Implementación y Comparación de Hill Climbing y Simulated Annealing para el problema Flow Shop

Winston Flores Yesenia Chirinos Piero Palacios

2025-05-30

Introducción

En la industria manufacturera moderna, la programación de la producción es un pilar esencial para garantizar la eficiencia operativa y la competitividad. Dentro de este ámbito, el problema Flow Shop —donde un conjunto de trabajos debe pasar por una serie de máquinas en un orden fijo— representa un caso real y frecuente: cada producto recorre varias etapas de procesamiento antes de completarse, y el tiempo total hasta que el último trabajo abandona la última máquina (el makespan) define la capacidad de respuesta de la planta.

Minimizar el makespan tiene un impacto directo sobre la reducción de inventarios en proceso, la acortación de los plazos de entrega y la disminución de costos de operación. Sin embargo, esta variante de Flow Shop es NP-duro, lo que implica que los métodos exactos resultan impracticables para cantidades de trabajos y máquinas de tamaño medio o grande: el espacio de soluciones crece de manera exponencial y el tiempo de cómputo se dispara.

Frente a esta dificultad, las heurísticas y metaheurísticas de búsqueda local ofrecen un compromiso entre calidad de la solución y tiempo de ejecución. En particular, Hill Climbing y Simulated Annealing destacan por su simplicidad y versatilidad:

- Hill Climbing explora sistemáticamente vecinos mejores para subir "colina arriba" hasta llegar a un óptimo local.
- Simulated Annealing introduce la posibilidad de aceptar empeoramientos controlados, analogía al enfriamiento de un metal, lo que facilita escapar de óptimos locales y explorar soluciones de mayor calidad.

El presente trabajo se motiva en comparar empíricamente estas dos estrategias en instancias representativas del Flow Shop, evaluando cómo balancean la reducción de makespan y el tiempo de cómputo a medida que aumenta la escala del problema. Este análisis aporta

criterios prácticos para seleccionar la metaheurística más adecuada según las necesidades de cada entorno productivo.

Metodología

Hill Climbing

Para abordar el Problema de Programación de Flujo de Taller (Flow Shop Scheduling Problem), se implementó un algoritmo de Escalada Simple (Hill Climbing). El objetivo es encontrar una secuencia de trabajos que minimice el makespan, es decir, el tiempo total requerido para completar todos los trabajos en todas las máquinas. La metodología se detalla a continuación:

1. Generación de la Matriz de Tiempos de Procesamiento:

- Se generó una matriz de tiempos de procesamiento donde las filas representan los trabajos y las columnas representan las máquinas. Las dimensiones de esta matriz son $N \times M$, siendo N el número de trabajos y M el número de máquinas.
- Cada celda (i,j) de la matriz contiene el tiempo que el trabajo i requiere en la máquina j. Estos tiempos se generaron aleatoriamente utilizando una distribución exponencial. Se eligió esta distribución debido a su propiedad de "no memoria" (memorylessness), la cual implica que la probabilidad de que una tarea dure un tiempo adicional no depende del tiempo que ya ha estado en procesamiento. Formalmente, si X es el tiempo de procesamiento de una tarea, entonces:

$$P(X > s + t | X > s) = P(X > t)$$

- Los valores generados se acotan dentro de un rango `minimo`, `maximo`para asegurar tiempos de procesamiento realistas, de acuerdo con los parámetros minimo y maximo definidos en la inicialización.

2. Generación de la Solución Inicial:

- El algoritmo comienza con una solución inicial. Esta solución es una secuencia (permutación) de los N trabajos.
- Si no se proporciona una solución inicial específica, se genera una de forma aleatoria barajando una lista ordenada de los índices de los trabajos (de 0 a $N - 1$).

3. Cálculo de los Tiempos de Finalización (Completion Times):

- Dada una secuencia de trabajos $S = (s_0, s_1, ..., s_{N-1})$ y la matriz de tiempos de procesamiento TP, se calcula una matriz de tiempos de finalización C. $C_{k,j}$ representa el tiempo en el que el k-ésimo trabajo de la secuencia S (es decir, s_k) finaliza su procesamiento en la máquina j.
- El cálculo se realiza mediante las siguientes recurrencias:
  - Para el primer trabajo en la secuencia (s_0) y la primera máquina (máquina 0): $C_{0,0} = TP_{s_0,0}$
  - Para el primer trabajo en la secuencia (s_0) y una máquina $j > 0$: $C_{0,j} = C_{0,j-1} + TP_{s_0,j}$
  - Para un trabajo $k > 0$ en la secuencia y la primera máquina (máquina 0): $C_{k,0} = C_{k-1,0} + TP_{s_k,0}$
  - Para un trabajo $k > 0$ en la secuencia y una máquina $j > 0$: $C_{k,j} = \max(C_{k-1,j}, C_{k,j-1}) + TP_{s_k,j}$
- Esta lógica se implementa en la función `_times_for_fitness`.

