Optimización del Job Shop Problem Usando Hill Climbing y Simulated Annealing

Silupu Peñaranda, Collin Rodrigo · 202431053

25 de mayo de 2024

Resumen

Este estudio aborda el problema de programación del Job Shop Problem (JSP), un desafío de optimización conocido en el ámbito industrial, que implica la asignación eficiente de recursos compartidos, como máquinas, para completar una serie de trabajos en el menor tiempo de producción posible. La complejidad del JSP radica en su naturaleza combinatoria, que requiere equilibrar múltiples objetivos, como minimizar el makespan y usar eficientemente los recursos. Este documento presenta la aplicación de dos algoritmos de optimización, Hill Climbing y Simulated Annealing, para resolver el JSP. El estudio demuestra que el Simulated Annealing muestra un rendimiento superior al explorar mejor el espacio de soluciones, logrando así un makespan menor.
Palabras claves: *Programación de talleres, Hill Climbing, Simulated Annealing.*

1. Introducción

En un mundo cada vez más competitivo, las empresas buscan constantemente maneras de reducir costos y tiempos de producción para mantenerse a la vanguardia. En este sentido, existe una necesidad crítica de ser eficientes en el uso de recursos, lo que lleva al planteamiento del Job Shop Problem (JSP).

El JSP es un desafío de optimización conocido por su complejidad en el ámbito industrial. Consiste en la asignación eficiente de recursos compartidos, como máquinas, para completar una serie de trabajos competitivos dentro de un tiempo mínimo de producción. La dificultad del JSP radica en su naturaleza combinatoria, donde la solución óptima debe equilibrar múltiples objetivos, como la minimización del tiempo total de finalización (makespan) y el uso eficiente de recursos energéticos y económicos. Diversas técnicas de optimización, como los algoritmos genéticos, la optimización por colonia de hormigas y la búsqueda tabú, han sido aplicadas para abordar este problema, demostrando mejoras significativas en la productividad y eficiencia industrial (Tamssaouet et al., 2021; Mokhtari y Hasani, 2017; Musser et al., 1993; Yusof et al., 2011; Leite, 2023).

2. Descripción del JSP

El JSP consiste en un conjunto $J = \{J_1, J_2, \ldots, J_n\}$ de n trabajos y un conjunto $M = \{M_1, M_2, \ldots, M_m\}$ con m máquinas. Cada trabajo J_i tiene n_i operaciones $O_i = \{O_{i,1}, O_{i,2}, \ldots, O_{i,n_i}\}$. Cada operación $O_{i,j}$ puede ser realizada por una máquina de un conjunto de máquinas factibles $M_{i,j} \subseteq M$, para $1 \leq i \leq n$ y $1 \leq j \leq n_i$. El tiempo de procesamiento de $O_{i,j}$ en M_k se representa por $p_{i,j,k}$ y $o = \sum n_i$ es el número total de operaciones (Escamilla-Serna et al., 2022).

Es necesario llevar a cabo todas las operaciones para completar un trabajo, respetando la precedencia de operaciones. El JSP tiene las siguientes condiciones: (1) Al inicio, todos los trabajos y

todas las máquinas están disponibles. (2) Cada operación solo puede ser realizada por una máquina. (3) Una máquina no puede ser interrumpida mientras procesa una operación. (4) Cada máquina puede realizar una operación a la vez. (5) Una vez definido, el orden de las operaciones no puede ser cambiado. (6) No se consideran fallos de máquinas. (7) Trabajos diferentes no tienen restricciones de precedencia entre ellos. (8) Las máquinas no dependen unas de otras. (9) El tiempo de procesamiento incluye la preparación de las máquinas y la transferencia de operaciones. (Escamilla-Serna et al., 2022).

3. Metodología

Para abordar el JSP, se han desarrollado y aplicado diversos modelos de optimización. En esta sección, se describen dos de estos modelos: Hill Climbing y Simulated Annealing, los cuales han demostrado ser efectivos en la mejora de la productividad y eficiencia industrial.

3.1. Hill Climbing

El Hill Climbing es un algoritmo de búsqueda local que itera buscando mejorar una solución actual moviéndose hacia estados vecinos con un valor más alto de una función objetivo. En el contexto del JSP, la función objetivo generalmente es minimizar el tiempo total requerido para completar todas las tareas (makespan). La idea principal detrás de Hill Climbing es siempre moverse hacia una solución vecina que mejore la solución actual. Esto se logra intercambiando tareas o reordenándolas en la secuencia de trabajo para encontrar una configuración que reduzca el makespan (OpenAI, 2024).

