Simulated Annealing y Genetic Algorithm aplicado a JSP

Proyecto Final

Nicolás Fuenzalida
Sebastián Lemus

Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ingeniería Matemática

16 de diciembre de 2023

Introducción

El problema de programación de trabajos, o JSP, es un problema muy estudiado en optimización combinatorial. Se abordará JSP con simulated annealing (SA) y el algoritmo genético (GA).

JSP se describe como, dado n trabajos, cada uno es resuelto por las m máquinas, a cada realización se le asigna una operación, siendo exactamente m operaciones en cada máquina. El objetivo es encontrar un orden de las operaciones tal que el tiempo de finalización de procesar todas se minimice. [Kolonko, 1999]

Introducción

A continuación, vemos un ejemplo de JSP. Este horario consiste en 3 máquinas en el eje OY, 4 tareas representadas con diferentes colores y tiene un costo de 29 minutos, vista por la última operación en ejecutarse.[Piroozfard et al., 2016]

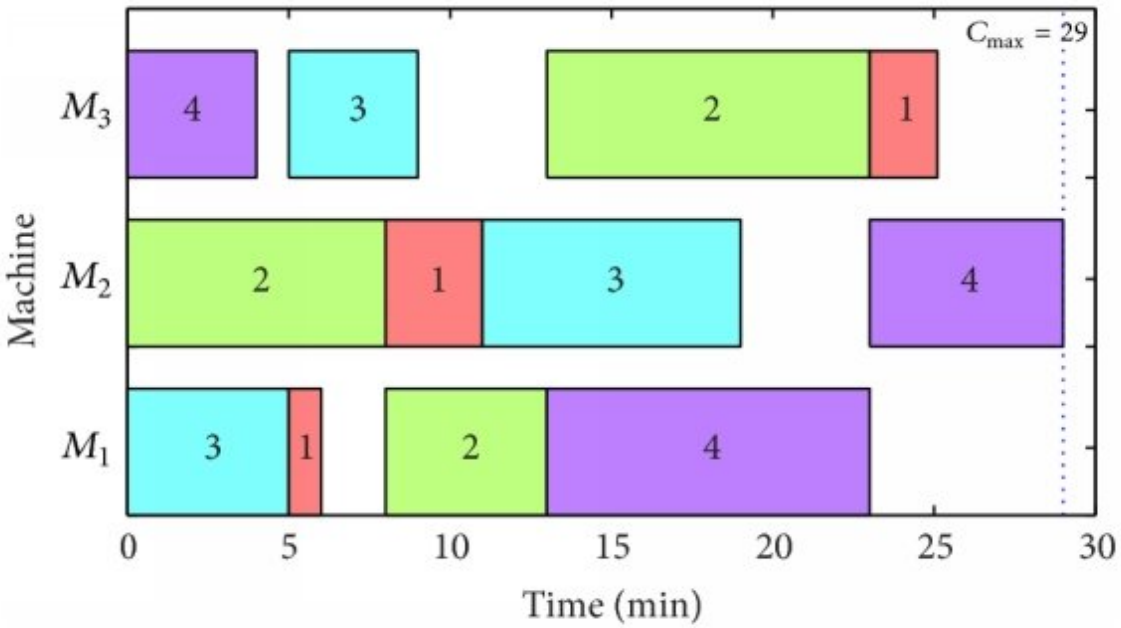

Figura 1: Horario de 3 máquinas y 4 tareas.

Contexto

- En JSP, se dan n tareas, cada una compuesta por m operaciones, y hay m máquinas. Cada máquina debe realizar una única vez cada tarea y no se puede realizar la misma tarea en distintas máquinas al mismo tiempo.
- El orden en el que las operaciones de una tarea debe realizarse está fijo para un horario.
- El objetivo es encontrar un orden σ de las operaciones en cada tarea de modo tal que el tiempo de finalización se minimice.

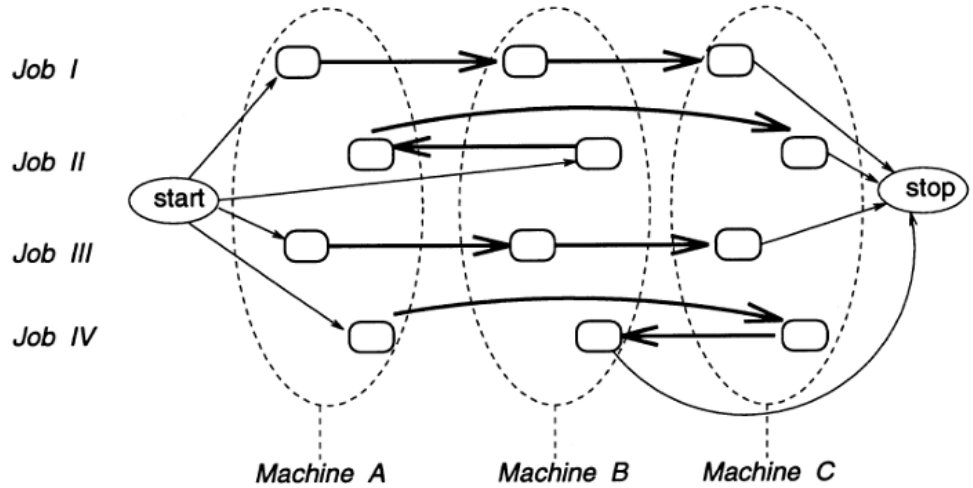

Figura 2: Horario de 3 máquinas y 4 tareas.

Contexto

- Se considera el conjunto de los horarios factibles como todas las posibles combinaciones de las operaciones, las cuales no pueden tener ciclos, denotado S. Un elemento de este conjunto se vería de la siguiente forma:

$$\sigma = (o_1, ..., o_{mn})$$

  donde $o_i, i \in \{1, ..., mn\}$ es una operación.

- El costo $c : S \to \mathbb{R}^+$ de un horario factible se obtiene como el tiempo de finalización de la última operación, se denota $c(\sigma)$, con σ un orden de operaciones. Esta es la función a ser minimizada.

Contexto

- Dada una solución factible σ de un horario, definimos la vecindad de σ como todas las posibles permutaciones entre pares de índices de σ, denotada $N(\sigma) \subseteq S$.
- Por ejemplo, para 2 tareas y 2 máquinas, una solución factible es

$$\sigma = (1, 2, 3, 4)$$

así un vecino $\tau \in N(\sigma)$ puede ser

$$\tau = (1, 3, 2, 4).$$

Contexto

- Para abordar el problema, se usarán dos algoritmos: el primero es Simulated Annealing y el segundo es el Algoritmo Genético. Ambos representan una solución heurística del problema, dado que, por la naturaleza del mismo, es muy difícil de resolver.

Simulated annealing

- Se crea una función *neighborhood*, la cual cambia de orden exactamente dos operaciones de un horario σ y retorna un vecino τ.
- Se crea la función beta y la función de costos (tiempo).
- Con todo esto, se implementa el método simulated annealing, la cual retorna la simulación de la cadena de Markov junto a los costos asociados.

Simulated Annealing

- La figura (3) muestra un estado inicial σ_0 dado, que tiene 10 máquinas y 10 tareas. A este estado se aplicará el método de simulated annealing.

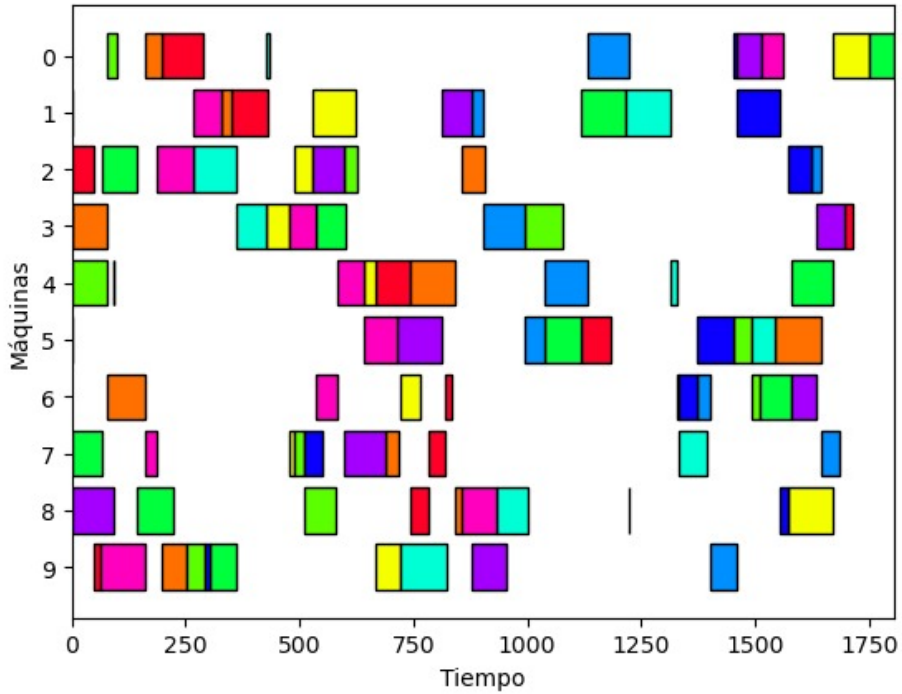

Figura 3: Estado inicial σ_0

- Donde $c(\sigma_0) = 1804$ y se iteró 20.000 pasos en la cadena de Markov.