4. Función de Evaluación (Fitness):
   - La calidad de una solución (secuencia de trabajos) se mide por su makespan. El makespan es el tiempo total transcurrido desde el inicio del primer trabajo en la primera máquina hasta la finalización del último trabajo en la última máquina.
   - Corresponde al valor $C_{N-1,M-1}$ de la matriz de tiempos de finalización.
   - El objetivo del algoritmo es minimizar esta función de fitness. Esta se calcula en la función `_fitness`.
5. Generación de Vecinos:
   - A partir de una solución actual, se genera un conjunto de soluciones vecinas.
   - El vecindario se define mediante el operador de intercambio (swap): se generan nuevas secuencias intercambiando las posiciones de todos los pares posibles de trabajos en la secuencia actual. Esta operación se realiza en la función `_get_neighbors`.
6. Proceso Iterativo de Escalada Simple (Steepest Ascent):
   - El algoritmo opera de forma iterativa:
     1. Se comienza con la solución inicial y se calcula su fitness (makespan).
     2. En cada iteración, se generan todos los vecinos de la solución actual utilizando el operador de intercambio.
     3. Se calcula el fitness de cada vecino.
     4. Se selecciona el vecino con el mejor fitness (menor makespan).
     5. Si el fitness del mejor vecino es estrictamente mejor (menor) que el fitness de la solución actual, la solución actual se reemplaza por este mejor vecino. El proceso vuelve al paso 2.
     6. Si ningún vecino ofrece una mejora en el fitness, significa que se ha alcanzado un óptimo local. El algoritmo termina y devuelve la solución actual como la mejor encontrada.

- Esta lógica principal reside en la función `_perform_hill_climbing`.

7. Visualización (Opcional):
   - El sistema permite la visualización de los cronogramas de Gantt para la solución inicial y la solución óptima encontrada, si la opción `enable_plotting` está activada. Esto ayuda a comprender gráficamente la distribución de los trabajos en las máquinas a lo largo del tiempo.

Simulated Annealing

1. Generación de la Matriz de Tiempos de Procesamiento:
   - Se generó una matriz de tiempos de procesamiento donde las filas representan los trabajos y las columnas representan las máquinas. Las dimensiones de esta matriz son $N \times M$, siendo N el número de trabajos y M el número de máquinas.
   - Cada celda (i, j) de la matriz contiene el tiempo que el trabajo i requiere en la máquina j. Estos tiempos se generaron aleatoriamente utilizando una distribución exponencial. Se eligió esta distribución debido a su propiedad de "no memoria" (memorylessness), la cual implica que la probabilidad de que una tarea dure un tiempo adicional no depende del tiempo que ya ha estado en procesamiento. Formalmente, si X es el tiempo de procesamiento de una tarea, entonces:

$$P(X > s + t | X > s) = P(X > t)$$

   - Los valores generados se acotan dentro de un rango $[minimo, maximo]$ para asegurar tiempos de procesamiento realistas, de acuerdo con los parámetros `minimo` y `maximo` definidos en la inicialización.

2. Generación de la Solución Inicial:
   - El algoritmo comienza con una solución inicial. Esta solución es una secuencia (permutación) de los N trabajos.
   - Si no se proporciona una solución inicial específica, se genera una de forma aleatoria barajando una lista ordenada de los índices de los trabajos (de 0 a $N - 1$).

3. Cálculo de los Tiempos de Finalización (Completion Times):
   - Dada una secuencia de trabajos $S = (s_0, s_1, ..., s_{N-1})$ y la matriz de tiempos de procesamiento TP, se calcula una matriz de tiempos de finalización C. $C_{k,j}$ representa el tiempo en el que el k-ésimo trabajo de la secuencia S (es decir, s_k) finaliza su procesamiento en la máquina j.
   - El cálculo se realiza mediante las siguientes recurrencias:
     - Para el primer trabajo en la secuencia (s_0) y la primera máquina (máquina 0): $C_{0,0} = TP_{s_0,0}$

  - Para el primer trabajo en la secuencia (s_0) y una máquina $j > 0$: $C_{0,j} = C_{0,j-1} + TP_{s_0,j}$
  - Para un trabajo $k > 0$ en la secuencia y la primera máquina (máquina 0): $C_{k,0} = C_{k-1,0} + TP_{s_k,0}$
  - Para un trabajo $k > 0$ en la secuencia y una máquina $j > 0$: $C_{k,j} = \max(C_{k-1,j}, C_{k,j-1}) + TP_{s_k,j}$
- Esta lógica se implementa en la función `_times_for_fitness`.

