de R `Plot de las Mejores Distribuciones` está diseñado para visualizar cómo varían los costes en función de diferentes parámetros, específicamente `Temperature` y `Alpha`, al utilizar el algoritmo de Simulated Annealing para encontrar la distribución óptima. Este análisis es fundamental para identificar los parámetros que resultan en un coste menor, lo que a su vez ayuda a optimizar el rendimiento del algoritmo en la búsqueda de soluciones eficientes.

El script realiza las siguientes acciones principales:

1. Carga los datos desde un archivo CSV, omitiendo las líneas que comienzan con "Seed"para evitar información irrelevante.

2. Realiza una inspección preliminar de los datos utilizando funciones como `head` y `summary` para asegurar que los datos se han cargado correctamente.

3. Convierte la variable `Alpha` a un factor para facilitar su uso en el análisis estadístico y la visualización.

4. Genera un **boxplot** que compara el coste en función de los parámetros `Temperature` y `Alpha`, permitiendo identificar visualmente las combinaciones de parámetros que resultan en costes menores.

A continuación, se presenta un breve pseudocódigo que describe la lógica del programa:

```
Inicio
    Cargar librerias necesarias (tidyverse)

    Definir file_path como la ruta al archivo CSV
```

```
5
6     Leer el archivo CSV en data_raw, omitiendo líneas que comienzan con "Seed"
7
8     Mostrar las primeras filas de data_raw
9     Mostrar resumen estadistico de data_raw
10
11    Convertir data_raw$Alpha a factor
12
13    Crear un gráfico de caja con ggplot:
14        Eje X: Temperature
15        Eje Y: Cost
16        Relleno: Alpha
17        Tipo de grafico: geom_boxplot con posicion "dodge"
18        Agregar etiquetas:
19            Titulo: "Cost by Temperature and Alpha"
20            Eje X: "Temperature"
21            Eje Y: "Cost"
22            Leyenda: "Alpha"
23        Aplicar tema minimalista
24 Fin
```

Listing 3.3: Pseudocódigo del Plot de las Mejores Distribuciones

3.2. Programas en C++

3.2.1. Generador de Matrices

El programa generador de matrices se encarga de crear una matriz simétrica de coeficientes de similitud con valores aleatorios entre 0 y 100. Este programa es útil para simular coeficientes de similitud en el proyecto, es decir, para crear una gran variedad de juegos de prueba asegurando la aleatoriedad, permitiendo variar el tamaño de la matriz cambiando el valor de `N`.

```
1  Inicio
2      Definir N como el tamaño de la matriz
3      Inicializar una matriz N x N con ceros
4
5      Inicializar generador de números aleatorios
6      Definir distribución uniforme entre 0 y 100
7
8      Para cada i desde 0 hasta N-1 hacer
9          Para cada j desde i+1 hasta N-1 hacer
10             Generar un valor aleatorio entre 0 y 100
11             Asignar valor a m[i][j] y m[j][i] para mantener la simetría
12         Fin Para
13     Fin Para
14
15     Imprimir la matriz en formato de inicialización de arreglo en Java
```

```
Fin
```

Listing 3.4: Pseudocódigo del Generador de Matrices

3.2.2. Generador de Matrices Perfectas

El programa `Generador de Matrices Perfectas` se encarga de crear una matriz simétrica de distancias entre puntos que cumple la desigualdad triangular. Esto es esencial para asegurar que los juegos de prueba sean adecuados para evaluar la eficacia de los algoritmos de aproximación para el Problema del Viajante de Comercio (TSP), específicamente el algoritmo de 2-aproximación basado en el Árbol de Expansión Mínima (MST) y la 3/2-aproximación de Christofides. Al generar puntos aleatorios uniformemente distribuidos dentro de un círculo, garantizamos que la distancia máxima entre cualquier par de puntos no exceda un valor predefinido, manteniendo así la consistencia requerida por estos algoritmos.