Función beta (1/5)

Original | Raíz | Polinomial | Cuadrática | Exponencial

Se define la función beta original como

$$\beta_n = \frac{1}{C} \ln(n + e)$$

Se simulan 20.000 pasos de la cadena y $C = \max_i t_i + \Delta t_i$.

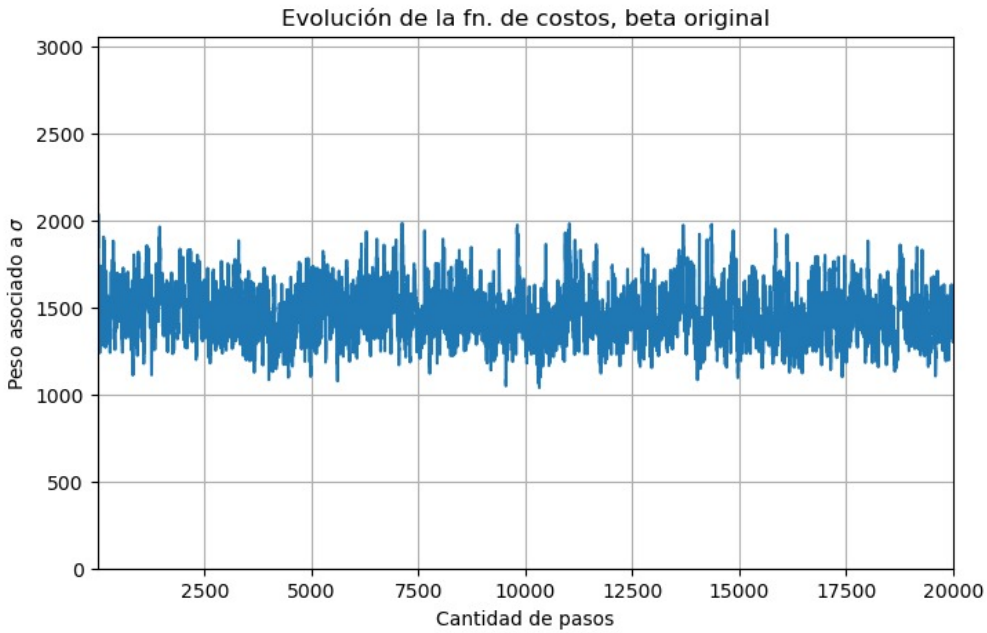

Figura 4: Evolución función de costos

Función beta (2/5)

Original | Raíz | Polinomial | Cuadrática | Exponencial

Se define la función beta raíz cuadrada como

$$\beta_n = \sqrt{\frac{n}{10^3}}$$

Se simulan 20.000 pasos de la cadena.

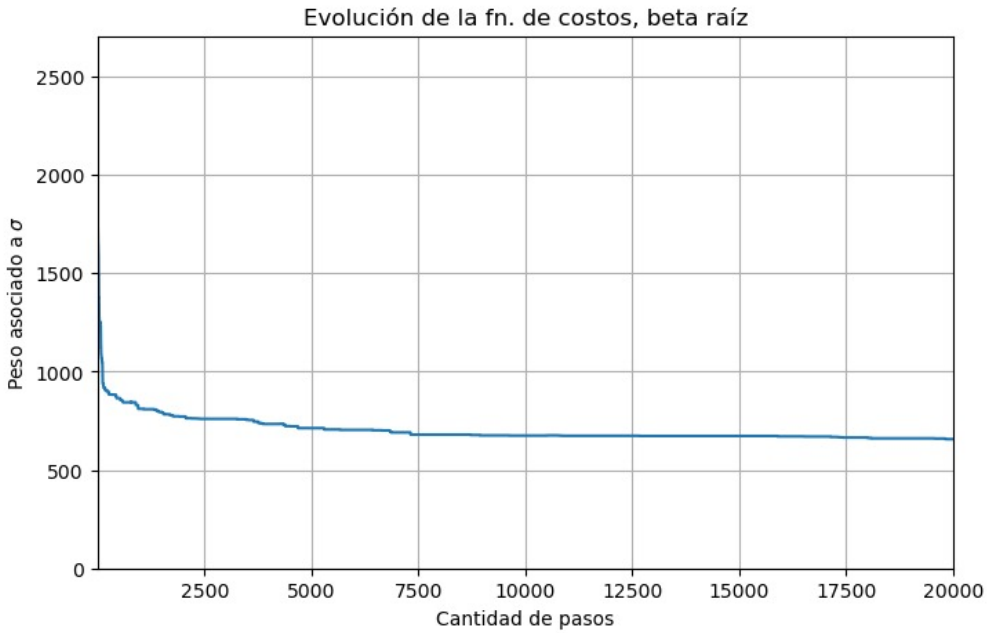

Figura 5: Evolución función de costos

Función beta (3/5)

Original | Raíz | **Polinomial** | Cuadrática | Exponencial

Se define la función beta polinomial como

$$\beta_n = \frac{n^4 + n^3 + n^2 + n}{10^{17}}$$

Se simulan 20.000 pasos de la cadena.

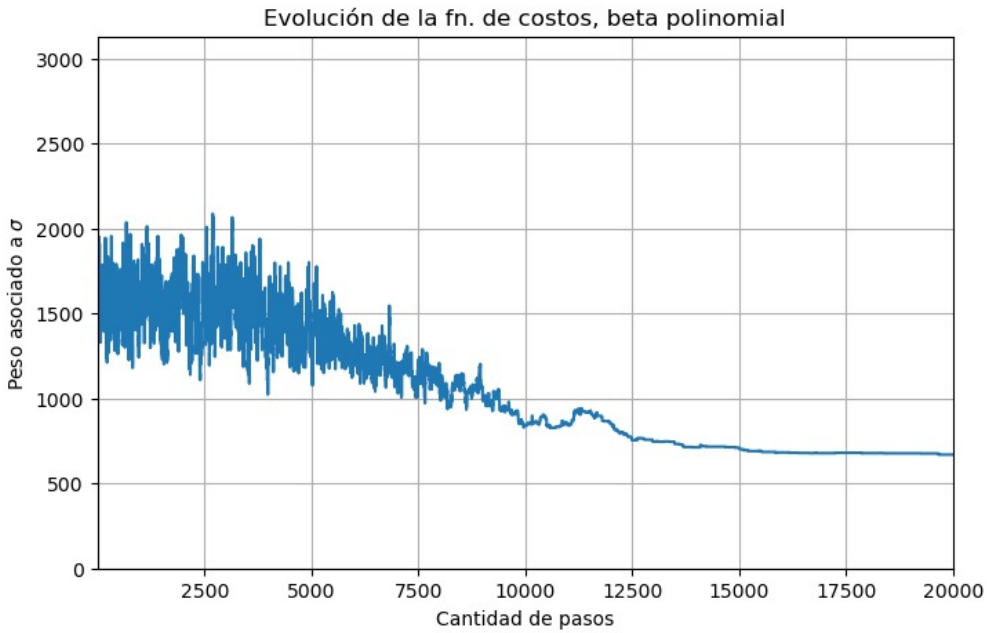

Figura 6: Evolución función de costos

Función beta (4/5)

Original | Raíz | Polinomial | **Cuadrática** | Exponencial

Se define la función beta cuadrática como

$$\beta_n = \frac{1}{10^7} n^2$$

Se simulan 20.000 pasos de la cadena.