La función objetivo f sería:

$$f : S \rightarrow \mathbb{R}$$

donde S es el espacio de todas las soluciones posibles y f(s) mide el makespan de la solución s.
El vecindario se define con N(s):

$$N(s) = \{s' \in S \mid s' \text{ es vecino de } s\}$$

donde un vecino s' se obtiene intercambiando tareas en la secuencia. Asimismo, el algoritmo se puede plantear de la siguiente manera:

1. Inicializar con una solución s.
2. Mientras no se alcance un criterio de parada:
   - Seleccionar $s' \in N(s)$ tal que $f(s') < f(s)$ (minimización).
   - Si no existe tal s', detenerse.
   - Si $f(s') < f(s)$, entonces $s \leftarrow s'$.

3.2. Simulated Annealing

El Simulated Annealing es un algoritmo de optimización probabilística inspirado en el proceso de enfriamiento de metales. A diferencia de Hill Climbing, este método permite movimientos hacia soluciones peores con una probabilidad decreciente a medida que avanza el tiempo. Esto es particularmente útil en el JSP, ya que el problema a menudo contiene múltiples óptimos locales. El algoritmo explora el espacio de soluciones permitiendo, ocasionalmente, configuraciones que empeoran el makespan actual para escapar de estos óptimos locales y potencialmente encontrar una mejor solución global (OpenAI, 2024).

La función objetivo f sería:

$$f : S \rightarrow \mathbb{R}$$

donde f(s) mide el makespan de la solución s. El vecindario N(s) se define como:

$$N(s) = \{s' \in S \mid s' \text{ es vecino de } s\}$$

La temperatura T se define como:

$$T : \mathbb{N} \rightarrow \mathbb{R}^+$$

La probabilidad de aceptación sería:

$$P(\Delta E, T) = \exp\left(\frac{\Delta E}{T}\right)$$

donde $\Delta E = f(s') - f(s)$. Asimismo, el algoritmo se puede plantear de la siguiente manera:

1. Inicializar con una solución s y una temperatura T.
2. Mientras no se alcance un criterio de parada:
   - Generar una solución vecina $s' \in N(s)$.
   - Calcular $\Delta E = f(s') - f(s)$.
   - Si $\Delta E \leq 0$, aceptar s' como la nueva solución.
   - Si $\Delta E > 0$, aceptar s' con una probabilidad $P(\Delta E, T)$.
   - Reducir la temperatura T.

4. Caso 1

En este apartado se presenta un caso en específico. Se plantea la tarea de organizar y secuenciar 10 trabajos distintos, cada uno compuesto por una serie de operaciones que deben ser realizadas en 4 máquinas diferentes. Cada trabajo debe pasar por todas las máquinas, y cada máquina realiza una operación específica en un tiempo determinado. Las operaciones están detalladas en una matriz donde cada elemento indica en qué máquina se realizará la operación y cuánto tiempo tomará.

Tabla 1: Matriz de trabajos y sus respectivas operaciones en cada máquina

Trabajo	(Máquina 0, Tiempo)	(Máquina 1, Tiempo)	(Máquina 2, Tiempo)	(Máquina 3, Tiempo)
0	(0, 3)	(1, 2)	(2, 2)	(3, 1)
1	(0, 2)	(2, 1)	(1, 4)	(3, 3)
2	(1, 4)	(2, 3)	(0, 2)	(3, 5)
3	(2, 2)	(3, 1)	(0, 4)	(1, 3)
4	(0, 3)	(2, 4)	(3, 2)	(1, 1)
5	(1, 2)	(0, 3)	(3, 4)	(2, 1)
6	(2, 3)	(3, 2)	(0, 1)	(1, 4)
7	(0, 4)	(1, 3)	(2, 2)	(3, 1)
8	(1, 2)	(0, 1)	(3, 4)	(2, 3)
9	(2, 1)	(3, 4)	(0, 3)	(1, 2)

Por ejemplo, el primer trabajo tiene cuatro operaciones que se realizan en las siguientes máquinas y tiempos:

- Máquina 0: 3 unidades de tiempo
- Máquina 1: 2 unidades de tiempo
- Máquina 2: 2 unidades de tiempo
- Máquina 3: 1 unidad de tiempo

El objetivo principal de este problema es minimizar el tiempo total necesario para completar todos los trabajos, conocido como makespan. Esto implica encontrar una secuencia óptima en la que los trabajos deben ser procesados en las diferentes máquinas para que el último trabajo se complete en el menor tiempo posible. Para lograr esto, se debe considerar las restricciones de orden de las operaciones dentro de cada trabajo, así como los conflictos que surgen cuando varios trabajos requieren la misma máquina al mismo tiempo. La meta es desarrollar un cronograma eficiente que minimice los tiempos de espera y maximice la utilización de las máquinas, reduciendo así el makespan. Tal y como se mencionó en el apartado anterior, para abordar este problema se aplicarán las técnicas Hill Climbing y Simulated Annealing.