4. Función de Evaluación (Fitness):
   - La calidad de una solución (secuencia de trabajos) se mide por su makespan. El makespan es el tiempo total transcurrido desde el inicio del primer trabajo en la primera máquina hasta la finalización del último trabajo en la última máquina.
   - Corresponde al valor $C_{N-1,M-1}$ de la matriz de tiempos de finalización.
   - El objetivo del algoritmo es minimizar esta función de fitness (energía). Esta se calcula en la función `_fitness`.
5. Generación de Vecinos (Simulated Annealing):
   - A partir de una solución actual S, se genera una única solución vecina S' de forma aleatoria.
   - El método implementado (`_random_neighbor`) consiste en seleccionar dos índices distintos al azar de la secuencia actual y intercambiar los trabajos en esas posiciones.
6. Proceso Iterativo de Recocido Simulado (Simulated Annealing):
   - El algoritmo opera de forma iterativa, simulando el proceso de enfriamiento lento de un metal. Los parámetros clave son:
     - T_{max}: Temperatura inicial.
     - T_{min}: Temperatura mínima (criterio de parada).
     - α: Tasa de enfriamiento (`cooling_rate`, típicamente un valor entre 0.8 y 0.99).
   - El proceso es el siguiente:
     1. Se inicializa la solución actual S con la solución generada en el paso 2. Se calcula su fitness $E = \text{fitness}(S)$.
     2. Se establece la mejor solución encontrada hasta el momento $S_{best} = S$ y su fitness $E_{best} = E$.
     3. Se inicializa la temperatura actual $T = T_{max}$.
     4. Mientras $T > T_{min}$:
        - Se genera una solución vecina S' a partir de S (como se describe en el punto 5).
        - Se calcula el fitness de la vecina $E' = \text{fitness}(S')$.
        - Se calcula la diferencia de energía (fitness): $\Delta E = E' - E$.
        - **Criterio de Aceptación (Metropolis):**
          * Si $\Delta E < 0$ (es decir, S' es mejor que S), se acepta S' como la nueva solución actual: $S = S'$, $E = E'$.

* Si $\Delta E \geq 0$ (es decir, S' es peor o igual que S), se acepta S' con una probabilidad P:

$$P = e^{-\Delta E/T}$$

Si un número aleatorio $rand \in [0, 1)$ es menor que P, entonces $S = S'$, $E = E'$. De lo contrario, se mantiene la solución S actual.

- Si la solución actual S tiene un fitness E mejor que E_{best}, se actualiza la mejor solución global: $S_{best} = S$, $E_{best} = E$.
- Se reduce la temperatura (enfriamiento): $T = T \times \alpha$.

5. El algoritmo termina cuando la temperatura actual T es menor o igual a T_{min}.

- La solución devuelta es S_{best}. Esta lógica principal reside en la función `_perform_simulated_annealing`.

7. Visualización (Opcional):

- El sistema permite la visualización de los cronogramas de Gantt para la solución inicial y la solución óptima encontrada por Simulated Annealing, si la opción `enable_plotting` está activada. Esto ayuda a comprender gráficamente la distribución de los trabajos en las máquinas a lo largo del tiempo.

Simulación de Montecarlo

1. Simulación de Montecarlo:

- Se comparó empíricamente el rendimiento de los algoritmos de Escalada Simple (*Hill Climbing*, HC) y Recocido Simulado (*Simulated Annealing*, SA) en la resolución del Problema de Programación de Flujo de Taller (*Flow Shop Scheduling Problem*).
- La comparación se centra en el tiempo de ejecución y, opcionalmente, en la calidad de la solución (fitness/makespan) obtenida por cada algoritmo bajo diversas configuraciones de tamaño del problema.

2. Parámetros de la Simulación:

- **Parámetros Comunes de Generación de Problemas:**
  - Tiempo medio para la distribución exponencial (`COMMON_MEAN_TIME`): 10.0 unidades.
  - Tiempo mínimo de procesamiento por tarea (`COMMON_MINIMO`): 2.0 unidades.
  - Tiempo máximo de procesamiento por tarea (`COMMON_MAXIMO`): 25.0 unidades.
- **Parámetros de Monte Carlo:**
  - Número de ejecuciones por configuración de problema (`NUM_MONTE_CARLO_RUNS`): 1000 (variable para pruebas).
  - Lista de número de trabajos a evaluar (`LIST_NUM_JOBS`): $[5, 10, 15]$.
  - Lista de número de máquinas a evaluar (`LIST_NUM_MACHINES`): $[10, 30, 60]$.

- **Parámetros del Algoritmo Simulated Annealing (SA) (fijos para esta comparación):**
  - Temperatura máxima inicial (`SA_T_MAX`): 1000.
  - Temperatura mínima final (`SA_T_MIN`): 0.1.
  - Tasa de enfriamiento (`SA_COOLING_RATE`): 0.95.
  - (Los parámetros del algoritmo Hill Climbing se asumen definidos dentro de su clase o con valores por defecto).