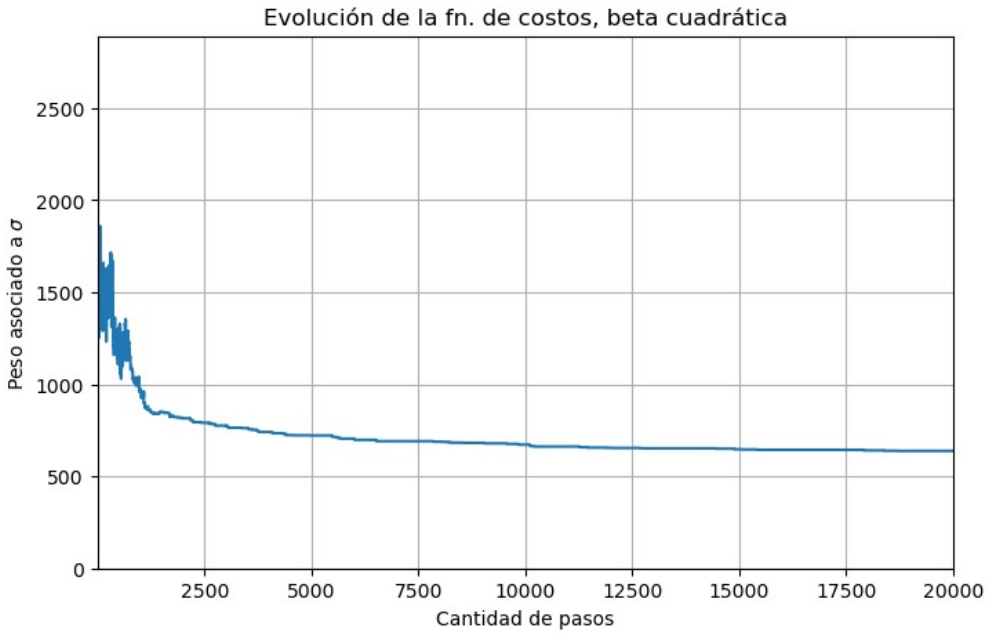

Figura 7: Evolución función de costos

Función beta (5/5)

Original | Raíz | Polinomial | Cuadrática | Exponencial

Se define la función beta exponencial como

$$\beta_n = e^n$$

Se simulan 20.000 pasos de la cadena.

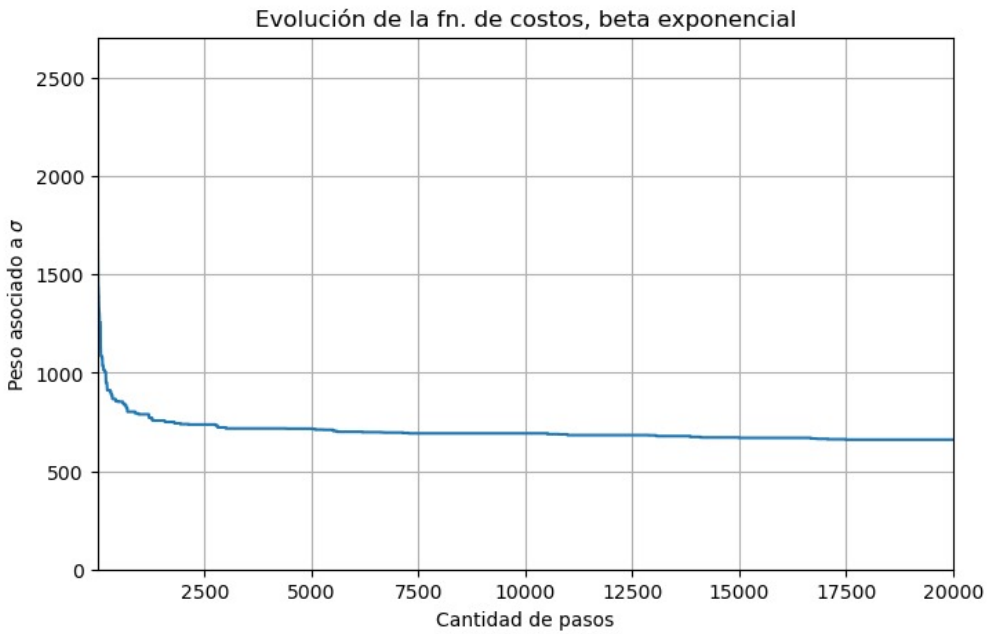

Figura 8: Evolución función de costos

Estado inicial simulated annealing

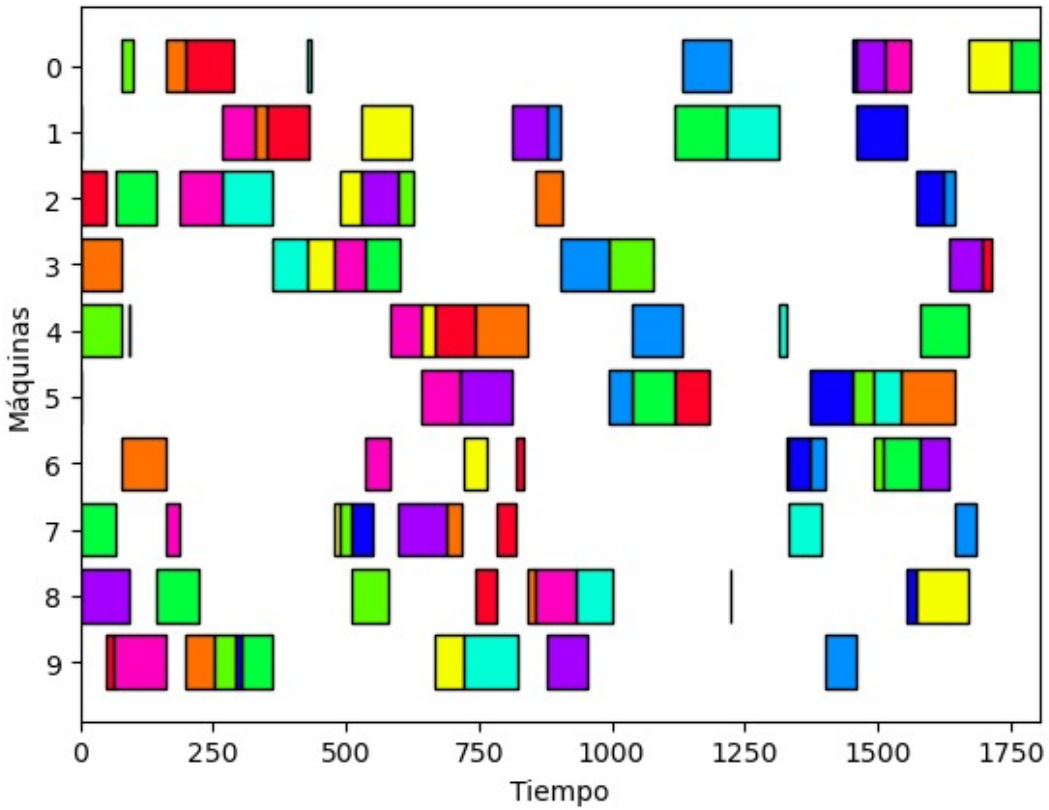

Figura 9: Original, raíz, polinomial, cuadrática y exponencial en estado inicial.