4.1. Hill Climbing

```
>>> print("Solución Hill Climbing :", hill_climbing_solution)
Solución Hill Climbing : [(7, [(0, 4), (1, 3), (2, 2), (3, 1)]), (1, [(0, 2), (2, 1), (1, 4), (3, 3)]), (9, [(2, 1), (3, 4), (0, 3), (1, 2)]), (4, [(0, 3), (2, 4), (3, 2), (1, 1)]), (5
, [(1, 2), (0, 3), (3, 4), (2, 1)]), (6, [(2, 3), (3, 2), (0, 1), (1, 4)]), (2, [(1, 4), (2, 3), (0, 2), (3, 5)]), (8, [(1, 2), (0, 1), (3, 4), (2, 3)]), (3, [(2, 2), (3, 1), (0, 4), (
1, 3)]), (0, [(0, 3), (1, 2), (2, 2), (3, 1)])]
>>> print("Hill Climbing Makespan:", hill_climbing_value)
Hill Climbing Makespan: 90
>>> print("Hill Climbing Tiempo de Ejecución:", hill_climbing_duration, "segundos")
Hill Climbing Tiempo de Ejecución: 0.030543565750122070 segundos
```

Figura 1: Caso 1, Corrida del código usando Hill Climbing

La solución obtenida mediante el algoritmo de Hill Climbing presenta una secuencia específica de trabajos. Esta secuencia comienza con el trabajo 7 y concluye con el trabajo 0, logrando un makespan de 90 unidades de tiempo. En términos de tiempo de ejecución, el algoritmo de Hill Climbing se ejecutó en 0.03 segundos, aproximadamente.

La Figura 2 presenta un diagrama de Gantt que ilustra el algoritmo de Hill Climbing. En este diagrama, cada trabajo está representado por un bloque de color distinto, mostrando sus operaciones secuenciales en diversas máquinas. El orden de los trabajos en el gráfico sigue la secuencia determinada por el algoritmo de Hill Climbing, comenzando con el trabajo 7 y terminando con el trabajo 0. Esta representación visual muestra cómo se distribuyen las operaciones a lo largo del tiempo, resultando en un makespan de 90 unidades de tiempo.
La utilización de cada máquina es variada, con algunos periodos de inactividad presentes. A pesar de esto, la estrategia de Hill Climbing busca optimizar el makespan mediante mejoras locales y deterministas.

