Optimización del Job Shop Problem Usando Hill Climbing y Simulated Annealing

Silupu Peñaranda, Collin Rodrigo · 202431053

19 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. Especificación del Problema

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.

5. Resultados

En este apartado se presentan y analizan los resultados obtenidos al ejecutar los algoritmos de programación implementados para resolver el problema de Job Shop Scheduling. Los métodos empleados incluyen Hill Climbing y Recocido Simulado, cuyas soluciones se evaluarán en términos de makespan y se visualizarán mediante gráficos de Gantt para una mejor interpretación de la asignación de tareas a las máquinas.

5.1. Hill Climbing

```
>>> # Hill Climbing
>>>
>>> hill_climbing_solution, hill_climbing_value = job_shop.hill_climbing()
>>> print("Hill Climbing Solution:", hill_climbing_solution)
Hill Climbing Solution: [(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
```

La solución obtenida utilizando el algoritmo de Hill Climbing presenta una secuencia de trabajos organizada en función del tiempo total requerido por cada uno, comenzando con el trabajo 7 y terminando con el trabajo 0. El makespan obtenido es de 90 unidades de tiempo, lo que indica el tiempo total necesario para completar todos los trabajos en la secuencia determinada.

La Figura 1 presenta el gráfico de Gantt. Se observa que cada trabajo está representado por un bloque de color único en el gráfico, mostrando sus operaciones secuenciales en diferentes máquinas. La secuencia de trabajos en el gráfico sigue el orden determinado por el algoritmo de Hill Climbing, empezando con el trabajo 7 y terminando con el trabajo 0. Esto indica que el trabajo 7 tiene la mayor prioridad, seguido por el trabajo 1, y así sucesivamente. En relación a la utilización de cada máquina, se observa que las máquinas están ocupadas durante casi todo el tiempo, con mínimos periodos de inactividad, lo que indica una alta eficiencia en la programación de tareas. Sin embargo, a pesar de la alta utilización de las máquinas, la solución presenta un makespan relativamente alto de 90 unidades de tiempo, lo que sugiere que la secuencia de trabajos podría no ser óptima.

Los tiempos de espera entre operaciones dentro de un mismo trabajo pueden estar contribuyendo significativamente a este makespan elevado. Además, el algoritmo Hill Climbing puede quedar atrapado en óptimos locales, limitando su capacidad para encontrar la solución más eficiente. Por lo tanto, aunque la solución muestra una distribución eficiente de las operaciones, existen oportunidades para mejorar el makespan explorando secuencias alternativas o utilizando algoritmos más sofisticados que puedan superar las limitaciones del Hill Climbing.