Estado final simulated annealing

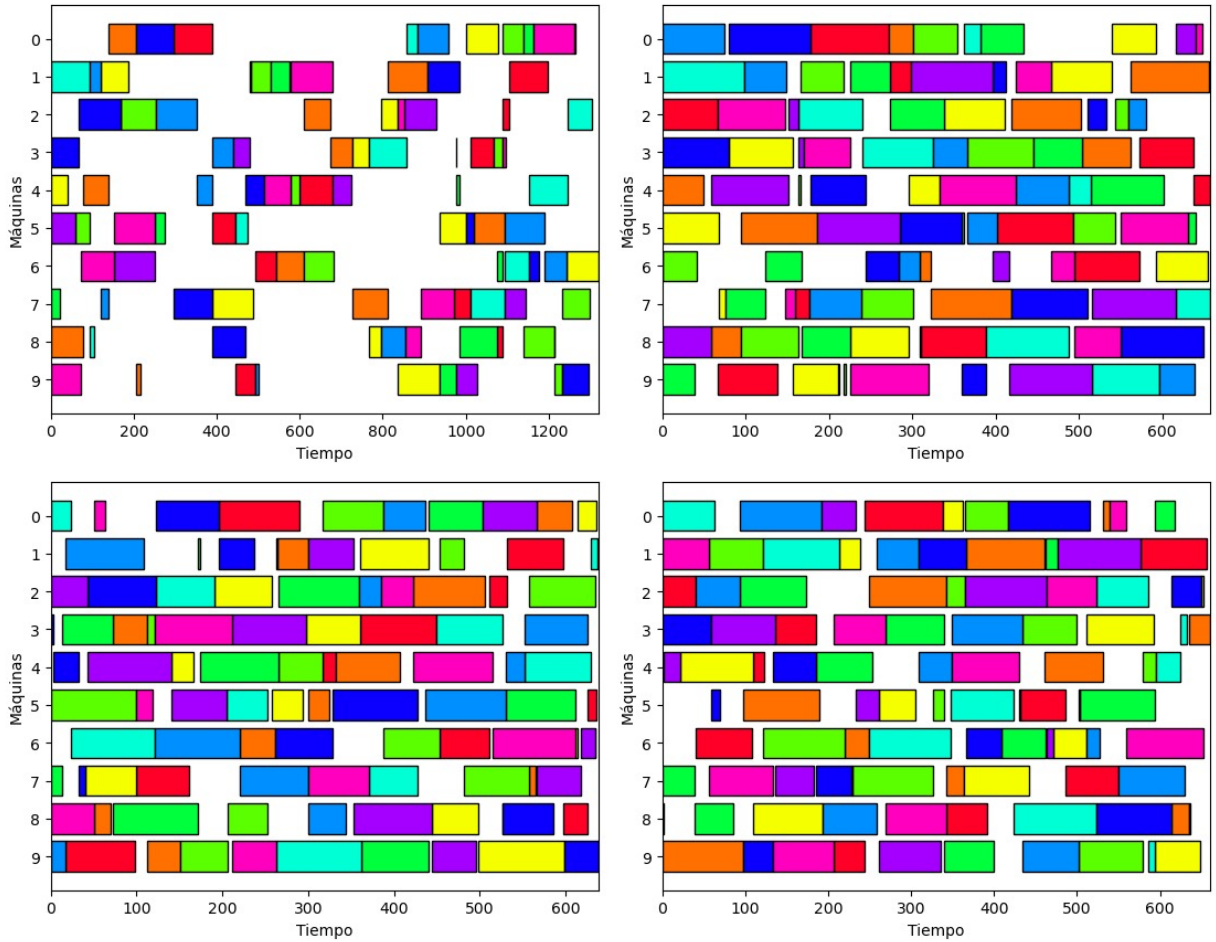

Figura 10: Estado final de beta original (izquierda superior), beta raiz (derecha superior), beta cuadrático (izquierda inferior) y beta exponencial (derecha inferior)

Estado final simulated annealing

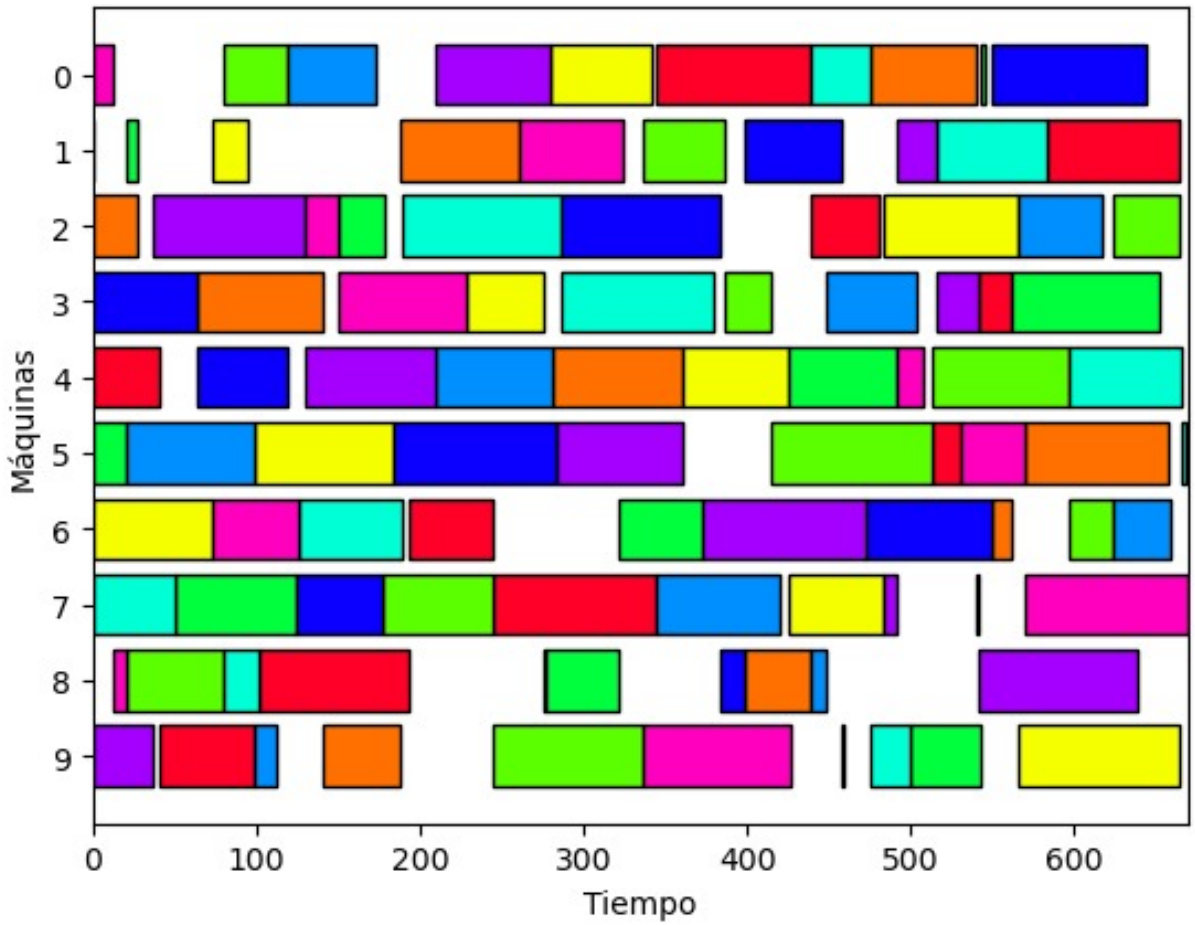

Figura 11: Estado final de beta polinomial

Costos asociados

Tabla 1: Costos de la cadena en ciertos instantes para cada beta.

Función beta	**Costo Final (seg.)**
Original	1.319
Raíz	657
Exponencial	638
Polinomial	669
Cuadrática	660

Casos particulares para caso polinomial

- A partir de este punto, fijamos la función beta como la polinomial, dado que esta es la que recorre de manera más completa la cadena de Markov.
- Ilustraremos lo anterior para tres casos particulares en donde tendremos 3 máquinas y 12 tareas:

(i) Con tiempos idénticos $t = 10$.

(ii) Con tiempos con diferencia muy grande entre $t = 10$ y $t = 50$.

(iii) Con diferencias aleatorias entre $t = 1$ y $t = 100$.