Con este código, se genera una matriz de distancias simétrica donde cada distancia entre dos puntos no excede el valor de `MAX_DISTANCE` (100 en este caso). Al distribuir los puntos uniformemente dentro de un círculo de radio 50, garantizamos que la distancia máxima entre cualquier par de puntos sea, como máximo, 100, cumpliendo así la desigualdad triangular.

A continuación, se presenta un breve pseudocódigo que describe la lógica del programa:

```
Inicio
    Definir N como el número de vértices
    Definir MAX_DISTANCE como la distancia máxima permitida
    Calcular RADIUS como MAX_DISTANCE / 2

    Inicializar generador de números aleatorios
    Definir distribución uniforme para ángulos entre 0 y 2
    Definir distribución uniforme para radio entre 0 y 1

    Crear una lista vacía de puntos

    Para cada i desde 0 hasta N-1 hacer
        Generar un punto aleatorio dentro del círculo de radio RADIUS
        Añadir el punto a la lista de puntos
    Fin Para

    Inicializar una matriz N x N con ceros

    Definir distancia_maxima_actual como 0

    Para cada i desde 0 hasta N-1 hacer
        Para cada j desde i+1 hasta N-1 hacer
            Calcular la distancia euclídea entre puntos[i] y puntos[j]
            Si la distancia > MAX_DISTANCE entonces
                Imprimir error y terminar el programa
            Fin Si
```

```
            Si la distancia > distancia_maxima_actual entonces
                Actualizar distancia_maxima_actual
            Fin Si
            Redondear la distancia a entero más cercano
            Asignar la distancia a m[i][j] y m[j][i]
        Fin Para
    Fin Para

    Imprimir la matriz de distancias en formato de arreglo de Java

    Generar e imprimir el arreglo 'productos' correspondiente a los vértices
Fin
```

Listing 3.5: Pseudocódigo del Generador de Matrices Perfectas

Este pseudocódigo resume los pasos principales del `Generador de Matrices Perfectas`, facilitando la comprensión de su funcionamiento y su implementación en otros lenguajes si es necesario. Al asegurar que la matriz de distancias cumple con la desigualdad triangular, este generador es una herramienta valiosa para evaluar la efectividad de los algoritmos de aproximación para el TSP en escenarios controlados y realistas.

Además, al generar las coordenadas de los puntos dentro de un círculo y calcular las distancias euclidianas entre ellos, se garantiza que la matriz resultante no solo sea simétrica, sino que también refleje una distribución espacial realista de los productos, lo que es crucial para análisis posteriores en el proyecto.

3.2.3. Parseador de Output de R a LaTeX (Coste Medio)

El programa `Parseador de Output de R a LaTeX (Coste Medio)` está diseñado para automatizar la conversión de los resultados generados por los scripts de R en tablas LaTeX, facilitando así la inclusión de datos en la documentación del proyecto. Este parser toma como entrada los outputs de los scripts de R que calculan los costes medios de diferentes configuraciones de parámetros en el algoritmo de Simulated Annealing, y los formatea en una tabla LaTeX bien estructurada y fácil de leer.

Este proceso es fundamental para agilizar la productividad, ya que elimina la necesidad de formatear manualmente los resultados, reduciendo errores y ahorrando tiempo durante la preparación de informes y documentación técnica.

El programa realiza las siguientes acciones principales:

1. Lectura de Datos: Lee los datos de entrada desde la salida generada por los scripts de R, ignorando las líneas que no contienen información relevante (como las que comienzan con "Seed").

2. Parseo y Almacenamiento: Analiza cada línea de datos, extrae los valores de `Temperature`, `Alpha`, `MinTemperature`, `mean_cost`, `median_cost`, y `count`, y los almacena en una estructura de datos adecuada.

3. Generación de Código LaTex: Formatea los datos almacenados en una tabla LaTeX, incluyendo encabezados y alineaciones adecuadas para una presentación clara.

4. Salida del Código LaTeX: Imprime el código LaTeX generado, listo para ser incluido en la documentación del proyecto.

Con este enfoque, el programa asegura que los resultados de los análisis estadísticos se presenten de manera consistente y profesional en la documentación, facilitando la interpretación y comparación de las diferentes configuraciones de parámetros utilizadas en el algoritmo de Simulated Annealing.