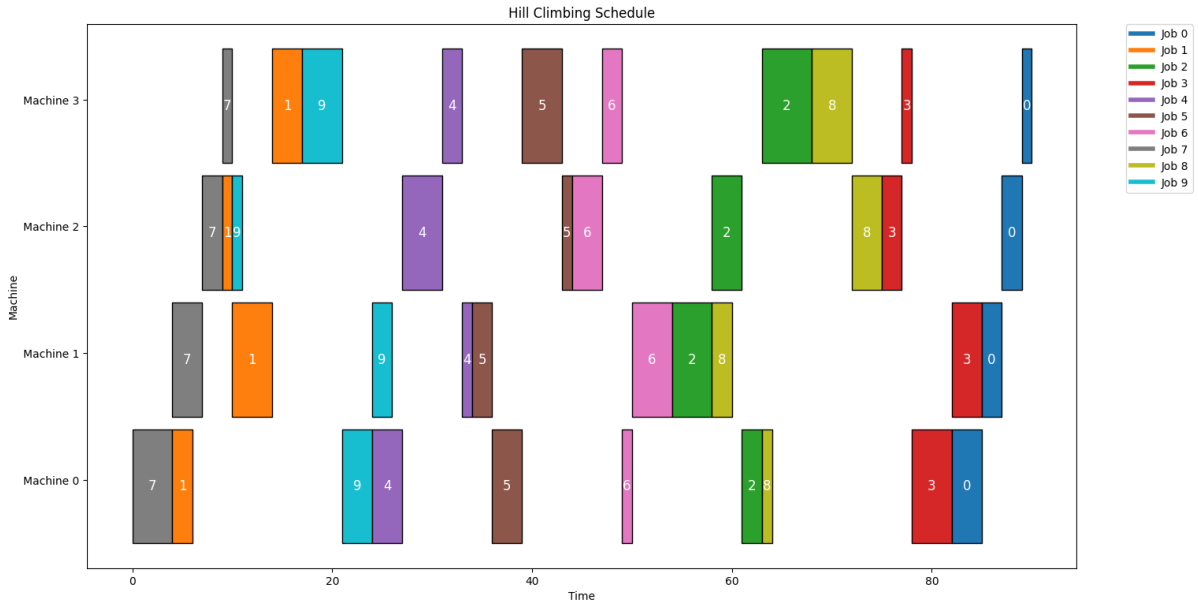

Figura 1: Gráfico de Gantt para la Solución Obtenida con Hill Climbing

5.2. Simulated Annealing

```
>>> print("Simulated Annealing Solution:", sa_solution)
Simulated Annealing Solution: [(7, [(0, 4), (1, 3), (2, 2), (3, 1)]), (3, [(2, 2), (3, 1),
 (0, 4), (1, 3)]), (8, [(1, 2), (0, 1), (3, 4), (2, 3)]), (4, [(0, 3), (2, 4), (3, 2), (1,
 1)]), (2, [(1, 4), (2, 3), (0, 2), (3, 5)]), (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)]), (5, [(1, 2), (0,
3), (3, 4), (2, 1)]), (6, [(2, 3), (3, 2), (0, 1), (1, 4)])]
>>> print("Simulated Annealing Makespan:", sa_value)
Simulated Annealing Makespan: 87
```

La solución obtenida utilizando el algoritmo de Simulated Annealing presenta una secuencia de trabajos diferente, organizada de manera que se exploran más combinaciones posibles debido a la naturaleza estocástica del algoritmo. Esta solución comienza con el trabajo 3 y termina con el trabajo 1. El makespan obtenido es de 87 unidades de tiempo, lo que indica una mejora respecto a la solución obtenida con Hill Climbing.

La Figura 2 presenta el gráfico de Gantt correspondiente. En este gráfico, cada trabajo está representado por un bloque de color único, mostrando sus operaciones secuenciales en diferentes máquinas. La secuencia de trabajos en el gráfico sigue el orden determinado por el algoritmo de Simulated Annealing, comenzando con el trabajo 3 y terminando con el trabajo 1. Esta secuencia permite una mejor distribución de las operaciones, lo que se refleja en el menor makespan. La utilización de cada máquina es alta, con periodos de inactividad mínimos, similar a lo observado con Hill Climbing. Sin embargo, el algoritmo de Simulated Annealing permite aceptar temporalmente

soluciones subóptimas para escapar de óptimos locales, lo que resulta en una mejor exploración del espacio de soluciones. Esto se traduce en una distribución más equilibrada de las tareas y una reducción en los tiempos de espera entre operaciones de un mismo trabajo.