3. Generación de Instancias de Problemas:

   - Se utilizó una función (`generate_problem_instance`) para crear cada instancia específica del problema.
   - Cada instancia se generó utilizando una semilla única (`problem_seed`) para asegurar la reproducibilidad y diversidad de los problemas. Esta semilla se usa para inicializar el generador de números aleatorios de NumPy (`np.random.seed()`) al inicio de la generación de cada instancia.
   - La **matriz de tiempos de procesamiento** ($N_{jobs} \times N_{machines}$) se generó utilizando tiempos aleatorios de una distribución exponencial con la media especificada (`COMMON_MEAN_TIME`). Posteriormente, estos tiempos son acotados (clipping) para que se encuentren dentro del rango [`COMMON_MINIMO`, `COMMON_MAXIMO`].
   - Se genera una **solución inicial aleatoria** como una permutación de los índices de los trabajos.

4. Diseño Experimental y Ejecución:

   - El experimento se estructuró mediante bucles anidados:
     - Los bucles exteriores iteran sobre todas las combinaciones de número de trabajos (`n_jobs` de `LIST_NUM_JOBS`) y número de máquinas (`n_machines` de `LIST_NUM_MACHINES`). Cada combinación define una **configuración de problema**.
     - Para cada configuración de problema, se instancia un resolvedor Hill Climbing (`hc_solver`) y un resolvedor Simulated Annealing (`sa_solver`) una única vez. Se les pasa `seed=None` para que utilicen el estado global de `np.random` (controlado por `problem_instance_seed` al generar la instancia) para cualquier aleatoriedad interna, asegurando que ambos algoritmos enfrenten condiciones aleatorias consistentes para una instancia dada si dependen de `np.random` de la misma manera.
     - El bucle interior ejecuta la simulación `NUM_MONTE_CARLO_RUNS` veces para la configuración actual.
   - Dentro de cada ejecución del bucle de Monte Carlo:
     - Se genera una instancia única del problema (matriz de tiempos y solución inicial) utilizando la función `generate_problem_instance` con una semilla `problem_instance_seed` única para esa ejecución (derivada de `n_jobs`, `n_machines`, y el índice de la corrida `i_run`).

- Se ejecuta el algoritmo **Hill Climbing**:
  - Se registra el tiempo de inicio (`time.perf_counter()`).
  - Se invoca el método `hc_solver.solve()` pasando copias profundas (`deepcopy`) de la matriz de tiempos y la solución inicial generadas para la instancia actual.
  - Se registra el tiempo de finalización. El tiempo de ejecución total y el fitness (makespan) de la solución obtenida se almacenan.
- Se ejecuta el algoritmo **Simulated Annealing**:
  - Se registra el tiempo de inicio.
  - Se invoca el método `sa_solver.solve()` pasando copias profundas (`deepcopy`) de la misma matriz de tiempos y solución inicial.
  - Se registra el tiempo de finalización. El tiempo de ejecución total y el fitness de la solución obtenida se almacenan.
- El uso de `deepcopy` asegura que ambos algoritmos reciban exactamente la misma instancia de problema sin modificaciones previas.

5. Recolección y Almacenamiento de Datos:

   - Los tiempos de ejecución de cada algoritmo para cada corrida de Monte Carlo se almacenaron en listas asociadas a su configuración de problema (tupla `(n_jobs, n_machines)`).
   - Similarmente, los valores de fitness (makespan) obtenidos se almacenan.
   - Se utilizan diccionarios (`results` para tiempos, `results_fitness` para fitness) donde la clave principal es el nombre del algoritmo ('hc' o 'sa') y el valor es otro diccionario cuya clave es la configuración del problema y el valor es la lista de resultados (tiempos o fitness) de todas las `NUM_MONTE_CARLO_RUNS`.

6. Análisis y Presentación de Resultados:

   - Una vez finalizadas todas las ejecuciones de Monte Carlo para todas las configuraciones:
     - Se calculan la **media** y la **desviación estándar** de los tiempos de ejecución para cada algoritmo (HC y SA) y para cada configuración de problema (`(n_jobs, n_machines)`).
     - Los resultados agregados (número de trabajos, número de máquinas, tiempo medio HC, desviación estándar HC, tiempo medio SA, desviación estándar SA) se organizan en una estructura de datos.
     - Estos datos se presentan en formato tabular utilizando un DataFrame de Pandas, que se imprime en la consola.
     - (Un análisis similar se podría realizar para los valores de fitness almacenados en `results_fitness`).

Resultados

Resultados

A continuación, se presentan los resultados obtenidos tanto de ejecuciones individuales de los algoritmos Hill Climbing y Simulated Annealing como de la simulación de Montecarlo realizada para comparar su rendimiento.