A continuación, se presenta un breve pseudocódigo que describe la lógica del programa:

```
Inicio
    Cargar librerías necesarias (iostream, string, vector, sstream, iomanip)

    Definir una estructura DataRow para almacenar cada fila de datos:
        Temperature
        Alpha
        MinTemperature
        mean_cost
        median_cost
        count

    Crear un vector de DataRow para almacenar los datos

    Leer y omitir las primeras dos líneas (encabezados)

    Mientras haya líneas para leer hacer
        Leer una línea
        Si la línea está vacía entonces
            Continuar
        Fin Si

        Crear un stream de la línea leída

        Leer y ignorar el número de fila

        Leer Temperature
        Leer Alpha
        Leer MinTemperature
        Leer mean_cost y redondear a entero
        Leer median_cost y redondear a entero
        Leer count

        Almacenar los valores en una instancia de DataRow
        Añadir la instancia al vector de datos
    Fin Mientras

    Iniciar la generación del código LaTeX:
        Imprimir el inicio de la tabla LaTeX
```

```
        Imprimir los encabezados de las columnas

        Para cada fila en el vector de datos hacer
            Imprimir los valores formateados separados por &
            Finalizar la línea con \\\\
        Fin Para

        Imprimir el cierre de la tabla LaTeX
    Fin

    Terminar el programa
Fin
```

Listing 3.6: Pseudocódigo del Parseador de Output de R a LaTeX (Coste Medio)

3.2.4. Parseador de Output de R a LaTeX (Coste Único)

El programa `Parseador de Output de R a LaTeX (Coste Único)` está diseñado para automatizar la conversión de los resultados generados por los scripts de R en tablas LaTeX, facilitando así la inclusión de datos en la documentación del proyecto. Este parser toma como entrada los outputs de los scripts de R que identifican las mejores configuraciones de parámetros basadas en el coste mínimo único en el algoritmo de Simulated Annealing, y los formatea en una tabla LaTeX bien estructurada y fácil de leer.

Este proceso es fundamental para agilizar la productividad, ya que elimina la necesidad de formatear manualmente los resultados, reduciendo errores y ahorrando tiempo durante la preparación de informes y documentación técnica.

El programa realiza las siguientes acciones principales:

1. **Lectura de Datos**: Lee los datos de entrada desde la salida generada por los scripts de R, ignorando las líneas que no contienen información relevante (como las que comienzan con "Seed").

2. **Parseo y Almacenamiento**: Analiza cada línea de datos, extrae los valores de `Temperature`, `Alpha`, `MinTemperature`, y `min_cost`, y los almacena en una estructura de datos adecuada.

3. **Conversión de Formato**: Convierte los valores de tipo `double` a cadenas de texto eliminando ceros innecesarios para una presentación más limpia.

4. **Generación de Código LaTeX**: Formatea los datos almacenados en una tabla LaTeX, incluyendo encabezados y alineaciones adecuadas para una presentación clara.

5. **Salida del Código LaTeX**: Imprime el código LaTeX generado, listo para ser incluido en la documentación del proyecto.

Con este enfoque, el programa asegura que los resultados de los análisis estadísticos se presenten de manera consistente y profesional en la documentación, facilitando la interpretación y comparación de las diferentes configuraciones de parámetros utilizadas en el algoritmo de Simulated Annealing.