Tiempos idénticos

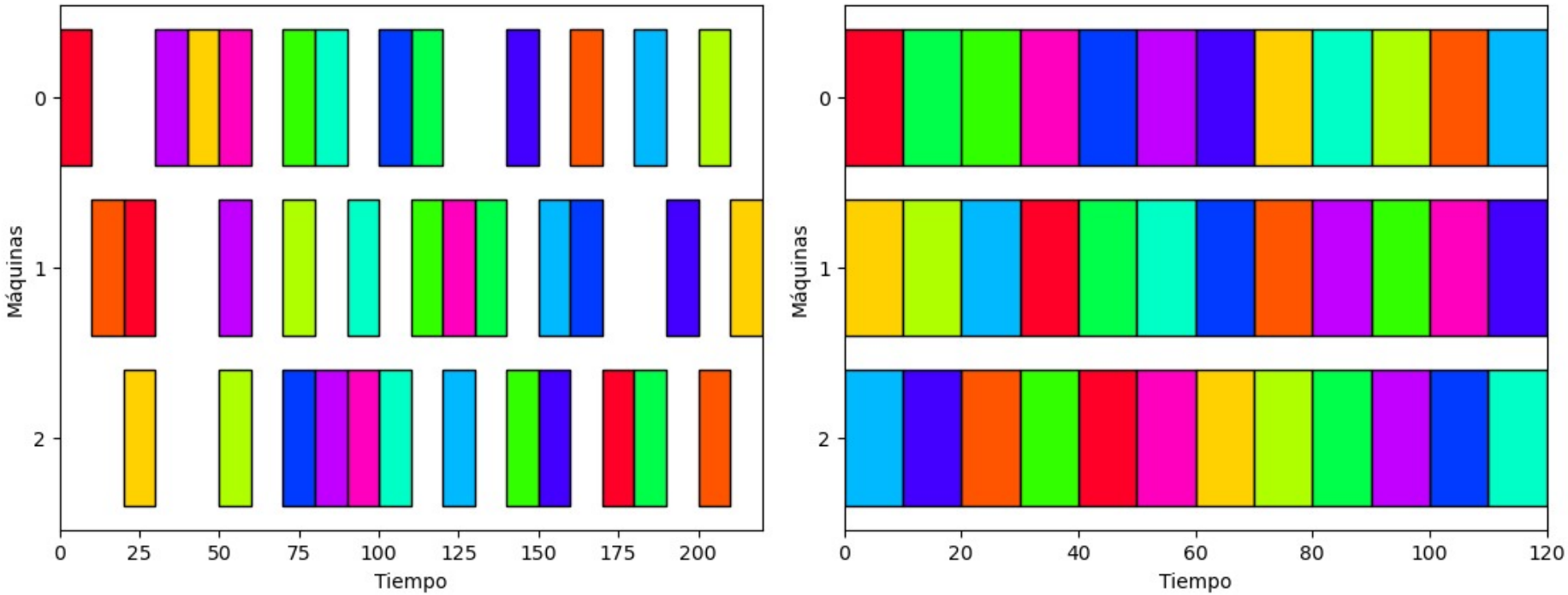

Figura 12: Estado inicial (izquierda) y estado final (derecha) para tiempos idénticos

Tiempos con gran diferencia

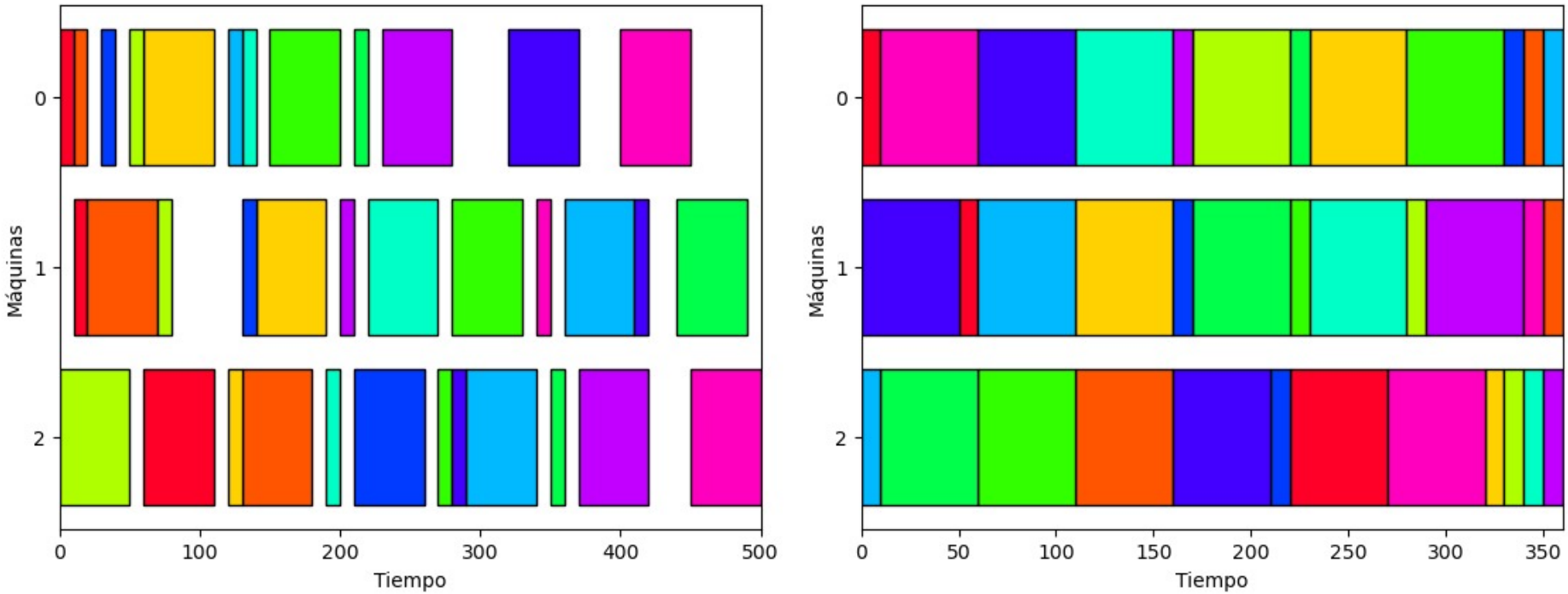

Figura 13: Estado inicial (izquierda) y estado final (derecha) para tiempos con gran diferencia

Tiempos aleatorios

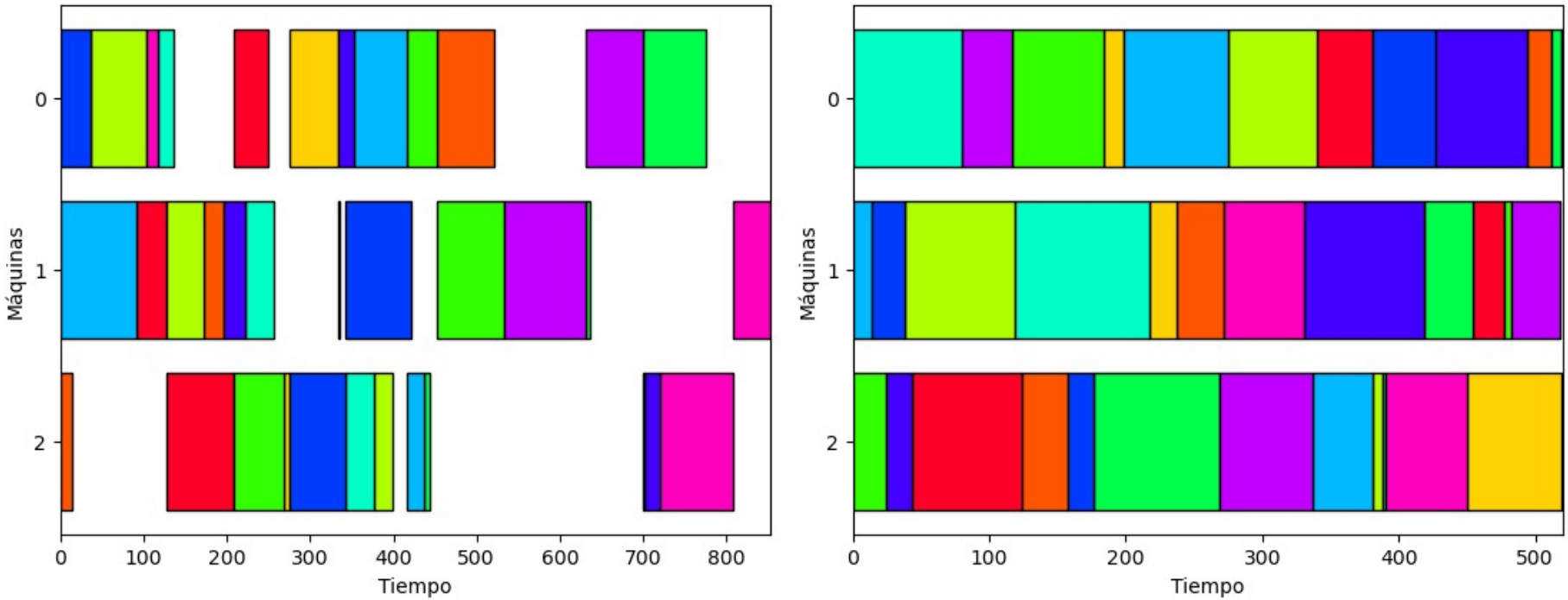

Figura 14: Estado inicial (izquierda) y estado final (derecha) para tiempos aleatorios

Costos asociados a cada ejemplo

Tabla 2: Costos de la cadena en los ejemplos para beta polinomial.