Hill Climbing

Los siguientes diagramas de Gantt ilustran el estado de la programación de tareas antes y después de la aplicación del algoritmo Hill Climbing para una instancia específica del problema.

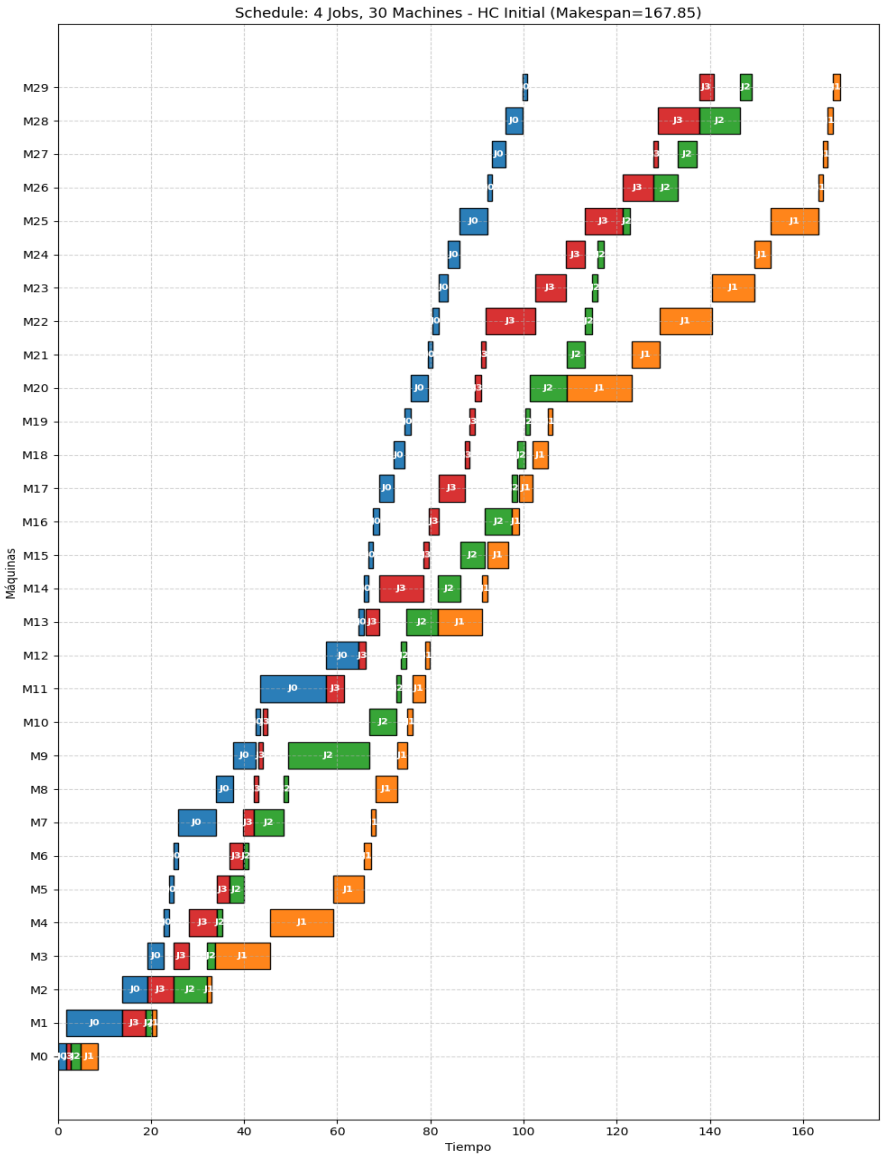

Figura 1: Inicio de Hill CLimbing

El diagrama de Gantt superior muestra la programación inicial para un problema de 4 trabajos y 30 máquinas, con un makespan de 167.85. Esta solución inicial se genera aleatoriamente.

