Resolución de un problema de Job Shop Scheduling, mediante Simulated Annealing con hiperparámetros

Felipe Ignacio Arrans-Mellado [a], Matias Andrés Pino-Veloso [b], Diego Daniel San Martín [c], González Derqui Ariel Sanhueza-Balboa [d]

[a] *Facultad de ingeniería, Universidad Católica de la santísima Concepción, Chile. farrans@ing.ucsc.cl*
[b] *Facultad de ingeniería, Universidad Católica de la santísima Concepción, Chile. dsanhuezaba@ing.ucsc.cl*
[c] *Facultad de ingeniería, Universidad Católica de la santísima Concepción, Chile. dsanmartin@ing.ucsc.cl*
[d] *Facultad de ingeniería, Universidad Católica de la santísima Concepción, Chile. mpino@ing.ucsc.cl*
Docente evaluación: Víctor Yamil Neira González

Abstracto
En este trabajo, abordamos el problema de la programación de trabajos en una empresa de manufactura con múltiples máquinas, conocido como "job-shop scheduling problem". El objetivo es minimizar el tiempo requerido para completar todas las órdenes de trabajo, teniendo en cuenta que cada máquina tiene un tiempo de proceso distinto para cada orden. Implementamos el algoritmo de Simulated Annealing, una metaheurística popular para problemas de optimización combinatoria, para encontrar una asignación de órdenes a máquinas que minimice el tiempo de finalización del último trabajo. Nuestros resultados indican que el algoritmo es efectivo para encontrar buenas soluciones en un tiempo razonable, haciendo de este un método viable para la programación de trabajos en entornos de producción. Se espera que estos hallazgos sirvan como base para futuras investigaciones en el campo de la optimización de la programación de trabajos.

1. Descripción del problema

Estamos frente a un problema planteado por la empresa GALEM de manufactura. Este problema se centra en la programación eficiente de tareas en el taller de pedidos. Con un número determinado de máquinas y órdenes de manufactura el objetivo es minimizar el tiempo total requerido por las órdenes solicitadas. Cada máquina tiene tiempos de procesamiento distintos por orden, lo que da la dificultad a esta problemática. La solución que se busca es asignar las órdenes a las máquinas de la manera más óptima posible, de modo que se pueda maximizar la eficiencia y reducir el tiempo de producción global. La idea principal es encontrar un algoritmo que distribuya las órdenes de manera equilibrada a las máquinas, considerando sus tiempos de trabajo individual por orden, con el fin de lograr una entrega rápida y eficiente de los pedidos.

2. Modelamiento matemático del problema

a. Parámetros

- N número de ordenes de trabajo
  M cantidad de maquinas
- t_{ij} tiempo para que la maquina i termine la orden j

b. Variables de decisión

- x_{ij} variable binaria si la orden i es asignada a la maquina j

c. Función objetivo

$$\min \sum_{i=1}^{N} \sum_{j=1}^{M} t_{ij} * x_{ij}$$

d. Restricciones

$$\sum_{j=1}^{M} x_{ij} = 1 \; \forall \; 1 \leq \; i \leq N$$

$$\sum_{i=1}^{N} x_{ij} \leq 1 \; \forall \; 1 \leq \; j \leq M$$

$$x_{ij} \, \epsilon \, \{0, 1\} \, \forall \, 1 \leq i \leq N, 1 \leq j \leq M$$

- Cada orden debe estar asignada solo a una única maquina
- Cada maquina puede trabajar como máximo en una orden a la vez
- x_{ij} pertenece {0, 1} para todo i = 1 hasta N, j = 1 hasta M

3. Método de solución

1. Iniciar definiendo los parámetros num_order, num_maq, T_inicial, enfriamiento, T_final.
2. Fijar los mejores hiperparámetros (best_hyperparameters) y el mejor valor (best_value) como no definidos y infinito, respectivamente.
3. Por cada valor en el conjunto T_inicial:
4. Para cada valor en el conjunto enfriamiento:
5. Para cada valor en el conjunto T_final:
6. Ejecutar la función de metaheurística con los parámetros num_maq, num_order, T_initial, C y Tfinal.
7. Si el valor de mejor_valor obtenido es inferior al best_value actual:
8. Actualizar best_value con el valor de mejor_valor.
9. Actualizar best_hyperparameters con los valores de T_initial, C, Tfinal y los arrays T_array y Costo_array.
10. Al final del proceso, retornar los mejores hperparámetros y el mejor valor.