Función beta	**Costo inicial (seg.)**	**Costo Final (seg.)**
Tiempos idénticos	220	120
Tiempos con gran diferencia	500	360
Tiempos aleatorios	852	520

De todas maneras, esta tabla no es representativa en cuanto a comparación de costos, dado que, los órdenes de los tiempos son distintos entre los tres ejemplos.

Genetic Algorithm

A continuación, se implementará un nuevo método llamado *Genetic Algorithm* (GA) o Algoritmo Genético para abordar JSP. Este algoritmo está basado en la teoría de Darwin llamada la supervivencia del más fuerte. [Decoderz, 2019]

Se genera una población inicial aleatoria de 10 individuos (en este caso, horarios), dando lugar a la primera generación. Esta generación pasará por procesos como la reproducción, cruce y mutación para crear una nueva y mejor población para la siguiente generación.

Para esto, se consideran principalmente 100 generaciones, 5 máquinas y 5 tareas. [Omar et al., 2006]

Genetic Algorithm

- Primero, se crea *DG distance*, función que representa el número de operaciones diferentes entre dos horarios σ_1 y σ_2.

$$\textit{DG distance}((1,2,3,4),\ (1,2,4,3)) = 2.$$

- Luego, se crea *Adjacent Swapping*(AS) o intercambio de elementos adyacentes, para obtener todos los vecinos de un horario σ.

$$\textit{AS}((1,2,3,4)) = [(2,1,3,4),(3,2,1,4),(4,2,3,1),$$
$$(1,3,2,4),(1,4,3,2),(1,2,4,3)]$$

Genetic Algorithm

- La función de costos/energía de un horario es la misma que en SA, es decir, es el tiempo de finalización de la última tarea.
- La población inicial es de 10 horarios generados aleatoriamente.
- La selección de padres se hace escogiendo 2 horarios al azar de los 10 totales.

Genetic Algorithm

- *Crossover* o Cruce: entre los padres seleccionados, tiene como objetivo generar un nuevo horario (*child* o hijo), el cual tenga mejor peso que sus padres, para que sea un miembro de la nueva generación, reemplazando al peor horario (en términos de costos).
- *Mutation* o mutación: se escoge un padre de los dos, y se escoge un vecino que tenga menor costo que el padre, para que sea un miembro de la nueva generación, reemplazando al peor horario (en términos de costos).
- *Genetic Algorithm*: luego de 100 generaciones seleccionadas con *Crossover* y *Mutation* (mediante *Adjacent Swapping*), se escoge al mejor horario de la generación.

Evolución generaciones GA (1/6)

Evolución de los costos mediante Genetic Algorithm con una generación constante de 10 individuos y parámetro 4.

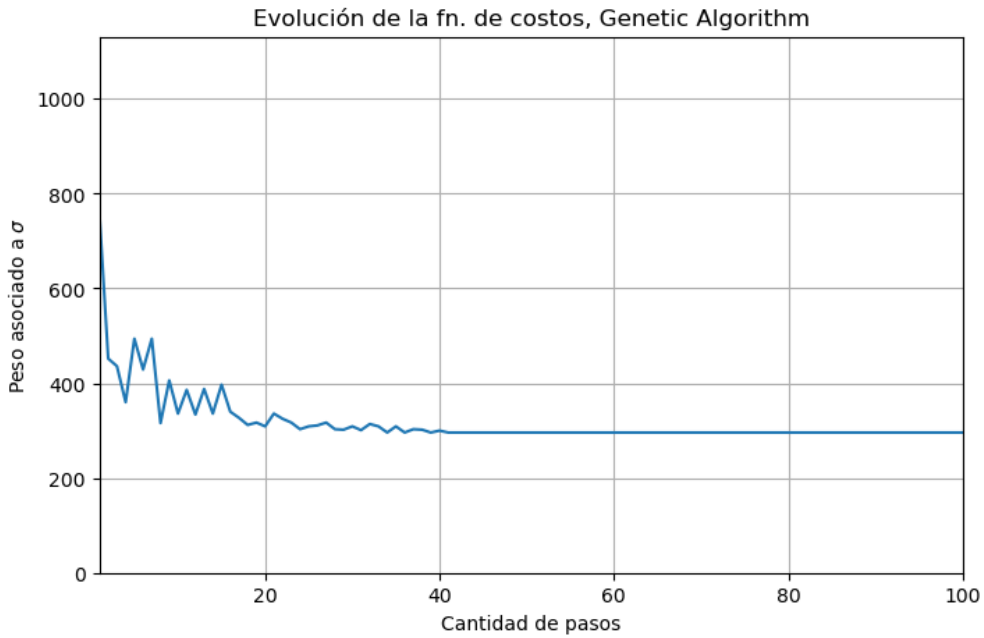

Figura 15: Evolución costos de los hijos.

Evolución generaciones GA (2/6)

$m, n = 5$ | **$m, n = 6$** | $m, n = 7$ | $m, n = 8$ | $m, n = 9$ | $m, n = 10$

Evolución de los costos mediante Genetic Algorithm con una generación constante de 10 individuos y parámetro 4.

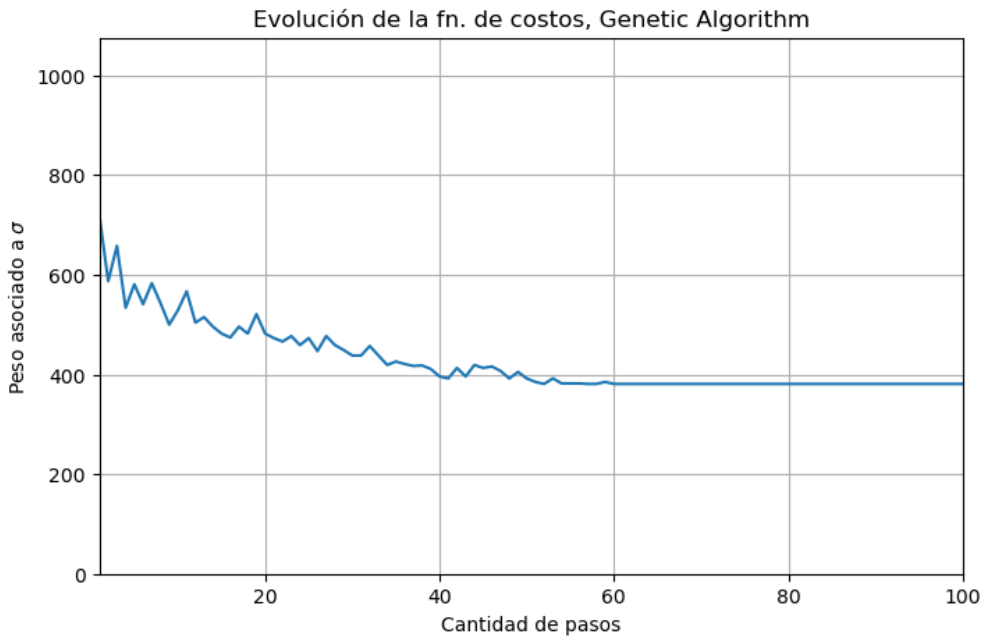

Figura 16: Evolución costos de los hijos.

Evolución generaciones GA (3/6)

$m, n = 5$ | $m, n = 6$ | **$m, n = 7$** | $m, n = 8$ | $m, n = 9$ | $m, n = 10$

Evolución de los costos mediante Genetic Algorithm con una generación constante de 10 individuos y parámetro 4.