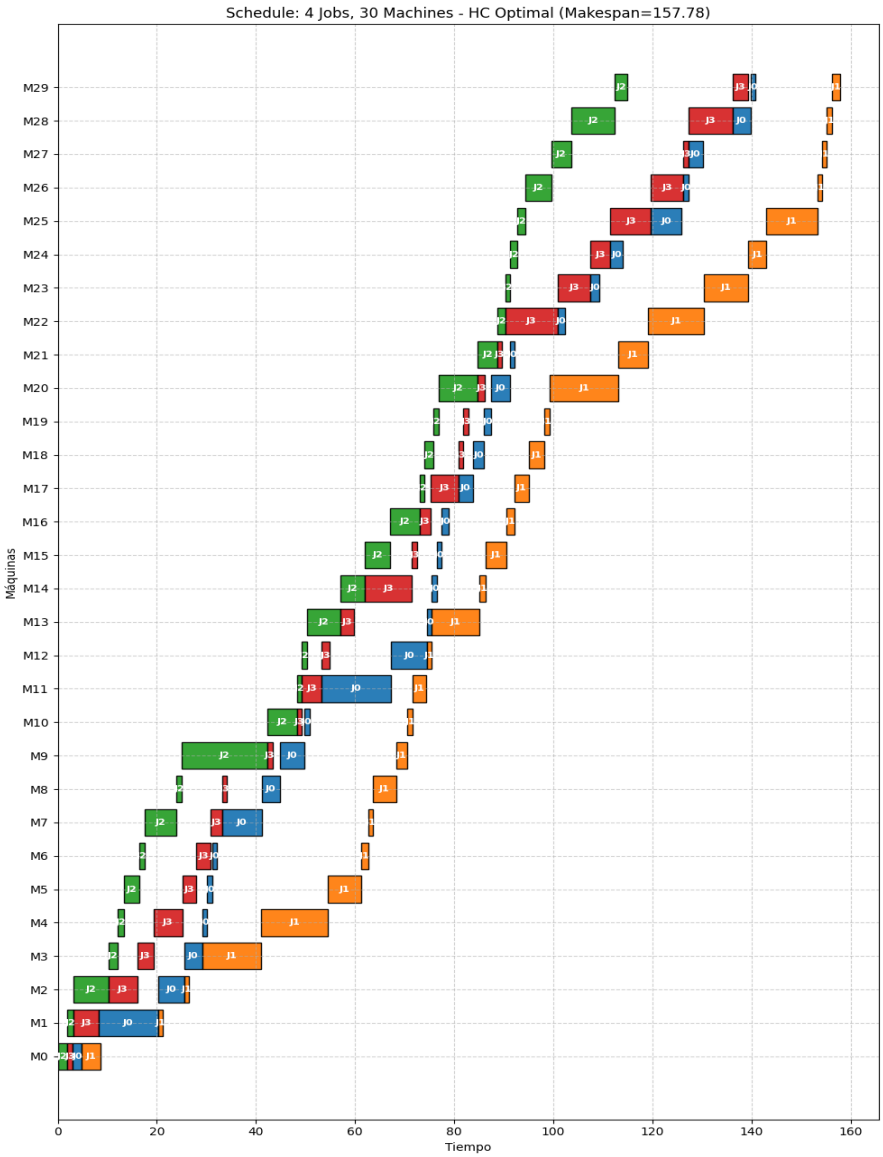

Figura 2: Final de Hill CLimbing

Tras la aplicación del algoritmo Hill Climbing, se obtiene la programación óptima local que se muestra arriba. El makespan se ha reducido a 157.78. Esta mejora es el resultado del proceso iterativo del algoritmo, que explora soluciones vecinas (mediante intercambios de trabajos) y se mueve consistentemente hacia aquellas que ofrecen un mejor makespan, hasta que no se encuentran mejoras adicionales en el vecindario inmediato.

Simulated Annealing

De manera similar, se presentan los diagramas de Gantt para el algoritmo Simulated Annealing, mostrando el progreso desde una solución inicial hasta la solución final obtenida.

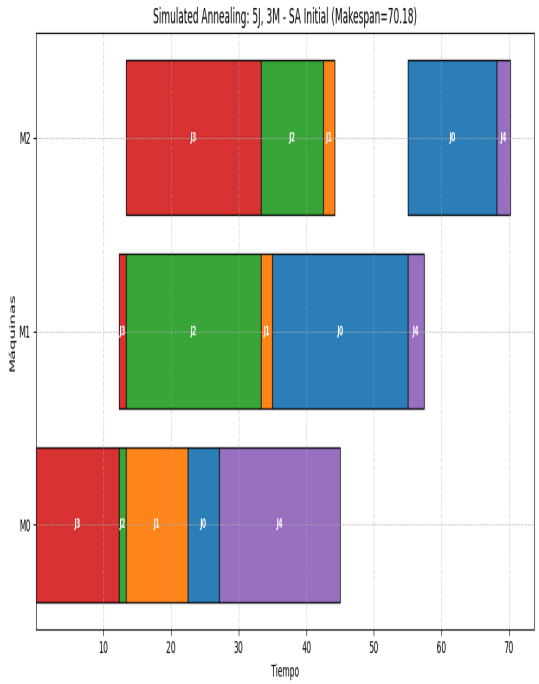

Figura 3: Inicio de Simulated Annealing

Para una instancia de 5 trabajos y 3 máquinas, la solución inicial aleatoria presenta un makespan de 70.18, como se observa en este diagrama.