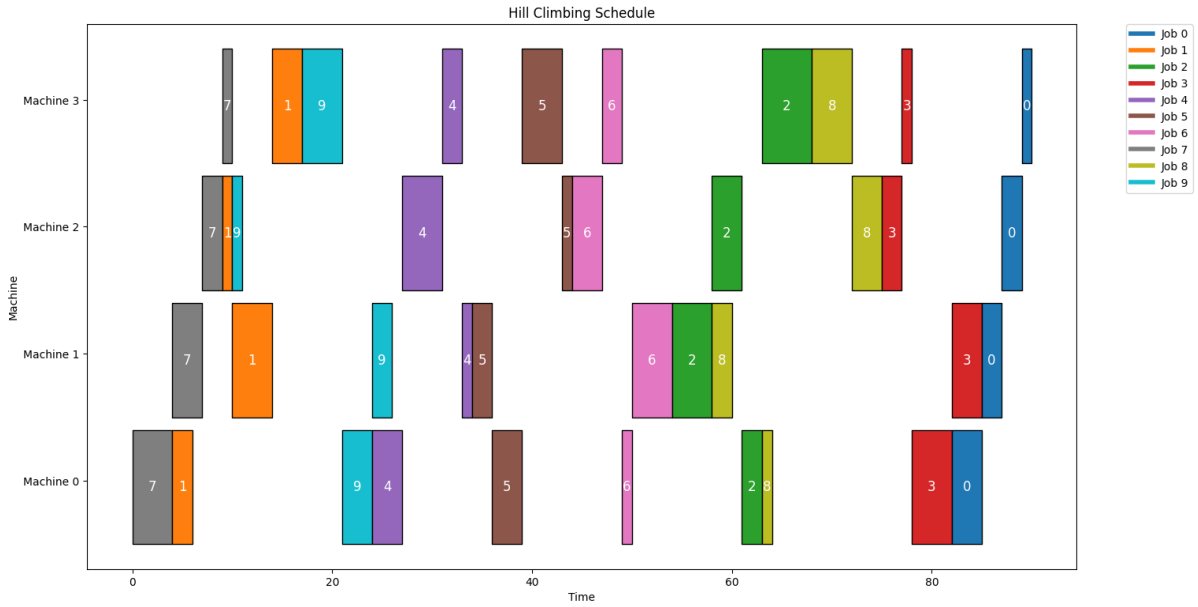

Figura 2: Caso 1, Gráfico de Gantt para la Solución Obtenida con Hill Climbing

4.2. Simulated Annealing

```
>>> end_time = time.time()
>>> sa_duration = end_time - start_time
>>> print("Solución Simulated Annealing:", sa_solution)
Solución Simulated Annealing: [(5, [(1, 2), (0, 3), (3, 4), (2, 1)]), (4, [(0, 3), (2, 4), (3, 2), (1, 1)]), (2, [(1, 4), (2, 3), (0, 2), (3, 5)]), (3, [(2, 2), (3, 1), (0, 4), (1, 3)]
), (7, [(0, 4), (1, 3), (2, 2), (3, 1)]), (6, [(2, 3), (3, 2), (0, 1), (1, 4)]), (8, [(1, 2), (0, 1), (3, 4), (2, 3)]), (1, [(0, 2), (2, 1), (1, 4), (3, 3)]), (9, [(2, 1), (3, 4), (0,
3), (1, 2)]), (0, [(0, 3), (1, 2), (2, 2), (3, 1)])]
>>> print("Simulated Annealing Makespan:", sa_value)
Simulated Annealing Makespan: 88
>>> print("Simulated Annealing Tiempo de Ejecución:", sa_duration, "segundos")
Simulated Annealing Tiempo de Ejecución: 0.016954660415649414 segundos
>>> job_shop.visualize_schedule(sa_solution, "Simulated Annealing Schedule")
```

Figura 3: Caso 1, Corrida del código usando Simulated Annealing

La solución obtenida mediante el algoritmo de Simulated Annealing presenta una secuencia específica de trabajos. Esta secuencia comienza con el trabajo 5 y concluye con el trabajo 0, logrando un makespan de 88 unidades de tiempo. En términos de tiempo de ejecución, el algoritmo de Simulated Annealing se ejecutó en aproximadamente 0.0197 segundos.

La Figura 4 presenta un diagrama de Gantt que ilustra el algoritmo de Simulated Annealing. En este diagrama, cada trabajo está representado por un bloque de color distinto, mostrando sus operaciones secuenciales en diversas máquinas. El orden de los trabajos en el gráfico sigue la secuencia determinada por el algoritmo de Simulated Annealing, comenzando con el trabajo 5 y terminando con el trabajo 0. Esta representación visual muestra cómo se distribuyen las operaciones a lo largo del tiempo, resultando en un makespan de 88 unidades de tiempo.
La utilización de cada máquina es variada, con algunos periodos de inactividad presentes. A pesar de esto, la estrategia de Simulated Annealing permite aceptar temporalmente soluciones subóptimas para escapar de óptimos locales, lo que resulta en una mejor exploración del espacio de soluciones y una distribución más equilibrada de las tareas.

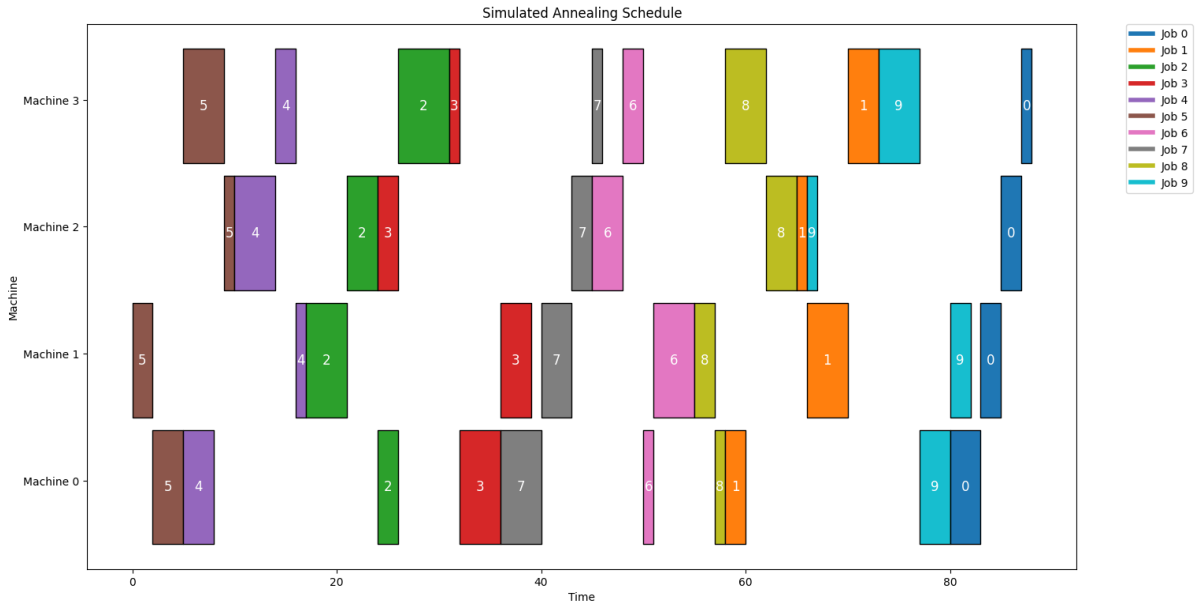

Figura 4: Caso 1, Gráfico de Gantt para la Solución Obtenida con Simulated Annealing