La función de metaheurística funcionaría de la siguiente manera:

1. Generar una solución inicial aleatoria.
2. Calcular el valor de la función objetivo para esta solución inicial.
3. Establecer tanto la solución actual (solucion_actual) como la mejor solución (mejor_solucion) con la solución inicial.
4. Establecer tanto el valor actual (valor_actual) como el mejor valor (mejor_valor) con el valor de la función objetivo.
5. Iniciar un ciclo que continuará mientras que la temperatura (T) sea mayor que la temperatura final (Tfinal):
6. Generar una nueva solución modificando aleatoriamente la solución actual.
7. Calcular el valor de la función objetivo para esta nueva solución.
8. Si el criterio de aceptación aprueba la nueva solución, actualizar la solución actual y el valor actual con la nueva solución y el nuevo valor, respectivamente.
9. Si el nuevo valor es inferior al mejor valor, actualizar la mejor solución y el mejor valor con la nueva solución y el nuevo valor, respectivamente.
10. Enfriar la temperatura (T) multiplicándola por el coeficiente de enfriamiento (C).
11. Incrementar un contador (i).
12. Agregar la temperatura y el nuevo valor a los arrays T_array y Costo_array, respectivamente.
13. Al final del ciclo, retornar el mejor valor y los hiperparámetros (T_initial, C, Tfinal, T_array, Costo_array).
14. Este proceso se repite para cada combinación de T_initial, C y Tfinal.

Al final de todo, se devolverán los hiperparámetros que resultaron en el mejor valor de la función objetivo.

4. Tabla con objetivos logrados

	Valor	T inicial	T final	Enfriamiento
1	163	10000	0.00001	0.95
2	146	1000	0.0001	0.95
3	309	500	0.001	0.9
4	337	100	0.01	0.9
5	300	50	0.1	0.85

5. Gráficos

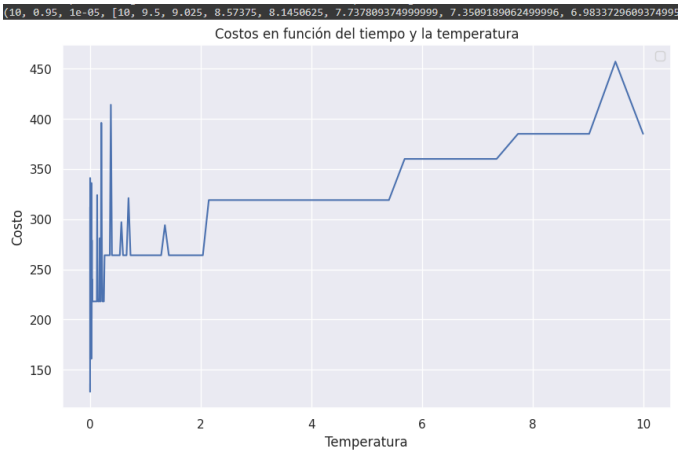

Gráfico 1. Temp inicial 10, Enfriamiento 0,95, Temp final 0,00001, resultado 128,0.

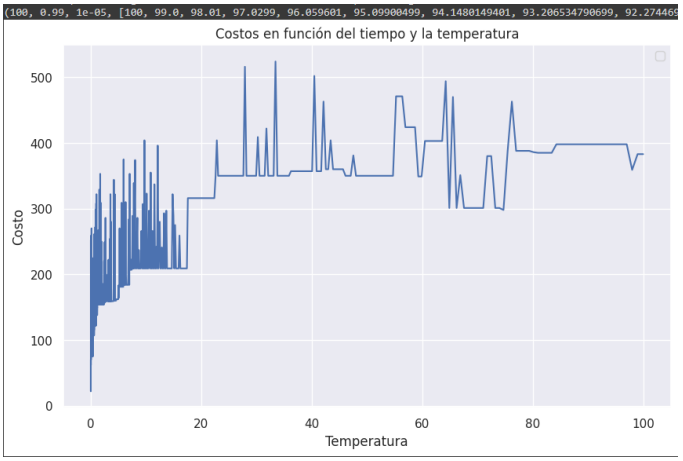

Gráfico 2. Temp inicial 100, Enfriamiento 0,99, Temp final 0,00001, tiempo solución 22,0 minutos.