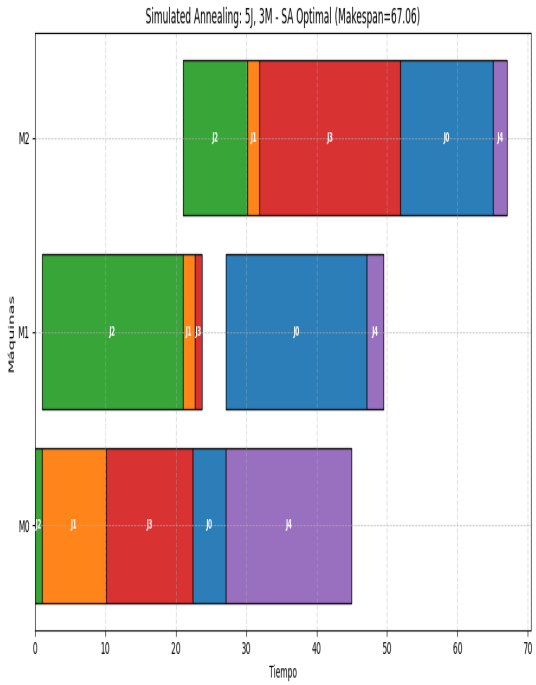

Figura 4: Final de Simulated Annealing

El algoritmo Simulated Annealing logra mejorar la programación, resultando en un makespan de 67.06. Esta optimización se alcanza mediante un proceso que no solo acepta soluciones estrictamente mejores, sino que también permite movimientos a soluciones peores con una cierta probabilidad (controlada por la "temperatura"). Esta característica permite al algoritmo Simulated Annealing escapar de óptimos locales y explorar una mayor parte del espacio de soluciones, llevando a una solución final de alta calidad a medida que el sistema se "enfría".

Simulación de Montecarlo

Para evaluar el rendimiento computacional de ambos algoritmos bajo diferentes condiciones, se realizó una simulación de Montecarlo. Los gráficos siguientes resumen los tiempos medios de ejecución.

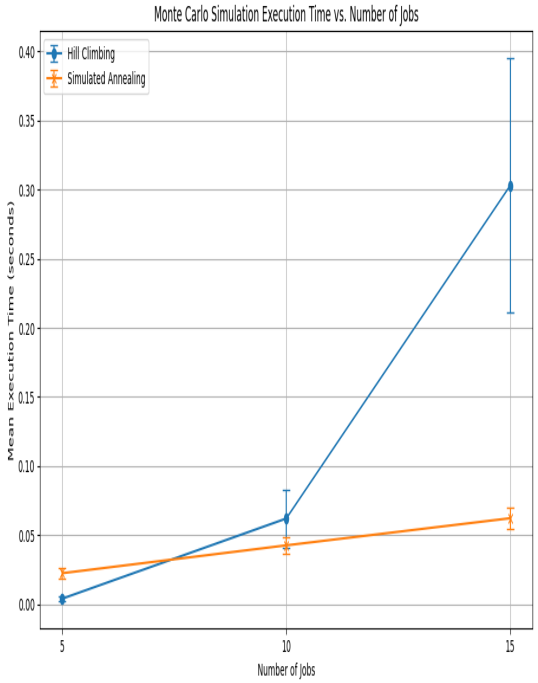

Figura 5: Trabajos vs Tiempo de Ejecución (Montecarlo)

Este gráfico ilustra la relación entre el número de trabajos y el tiempo medio de ejecución para Hill Climbing y Simulated Annealing, manteniendo constante el número de máquinas. Se observa que el tiempo de ejecución para ambos algoritmos tiende a aumentar con el número de trabajos. Notablemente, Simulated Annealing muestra un tiempo de ejecución medio consistentemente menor y una escalabilidad más favorable (un incremento más suave en el tiempo) a medida que aumenta el número de trabajos. Hill Climbing, en cambio, experimenta un aumento más pronunciado en su tiempo de ejecución, especialmente para 15 trabajos, y también presenta una mayor variabilidad en sus tiempos (representada por las barras de error).

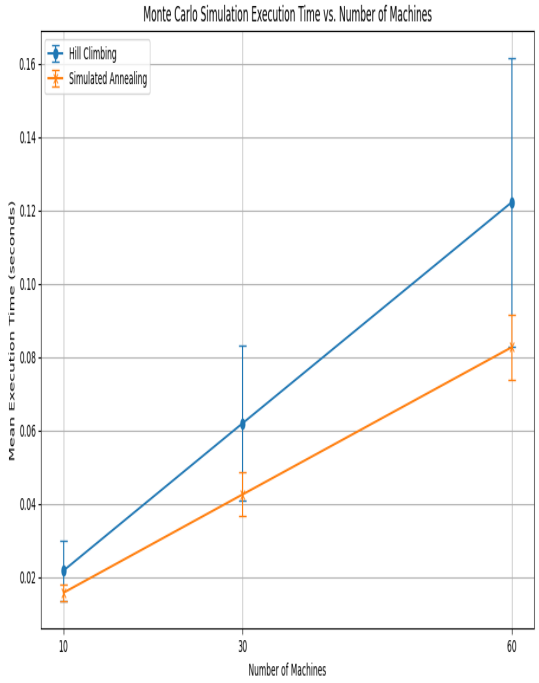

Figura 6: Máquinas vs Tiempo de Ejecución (Montecarlo)