4.3. Comparación de los Modelos

- El algoritmo de Hill Climbing obtiene un makespan de 90 unidades de tiempo, mientras que el Simulated Annealing logra un makespan de 88 unidades de tiempo. Esto indica que, en este caso específico, Simulated Annealing encuentra una solución ligeramente más eficiente en términos de makespan.
- Hill Climbing se ejecuta en aproximadamente 0.03 segundos, destacándose por su rapidez debido a su naturaleza determinista; sin embargo, el modelo Simulated Annealing se ejecuta en aproximadamente 0.01 segundos, siendo también rápido, pero ligeramente más eficiente en términos de tiempo de ejecución en este caso particular.

5. Caso 2

Para el segundo caso, se generará una matriz aleatoria de 60x60, en donde se plantean 60 trabajos distintos y 60 máquinas diferentes. Al igual que el Caso 1, se busca minimizar el makespan.

5.1. Hill Climbing

```
>>> hc_solution, hc_makespan = job_shop.hill_climbing()
>>> hc_duration = time.time() - start_time
>>> print(f"Hill Climbing Solution: {hc_solution[:5]}...")
Hill Climbing Solution: [(42, [(0, 10), (1, 4), (2, 8), (3, 3), (4, 8), (5, 5), (6, 3), (7, 3), (8, 10), (9, 5), (10, 9), (11, 10), (12, 6), (13, 2), (14, 7), (15, 8), (16, 1), (17, 7)
, (18, 6), (19, 6), (20, 6), (21, 7), (22, 2), (23, 10), (24, 1), (25, 8), (26, 6), (27, 4), (28, 8), (29, 8), (30, 2), (31, 8), (32, 1), (33, 2), (34, 3), (35, 6), (36, 7), (37, 3), (
38, 1), (39, 1), (40, 9), (41, 8), (42, 2), (43, 5), (44, 10), (45, 4), (46, 6), (47, 3), (48, 6), (49, 9), (50, 8), (51, 2), (52, 2), (53, 3), (54, 2), (55, 4), (56, 4), (57, 6), (58,
 8), (59, 6)]), (17, [(0, 8), (1, 10), (2, 8), (3, 9), (4, 5), (5, 4), (6, 2), (7, 3), (8, 8), (9, 4), (10, 2), (11, 6), (12, 9), (13, 8), (14, 6), (15, 9), (16, 6), (17, 3), (18, 7),
(19, 10), (20, 2), (21, 9), (22, 9), (23, 8), (24, 3), (25, 9), (26, 6), (27, 7), (28, 5), (29, 8), (30, 10), (31, 8), (32, 8), (33, 4), (34, 6), (35, 1), (36, 7), (37, 5), (38, 1), (3
9, 2), (40, 2), (41, 10), (42, 2), (43, 3), (44, 3), (45, 7), (46, 9), (47, 2), (48, 10), (49, 4), (50, 8), (51, 9), (52, 4), (53, 7), (54, 2), (55, 6), (56, 6), (57, 3), (58, 4), (59,
 8)]), (16, [(0, 6), (1, 1), (2, 5), (3, 10), (4, 9), (5, 8), (6, 8), (7, 3), (8, 9), (9, 7), (10, 6), (11, 5), (12, 8), (13, 3), (14, 9), (15, 2), (16, 9), (17, 2), (18, 10), (19, 2),
 (20, 7), (21, 7), (22, 10), (23, 7), (24, 5), (25, 9), (26, 2), (27, 4), (28, 8), (29, 4), (30, 5), (31, 2), (32, 7), (33, 6), (34, 8), (35, 6), (36, 9), (37, 3), (38, 4), (39, 2), (4
0, 5), (41, 3), (42, 3), (43, 3), (44, 4), (45, 6), (46, 3), (47, 1), (48, 3), (49, 8), (50, 3), (51, 7), (52, 6), (53, 7), (54, 3), (55, 1), (56, 2), (57, 2), (58, 7), (59, 8)]), (8,
[(0, 9), (1, 4), (2, 8), (3, 8), (4, 10), (5, 4), (6, 7), (7, 7), (8, 10), (9, 6), (10, 7), (11, 10), (12, 4), (13, 10), (14, 5), (15, 7), (16, 8), (17, 8), (18, 2), (19, 9), (20, 1),
(21, 5), (22, 9), (23, 8), (24, 1), (25, 1), (26, 9), (27, 3), (28, 10), (29, 1), (30, 10), (31, 6), (32, 1), (33, 9), (34, 8), (35, 8), (36, 4), (37, 10), (38, 8), (39, 1), (40, 5), (
41, 1), (42, 4), (43, 4), (44, 9), (45, 6), (46, 6), (47, 10), (48, 3), (49, 9), (50, 9), (51, 5), (52, 7), (53, 1), (54, 4), (55, 3), (56, 3), (57, 4), (58, 2), (59, 3)]), (33, [(0, 6
), (1, 5), (2, 7), (3, 9), (4, 10), (5, 9), (6, 3), (7, 3), (8, 9), (9, 3), (10, 7), (11, 7), (12, 2), (13, 7), (14, 7), (15, 10), (16, 5), (17, 3), (18, 4), (19, 4), (20, 9), (21, 2),
 (22, 6), (23, 10), (24, 4), (25, 8), (26, 6), (27, 4), (28, 7), (29, 10), (30, 4), (31, 9), (32, 7), (33, 7), (34, 3), (35, 3), (36, 10), (37, 7), (38, 10), (39, 5), (40, 8), (41, 5),
 (42, 10), (43, 3), (44, 8), (45, 7), (46, 10), (47, 2), (48, 5), (49, 5), (50, 9), (51, 1), (52, 4), (53, 1), (54, 9), (55, 8), (56, 4), (57, 2), (58, 2), (59, 2)])]...
>>> print(f"Hill Climbing Makespan: {hc_makespan}")
Hill Climbing Makespan: 1080
>>> print(f"Hill Climbing Duration: {hc_duration} seconds")
Hill Climbing Duration: 41.7567834854126 seconds
```