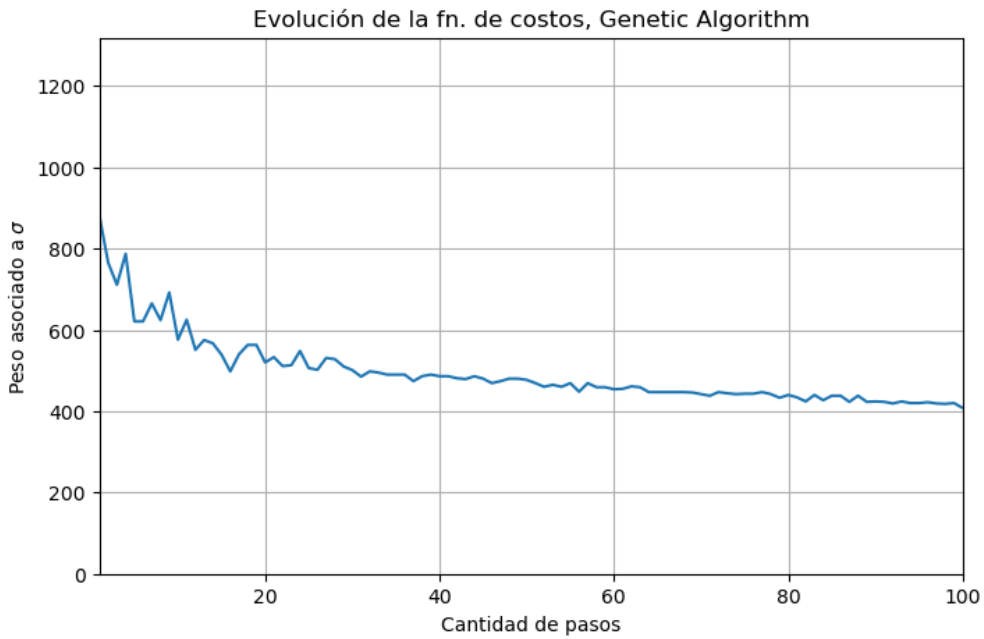

Figura 17: Evolución costos de los hijos.

Evolución generaciones GA (4/6)

Evolución de los costos mediante Genetic Algorithm con una generación constante de 10 individuos y parámetro 4.

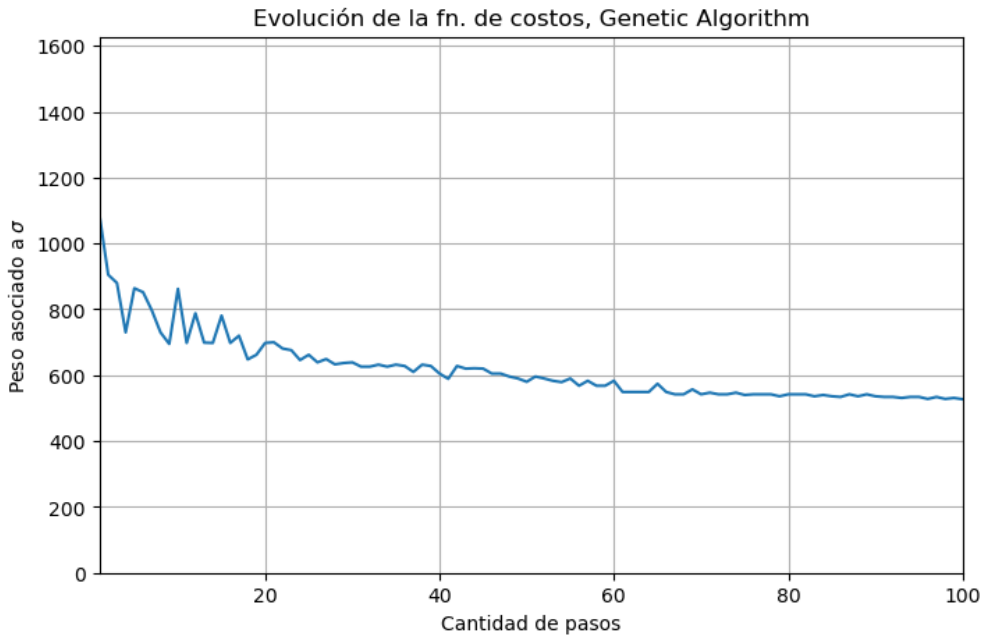

Figura 18: Evolución costos de los hijos.

Evolución generaciones GA (5/6)

Evolución de los costos mediante Genetic Algorithm con una generación constante de 10 individuos y parámetro 4.

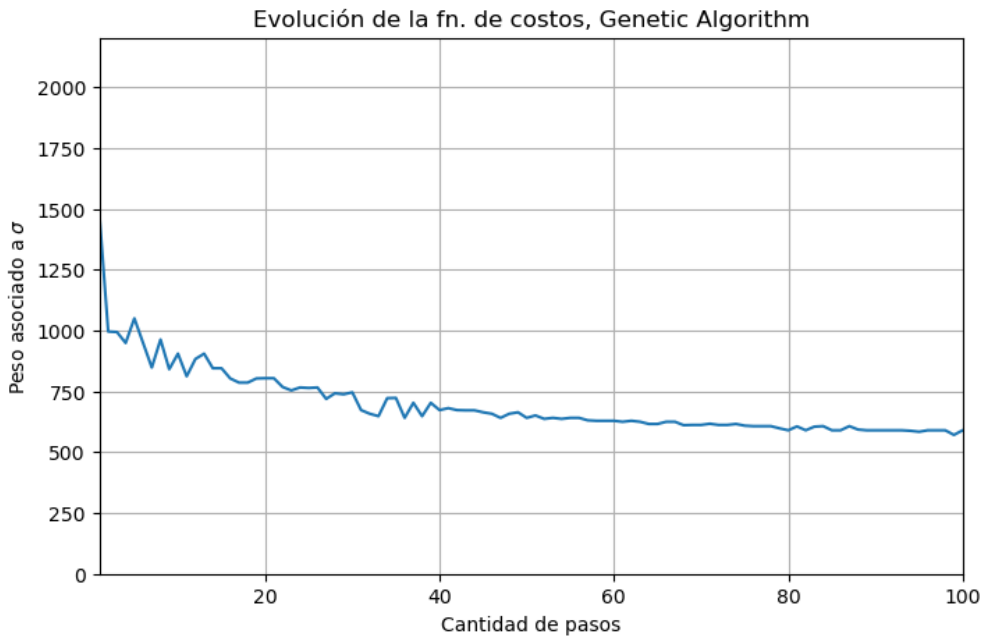

Figura 19: Evolución costos de los hijos.

Evolución generaciones GA (6/6)

Evolución de los costos mediante Genetic Algorithm con una generación constante de 10 individuos y parámetro 4.

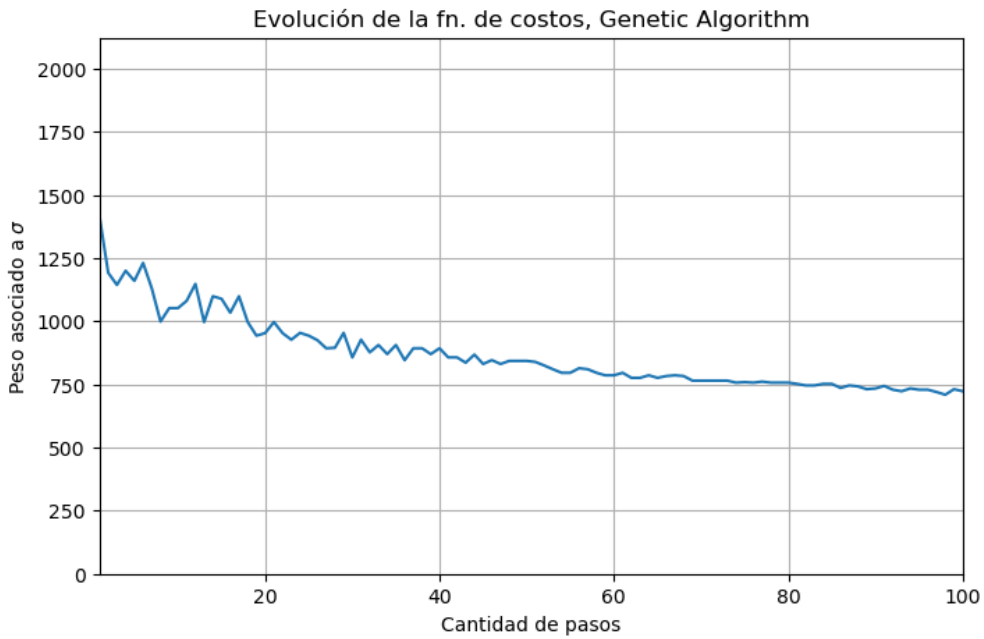

Figura 20: Evolución costos de los hijos.