Este segundo gráfico muestra el tiempo medio de ejecución en función del número de máquinas, manteniendo constante el número de trabajos. Al igual que con el número de trabajos, el tiempo de ejecución para ambos algoritmas se incrementa a medida que aumenta el número de máquinas, lo cual es esperado debido a la mayor complejidad en la evaluación de la función de fitness (cálculo del makespan). Simulated Annealing vuelve a mostrar un menor tiempo medio de ejecución y una tendencia de crecimiento más contenida en comparación con Hill Climbing. La variabilidad en los tiempos de ejecución también parece ser menor para Simulated Annealing en la mayoría de las configuraciones mostradas. Estos resultados sugieren que, bajo las condiciones y parámetros probados, Simulated Annealing es computacionalmente más eficiente.

Conclusiones

Este proyecto se centró en la implementación y comparación de dos algoritmos metaheurísticos, Escalada Simple (Hill Climbing) y Recocido Simulado (Simulated Annealing), para la resolución del Problema de Programación de Flujo de Taller (Flow Shop Scheduling Problem), culminando con una evaluación comparativa mediante simulación de Montecarlo.

1. Implementación del Algoritmo Hill Climbing (HC) para Flow Shop:

Se logró implementar exitosamente el algoritmo Hill Climbing. Las pruebas individuales, visualizadas mediante diagramas de Gantt, demostraron la capacidad del algoritmo para tomar una solución inicial aleatoria y mejorarla iterativamente, reduciendo el makespan. Por ejemplo, en una instancia de 4 trabajos y 30 máquinas, el makespan se redujo de 167.85 a 157.78. Esto confirma que la implementación puede converger hacia óptimos locales mediante una búsqueda de ascenso más pronunciado en el espacio de soluciones.

2. Implementación del Algoritmo Simulated Annealing (SA) para Flow Shop:

El algoritmo Simulated Annealing también fue implementado de manera satisfactoria. Al igual que con HC, los diagramas de Gantt para instancias de prueba (e.g., 5 trabajos y 3 máquinas con una reducción del makespan de 70.18 a 67.06) evidenciaron la efectividad del SA para encontrar soluciones de alta calidad. La capacidad del SA para aceptar soluciones peores de forma probabilística le permite explorar más ampliamente el espacio de búsqueda y evitar quedar atrapado prematuramente en óptimos locales, lo cual es una ventaja fundamental sobre el Hill Climbing simple.

3. Comparación de Hill Climbing y Simulated Annealing mediante Simulación de Montecarlo:

La simulación de Montecarlo permitió una comparación cuantitativa del rendimiento de ambos algoritmos en términos de eficiencia computacional bajo diversas configuraciones de problema:

- **Calidad de la Solución**: Ambos algoritmos demostraron ser capaces de mejorar significativamente las soluciones iniciales aleatorias, como se observó en las ejecuciones individuales. Si bien la simulación de Montecarlo se centró principalmente en el tiempo de ejecución, la naturaleza del Simulated Annealing, con su mecanismo para escapar de óptimos locales, sugiere una mayor robustez para encontrar soluciones de mejor calidad global en escenarios complejos, aunque esto requeriría un análisis más profundo de los datos de fitness de la simulación.
- **Eficiencia Computacional**: Los resultados de la simulación de Montecarlo fueron concluyentes respecto al tiempo de ejecución. Simulated Annealing demostró ser consistentemente más rápido y escalable que Hill Climbing a medida que aumentaba el número de trabajos y máquinas. Hill Climbing mostró un incremento más pronunciado en el tiempo de cómputo y una mayor variabilidad, especialmente en problemas de mayor tamaño. Por ejemplo, con 15 trabajos, la diferencia en el tiempo medio de ejecución fue notablemente favorable al SA.
- **Evaluación General**: Considerando tanto la capacidad de optimización como la eficiencia computacional, Simulated Annealing se perfila como una alternativa superior a Hill Climbing para el problema de Flow Shop en las configuraciones estudiadas. Su mejor rendimiento en tiempo y su inherente capacidad para una exploración más exhaustiva del espacio de soluciones lo convierten en una herramienta más robusta

y eficiente, especialmente para problemas de mayor envergadura. La simplicidad de Hill Climbing puede ser adecuada para problemas pequeños o como componente de algoritmos híbridos, pero su rendimiento y calidad de solución pueden ser limitados en escenarios más complejos.

En resumen, ambos objetivos de implementación fueron cumplidos, y la comparación mediante Montecarlo proveyó evidencia empírica que respalda la superioridad de Simulated Annealing en términos de eficiencia y potencial de optimización para el problema de Flow Shop bajo las condiciones analizadas.