Figura 5: Caso 2, Corrida del código usando Hill Climbing

Como se observa en la Figura 5, el makespan usando el modelo Hill Climbing es de 1080, con un tiempo de ejecución de 41.8 segundos, aproximadamente.

5.2. Simulated Annealing

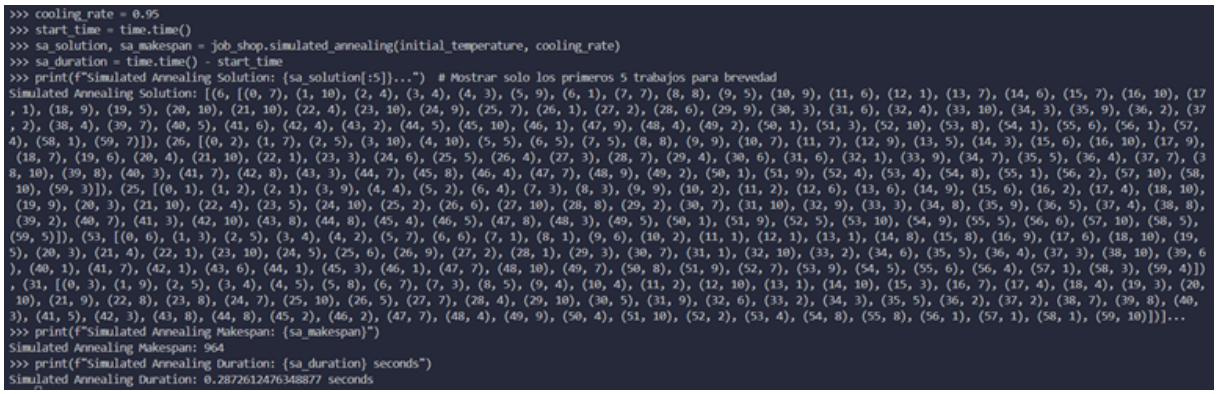

Figura 6: Caso 2, Corrida del código usando Simulated Annealing

Como se observa en la Figura 6, el makespan usando el modelo Simulated Annealing es de 964, con un tiempo de ejecución de 0.29 segundos, aproximadamente.

5.3. Comparación de los Modelos

- El modelo de Simulated Annealing logró un makespan de 964, que es considerablemente menor que el makespan de 1080 obtenido con Hill Climbing. Esto indica que Simulated Annealing encontró una secuencia de trabajos más eficiente, reduciendo el tiempo total necesario para completar todos los trabajos.
- En términos de tiempo de ejecución, Simulated Annealing fue más rápido, con un tiempo de ejecución de aproximadamente 0.29 segundos, en comparación con los 41.8 segundos de Hill Climbing.

6. Caso 3

Para el tercer caso, se generará una matriz aleatoria de 100x100, en donde se plantean 100 trabajos distintos y 100 máquinas diferentes. Al igual que el Caso 1 y 2, se busca minimizar el makespan.