Horarios finales

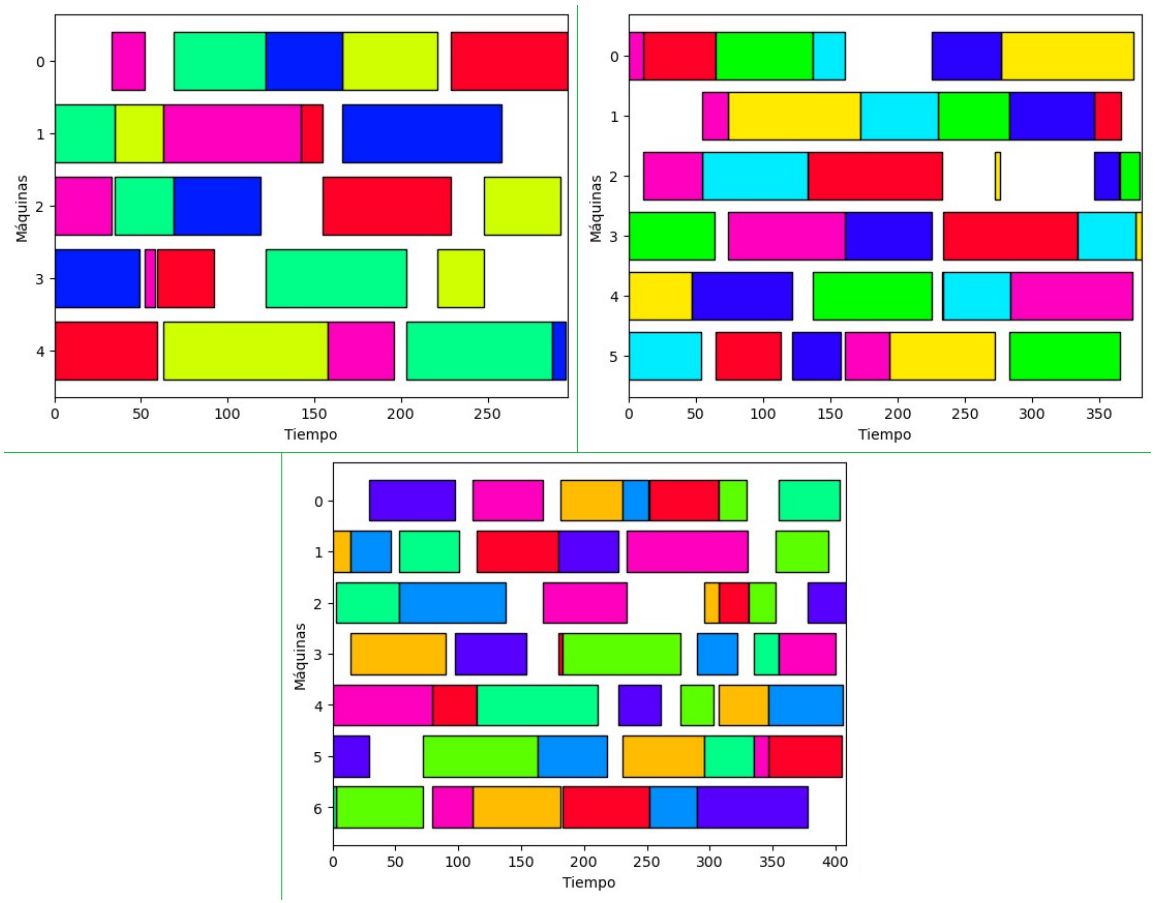

Figura 21: Horarios finales para cantidades de máquinas y tareas $m, n = 5, 6, 7$.

Horarios finales

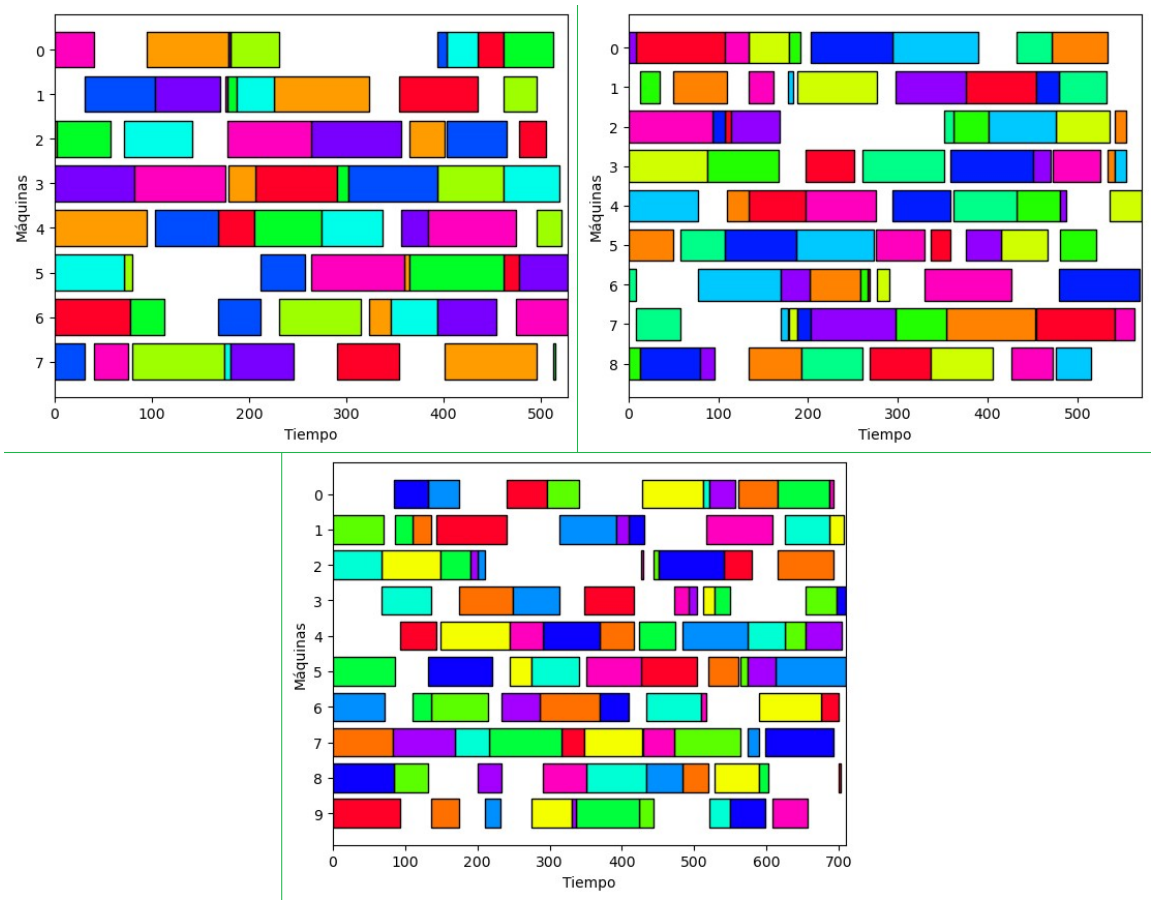

Figura 22: Horarios finales para cantidades de máquinas y tareas $m, n = 8, 9, 10$.

Costos asociados a distintos m, n con GA

Tabla 3: Promedios de 5 iteraciones de costo hijo del final y tiempo en ejecutar con GA.

Cantidad m, n	**Costo final (seg.)**	**Tiempo (seg.)**
5	271	4
6	347	13
7	432	37
8	510	94
9	596	219
10	704	475

Costos asociados a distintos m, n con SA

Tabla 4: Promedios de 5 iteraciones de costo final y tiempo en ejecutar con SA.

Cantidad m, n	**Costo final (seg.)**	**Tiempo (seg.)**
5	302	7
6	365	11
7	446	16
8	531	24
9	605	35
10	694	48

Conclusión

Se utilizaron los métodos simulated annealing y genetic algorithm en el problema estudiado, los cuales tienen como objetivo minimizar el tiempo de los horarios, lo cual permitió abordar un problema muy demandante en términos de memoria de una manera más fácil. Se estudiaron varias funciones beta para SA, concluyendo que la mejor de todas en cuanto a salir del mínimo local, es la función polinomial.

Por otro lado, se utilizó GA para hacer el cruce o mutación de los horarios de una generación, permitiendo obtener mejores resultados para una cantidad baja de máquinas y tareas que SA (entre 5 y 9). Esto ocurre debido a que GA rompe la característica de búsqueda local, al tener varios individuos.

Referencias I

Decoderz (2019).

Theory and applications of genetic algorithms: Darwin's theory of evolutions.

Kolonko, M. (1999).

Some new results on simulated annealing applied to the job shop scheduling problem.

Omar, M., Baharum, A., and Hasan, Y. (2006).

A job-shop scheduling problem (jssp) using genetic algorithm (ga).

Piroozfard, H., Wong, K., and Hassan, A. (2016).

Horario de 3 máquinas y 4 tareas.

¡Muchas gracias por su atención!