A continuación, se presenta un breve pseudocódigo que describe la lógica del programa:

```
Inicio
    Cargar librerías necesarias (iostream, string, sstream, iomanip, vector)

    Definir una estructura DataRow para almacenar cada fila de datos:
        Temperature
        Alpha
        MinTemperature
        min_cost

    Crear un vector de DataRow para almacenar los datos

    Leer y omitir las primeras dos líneas (encabezados)

    Mientras haya líneas para leer hacer
        Leer una línea
        Si la línea está vacía entonces
            Continuar
        Fin Si

        Crear un stream de la línea leída

        Leer y ignorar el número de fila

        Leer Temperature
        Leer Alpha
        Leer MinTemperature
        Leer min_cost

        Almacenar los valores en una instancia de DataRow
        Añadir la instancia al vector de datos
    Fin Mientras

    Iniciar la generación del código LaTeX:
        Imprimir el inicio de la tabla LaTeX
        Imprimir los encabezados de las columnas

        Para cada fila en el vector de datos hacer
            Imprimir Temperature, Alpha, MinTemperature, y min_cost separados por &
            Finalizar la línea con \\\\
        Fin Para

        Imprimir el cierre de la tabla LaTeX
    Fin

    Terminar el programa
Fin
```

Listing 3.7: Pseudocódigo del Parseador de Output de R a LaTeX (Coste Único)

Capítulo 4

Conclusión

En este trabajo fina de la asignatura **Proyectos de Programación**, se ha abordado con éxito el desarrollo de una aplicación en *Java* para la optimización de la distribución de productos en un supermercado, basándose en la idea de maximizar el beneficio al ubicar de forma adyacente aquellos artículos con mayor probabilidad de compra conjunta. El proyecto se ha organizado siguiendo la arquitectura de tres capas (dominio, persistencia y presentación), y se han aplicado los conocimientos adquiridos en asignaturas anteriores en cuanto a la programación orientada a objetos, el uso de patrones de diseño, la elaboración de diagramas UML y la planificación y ejecución de pruebas.

Para resolver el problema, se han implementado y comparado siete algoritmos. En primer lugar, se han desarrollado cuatro algoritmos que modelan la problemática como una adaptación del *Traveling Salesman Problem* (*TSP*), en la que se busca un camino que minimice la distancia (o maximice la afinidad) entre productos y se obtiene la solución en forma de disposición lineal. En segundo lugar, se han propuesto tres algoritmos inspirados en el *Quadratic Assignment Problem* (*QAP*), donde se procura optimizar las adyacencias de los productos (sin considerar diagonales) para minimizar la distancia y, en consecuencia, mejorar la distribución en estanterías. Esto ha permitido comparar distintos enfoques y seleccionar aquel que ofrezca mejores resultados en términos de eficiencia y facilidad de implementación.

A lo largo del desarrollo, se han cumplido los objetivos formativos propuestos:

- Se ha puesto en práctica la selección y aplicación de estructuras de datos y algoritmos adecuados para un proyecto de programación de tamaño mediano, valorando la escalabilidad y eficiencia de cada enfoque.

- Se ha logrado organizar la planificación y reparto de tareas dentro de un grupo de trabajo, lo que ha fomentado la cooperación, el control de versiones (*Git*) y la integración continua (*JUnit*) para la realización de pruebas.

- Se ha llevado a cabo un cuidadoso diseño orientado a objetos, identificando las partes susceptibles de factorizar y aplicando mecanismos como la encapsulación, herencia y polimorfismo para lograr un código mantenible y escalable.

- Se han aplicado principios de usabilidad en el diseño de la interfaz gráfica, asegurando la efectividad y la comodidad de uso.

Asimismo, se han incluido *UMLs*, casos de uso, diagramas de clases y descripciones textuales para reflejar el diseño de la arquitectura. Este proceso ha involucrado la creación de un modelo de dominio claro, la definición de requerimientos funcionales y el desarrollo de prototipos hasta llegar a la solución final. Además, se ha empleado una metodología de pruebas exhaustivas, donde se ha verificado el correcto manejo de errores y excepciones, garantizando la fiabilidad del sistema.

En definitiva, el proyecto ha permitido integrar los conocimientos de programación orientada a objetos, la planificación y organización del trabajo en equipo y la aplicación de distintas metodologías de desarrollo de software. De esta forma, se han cumplido los objetivos académicos y profesionales propuestos, lo que confirma la validez de las decisiones de diseño adoptadas y la efectividad de los algoritmos implementados. Esta experiencia sienta, además, una base sólida para futuros desarrollos y mejoras, contribuyendo así a la formación integral en ingeniería de software.