6.1. Hill Climbing

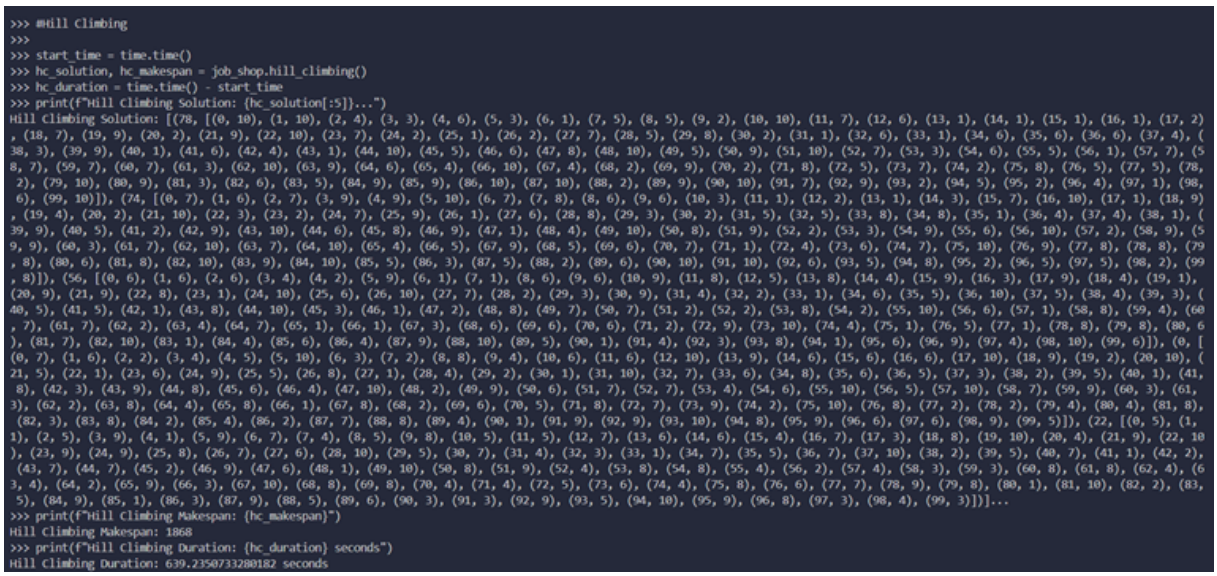

Figura 7: Caso 3, Corrida del código usando Hill Climbing

Como se observa en la Figura 7, el makespan usando el modelo Hill Climbing es de 1868 unidades de tiempo, con un tiempo de ejecución de 639.24 segundos, aproximadamente.

6.2. Simulated Annealing

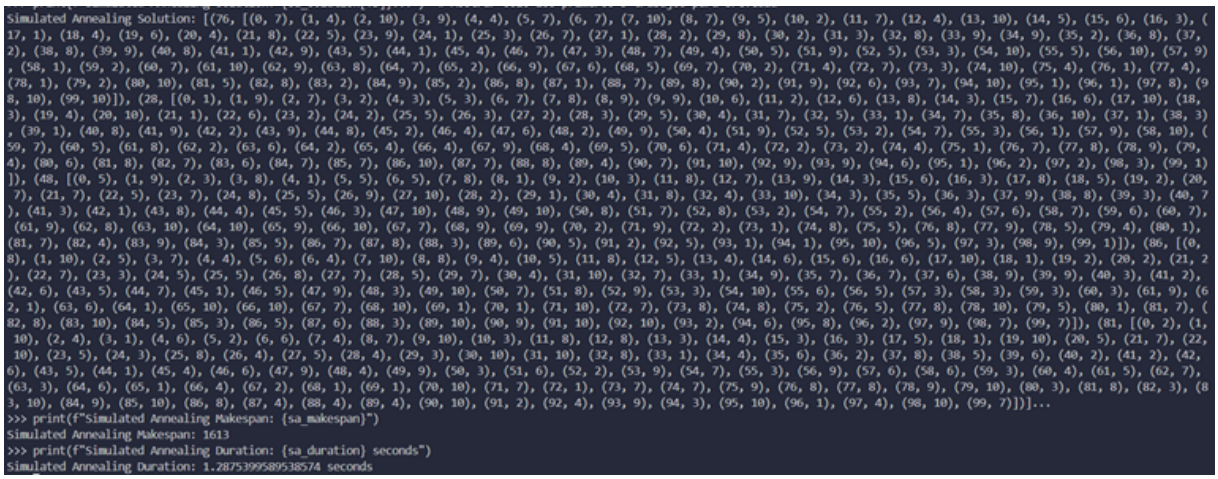

Figura 8: Caso 3, Corrida del código usando Simulated Annealing

Como se observa en la Figura 8, el makespan usando el modelo Simulated Annealing es de 1613, con un tiempo de ejecución de 1.29 segundos, aproximadamente.