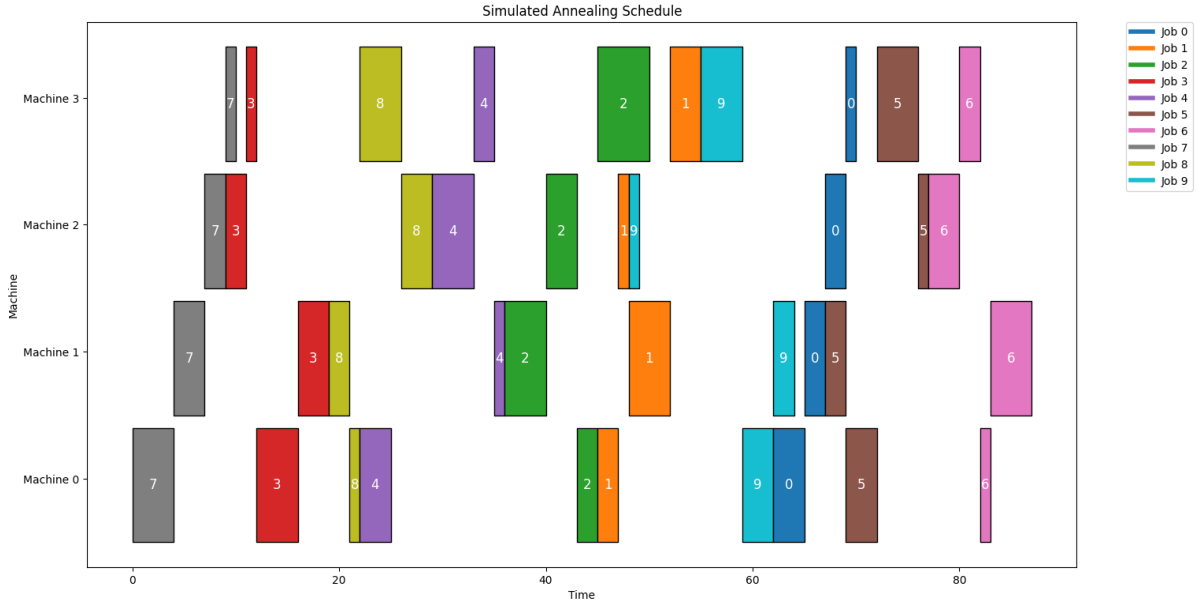

Figura 2: Gráfico de Gantt para la Solución Obtenida con Simulated Annealing

En este sentido, se observa que, aunque ambas soluciones muestran una alta eficiencia en la utilización de máquinas, la solución de Recocido Simulado es superior al lograr un makespan menor. Este resultado destaca la ventaja del Recocido Simulado en encontrar soluciones más cercanas al óptimo global al equilibrar la exploración y la explotación del espacio de soluciones.

6. Conclusiones

En este estudio se abordó el problema de programación de talleres flexibles (JSP) utilizando dos algoritmos de optimización: Hill Climbing y Simulated Annealing. A través de la implementación y análisis de ambos métodos, se obtuvieron soluciones para la secuenciación y asignación de trabajos en múltiples máquinas, con el objetivo de minimizar el tiempo total necesario para completar todos los trabajos, conocido como makespan.

Los resultados mostraron que:

- El algoritmo Hill Climbing, aunque eficiente en la asignación de tareas a las máquinas, tiende a quedar atrapado en óptimos locales, lo que limita su capacidad para encontrar la solución más eficiente. La solución obtenida con este método presentó un makespan de 90 unidades de tiempo.
- El algoritmo de Simulated Annealing, por su parte, demostró una mayor capacidad para explorar el espacio de soluciones al aceptar temporalmente soluciones subóptimas. Esto permitió escapar de los óptimos locales y encontrar una mejor secuencia de operaciones, resultando en un makespan de 87 unidades de tiempo, que es inferior al obtenido con Hill Climbing.

Ambos métodos mostraron una alta utilización de las máquinas con periodos mínimos de inactividad, lo que refleja una programación eficiente. Sin embargo, la solución de Simulated Annealing

superó a la de Hill Climbing al lograr una mejor distribución de las operaciones y reducir los tiempos de espera entre ellas.

En conclusión, aunque Hill Climbing es un método sencillo y rápido, el Simulated Annealing demostró ser más eficaz para este tipo de problemas, proporcionando soluciones de mayor calidad al explorar más combinaciones posibles. Para futuros trabajos, se sugiere explorar la combinación de ambos métodos o la aplicación de otros algoritmos de optimización, como los algoritmos genéticos o técnicas de inteligencia artificial, para mejorar aún más los resultados obtenidos y acercarse al óptimo global del problema.

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