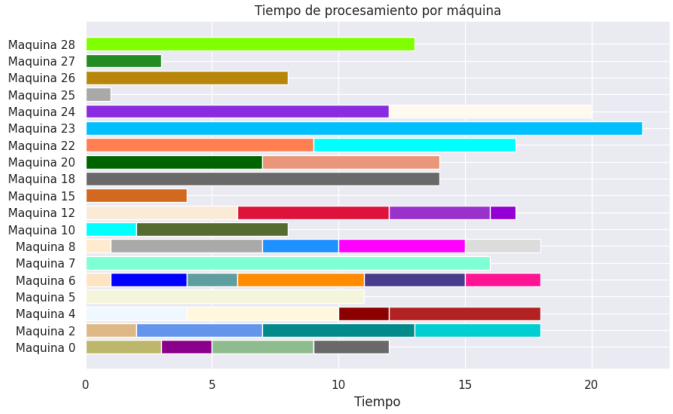

Gráfico 3. Grafico de barras que presenta el tiempo que tardan las maquinas por orden asignada.

```
Maquina 0: [27, 28, 34, 37, 42]
Maquina 1: []
Maquina 2: [12, 17, 22, 38]
Maquina 3: []
Maquina 4: [0, 11, 18, 32, 45]
Maquina 5: [5]
Maquina 6: [6, 9, 13, 30, 35, 36, 40]
Maquina 7: [3]
Maquina 8: [7, 8, 21, 26, 44, 48, 49]
Maquina 9: []
Maquina 10: [2, 29]
Maquina 11: []
Maquina 12: [1, 4, 19, 31, 39]
Maquina 13: []
Maquina 14: []
Maquina 15: [15]
Maquina 16: []
Maquina 17: []
Maquina 18: [43]
Maquina 19: []
Maquina 20: [25, 33]
Maquina 21: []
Maquina 22: [16, 20]
Maquina 23: [41]
Maquina 24: [10, 46]
Maquina 25: [24]
Maquina 26: [23]
Maquina 27: [47]
Maquina 28: [14]
Maquina 29: []
```

Datos de asignación por maquina y órdenes.

6. Conclusión

En este estudio se trabajó la metodología Simulated Annealing basada en una metaheurística la cual busca encontrar la mejor solución en un espacio de búsqueda grande y complejo, en este caso es la asignación de órdenes a cada máquina, en la cual, basándose en pruebas, aceptación, temperatura actual y la diferencia entre los valores, permite encontrar una solución óptima. Ajustando la temperatura inicial, factor de enfriamiento y la temperatura final, el algoritmo busca encontrar la mejor solución posible generando gráficos que nos entregan visualmente el comportamiento del algoritmo mostrando que como resultado tenemos (valores) de costo.

Los resultados nos entregan la mejor solución encontrada en 100 ejecuciones del algoritmo, teniendo en cuenta la variabilidad de los valores de entrada y parámetros utilizados. En resumidas palabras esta implementación de este método nos demuestra la efectividad que posee para abordar este problema de asignación de órdenes por máquina, basándonos en un código que nos sirve para explorar y optimizar las soluciones generando gráficos que permiten la mejor comprensión del proceso de optimización y resultados obtenidos. Cabe mencionar que las soluciones encontradas podrían no ser las mejores soluciones factibles reales, estas fueron las que nosotros encontramos mediante nuestro algoritmo.

7. Referencias

Géron, A. (2019). Simulated Annealing from Scratch in Python. Machine Learning Mastery. Recuperado de: https://machinelearningmastery.com/simulated-annealing-from-scratch-in-python/ (forma del codigo y funciones a considerar)

Besson, O. (s.f.). Simulated Annealing in Python. Recuperado de: https://perso.crans.org/besson/publis/notebooks/Simulated_annealing_in_Python.html(Variables para graficar)

Bergstra, J., Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. Journal of Machine Learning Research, 13, 281-305. Recuperado de: https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf (Hiperparametros de búsqueda en cuadricula)

Binato, S., Hery, W. J., Loewenstern, D. M., & Resende, M. G. C. (2000). A greedy randomized adaptive search procedure for job shop scheduling. Research Society, Recuperado de: https://www.researchgate.net/publication/228792021_A_greedy_randomized_adaptive_search_procedure_for_job_shop_scheduling (Base estructural de simulated annealing)