6.3. Comparación de los Modelos

- El modelo de Simulated Annealing logró un makespan de 1613, que es considerablemente menor que el makespan de 1868 obtenido con Hill Climbing. Esto indica que Simulated Annealing encontró una secuencia de trabajos más eficiente, reduciendo el tiempo total necesario para completar todos los trabajos.

- En términos de tiempo de ejecución, Simulated Annealing fue extremadamente más rápido, con un tiempo de ejecución de aproximadamente 1.29 segundos, en comparación con los 639.24 segundos de Hill Climbing.

7. Conclusiones

En este estudio se evaluaron dos algoritmos de optimización, Hill Climbing y Simulated Annealing, aplicados a problemas de Job Shop Scheduling con diferentes escalas de trabajo. A través de la implementación y análisis de estos algoritmos en tres casos específicos, se lograron las siguientes conclusiones:

- **Desempeño de Simulated Annealing vs Hill Climbing:** En los tres casos, Simulated Annealing demostró ser superior a Hill Climbing en términos de makespan y tiempo de ejecución.
- **Eficiencia Computacional:** Simulated Annealing no solo logró un menor makespan en todos los casos, sino que también fue más eficiente en términos de tiempo de ejecución. En el Caso 1, el tiempo de ejecución fue de 0.0197 segundos frente a 0.03 segundos de Hill Climbing. En el Caso 2, Simulated Annealing se ejecutó en 0.29 segundos, mientras que Hill Climbing tomó 41.8 segundos. Finalmente, en el Caso 3, Simulated Annealing tuvo un tiempo de ejecución de 1.29 segundos comparado con los 639.24 segundos de Hill Climbing. Esto resalta la eficiencia computacional de Simulated Annealing, especialmente en problemas de mayor escala.
- **Escalabilidad:** Los resultados del Caso 2 y Caso 3, que involucraron matrices de 60x60 y 100x100 respectivamente, evidencian que Simulated Annealing maneja mejor los problemas de gran escala, manteniendo un rendimiento óptimo tanto en makespan como en tiempo de ejecución. Hill Climbing, por otro lado, presentó mayores dificultades y tiempos de ejecución significativamente más altos a medida que aumentaba la escala del problema.
- **Aplicabilidad en Problemas Reales:** Dado que el objetivo principal en problemas de Job Shop Scheduling es minimizar el makespan, Simulated Annealing se presenta como una herramienta más robusta y confiable para abordar problemas similares en entornos industriales y de manufactura. La capacidad de obtener soluciones eficientes en tiempos de ejecución cortos es crucial en aplicaciones prácticas donde los recursos y el tiempo son limitados.

Referencias

Escamilla-Serna, N., Seck-Tuoh-Mora, J., Medina-Marin, J., Barragan-Vite, I., y Corona-Armenta, J. (2022). A hybrid search using genetic algorithms and random-restart hill-climbing for flexible job shop scheduling instances with high flexibility. *Applied Sciences*, *12*, 8050. Descargado de `https://doi.org/10.3390/app12168050` doi: 10.3390/app12168050

Leite, B. S. C. (2023, marzo). El problema de la programación del taller: Modelos de programación entera mixta. *Medium*.

Mokhtari, H., y Hasani, A. (2017). An energy-efficient multi-objective optimization for flexible job-shop scheduling problem. *Computers and Chemical Engineering*, *104*, 339-352. doi: 10.1016/j.compchemeng.2017.05.004

Musser, K., Dhingra, J., y Blankenship, G. (1993). Optimization based job shop scheduling. *IEEE Transactions on Automatic Control*, *38*, 808-813. doi: 10.1109/9.277252

OpenAI. (2024). *ChatGPT: Optimizing Language Models for Dialogue.* (Un modelo de lenguaje conversacional desarrollado por OpenAI, basado en la arquitectura de transformadores, utilizado para generar respuestas de texto en un formato de diálogo.)

Tamssaouet, K., Dauzére-Pérés, S., Knopp, S., Bitar, A., y Yugma, C. (2021). Multiobjective optimization for complex flexible job-shop scheduling problems. *European Journal of Operational Research*, *296*, 87-100. doi: 10.1016/J.EJOR.2021.03.069

Yusof, R., Khalid, M., Hui, G. T., Yusof, S. M., y Othman, M. (2011). Solving job shop scheduling problem using a hybrid parallel micro genetic algorithm. *Applied Soft Computing*, *11*, 5782-5792. doi: 10.1016/j.asoc.2011.01.046