DESDE 2013
https://repository.uaeh.edu.mx/revistas/index.php/icbi/issue/archive
Pädi Boletín Científico de Ciencias Básicas e Ingenierías del ICBI
ISSN: 2007-6363

Publicación Semestral Pädi Vol. 14 No. Especial 3 (2026) 284-290

Estudio paramétrico de un algoritmo genético con redes de Petri
Parametric study of a genetic algorithm with Petri nets

G. E. Heredia-Medina [a], J. Medina-Marín [a,*], J. C. Seck-Tuoh-Mora [a], N. Hernández-Romero [a], I. Barragán-Vite [a]

[a] *Área Académica de Ingeniería y Arquitectura, Universidad Autónoma del Estado de Hidalgo, 42184, Pachuca, Hidalgo, México.*

Resumen

El problema de programación de tareas tipo *job-shop* (JSSP) es NP-difícil y requiere métodos aproximados. Este trabajo evalúa un algoritmo genético acoplado a una red de Petri temporizada adaptativa, empleada como evaluador exacto del *makespan*. Mediante un estudio paramétrico factorial completo 3×3 se combinaron tres tamaños de población (50, 75 y 100) y tres umbrales de reinicio por estancamiento (10, 15 y 20), sobre 18 instancias estándar de dimensión 10×10, con diez corridas independientes por configuración (1620 corridas). La calidad se midió mediante la desviación porcentual relativa (RPD) respecto a la mejor solución conocida. El tamaño de población fue el principal factor regulador de la calidad, mientras que el umbral de reinicio tuvo un efecto marginal; la mejor configuración agregada fue 100×15. El tiempo escaló de forma aproximadamente lineal con la población, lo que sugiere priorizar su incremento sobre el ajuste del umbral de reinicio.

Palabras Clave: Programación job-shop, Algoritmo genético, Redes de Petri, Makespan, Estudio paramétrico.

Abstract

The job-shop scheduling problem (JSSP) is NP-hard and requires approximate methods for non-trivial instances. This work evaluates a genetic algorithm coupled with an adaptive timed place Petri net, used as an exact evaluator of the makespan of each operation sequence. A full 3×3 factorial parametric study was conducted, combining three population sizes (50, 75 and 100) and three stagnation-restart thresholds (10, 15 and 20), over 18 standard 10×10 benchmark instances, with ten independent runs per configuration (1620 runs). Solution quality was measured through the relative percentage deviation (RPD) from the best known solution. Population size was the main factor governing quality, whereas the restart threshold had a marginal effect; the best aggregate configuration was 100×15 and the closest approximation reached an RPD of 1.80%. Computational time scaled approximately linearly with population size, which suggests prioritizing population size over fine-tuning the restart threshold in the GA+ATPPN method.

Keywords: Job-shop scheduling, Genetic algorithm, Petri nets, Makespan, Parametric study

1. Introducción

El problema de programación de tareas tipo *job-shop* (*Job-Shop Scheduling Problem,* JSSP) es uno de los problemas de optimización combinatoria más estudiados en la ingeniería de manufactura, por su capacidad para representar de manera realista los sistemas de producción discreta en los que distintos productos se procesan sobre un mismo conjunto de máquinas (Jain y Meeran, 1999). En su formulación clásica, el problema consiste en determinar la secuencia de procesamiento de un conjunto de trabajos sobre un conjunto de máquinas, respetando las restricciones de precedencia tecnológica y de capacidad, con el fin de optimizar un criterio de desempeño; el más adoptado es la minimización del *makespan*, es decir, el instante de finalización de la última operación del sistema (Pinedo, 2008). El JSSP clásico mantiene su vigencia como problema de referencia para el desarrollo metodológico y la evaluación comparativa de algoritmos, y constituye el caso fundamental sobre el que se construyen extensiones como el *flexible job-shop* (Xiong et al., 2022; Dauzère-Pérès et al., 2024).

La dificultad del JSSP tiene fundamentos teóricos precisos. Garey et al. (1976) demostraron que el problema con tres o más máquinas pertenece a la clase NP-difícil, lo que implica que, salvo que $P = NP$, no se espera la existencia de un algoritmo capaz de resolverlo al óptimo en tiempo polinomial. Esta dificultad se manifiesta en el tamaño del espacio de soluciones, acotado por $(n!)^m$ para una instancia de n trabajos y m máquinas, lo que vuelve impracticable la enumeración exhaustiva incluso para instancias moderadas (Jain y Meeran,

*Autor para la correspondencia: jmedina@uaeh.edu.mx
Correo electrónico: he526682@uaeh.edu.mx (Gonzalo Eduardo Heredia-Medina), jmedina@uaeh.edu.mx (Joselito Medina-Marin), jseck@uaeh.edu.mx (Juan Carlos Seck-Tuoh-Mora), nhromero@uaeh.edu.mx (Norberto Hernández Romero), irvingb @uaeh.edu.mx (Irving Barragán-Vite).

Historial del manuscrito: recibido el 19/06/2026, última versión-revisada recibida el 30/07/2026, aceptado el 18/09/2026, publicado el 06/11/2026. **DOI:** https://doi.org/10.29057/icbi.v14iEspecial3.18131

1999). La evidencia empírica lo confirma: la instancia ft10, de dimensión 10×10, propuesta en 1963, permaneció sin resolución comprobada al óptimo durante 26 años, hasta el trabajo de Carlier y Pinson (1989).

Esta condición ha motivado el desarrollo de métodos aproximados. Frente a los métodos exactos, como los esquemas de bifurcación y acotamiento (Brucker et al., 1994) o, más recientemente, la programación con restricciones (Da Col y Teppan, 2022), que garantizan la optimalidad a costa de un esfuerzo computacional creciente, las metaheurísticas renuncian a esa garantía a cambio de soluciones de buena calidad en tiempos manejables (Pinedo, 2008). Entre ellas, los algoritmos genéticos se han consolidado a lo largo de tres décadas como una de las estrategias más utilizadas para el JSSP, gracias a su naturaleza poblacional y a su capacidad para combinar exploración y explotación sobre espacios de búsqueda extensos (Cheng et al., 1996). Una tendencia consolidada es, además, su hibridación con otras estrategias, incluida la incorporación de evaluadores especializados dentro del ciclo de búsqueda (Cheng et al., 1999).

En esta última dirección, las redes de Petri ofrecen un formalismo matemático y gráfico idóneo para modelar sistemas de eventos discretos como los de manufactura, al representar de manera natural la concurrencia, la compartición de recursos y la sincronización entre operaciones (Murata, 1989; Zhou y Venkatesh, 1999). Sobre esta base, Medina-Marín et al. (2014) proponen una red de Petri temporizada adaptativa (ATPPN) capaz de calcular de manera exacta el *makespan* asociado a una secuencia de operaciones dada, lo que la habilita como función de evaluación dentro de una metaheurística. El acoplamiento de un algoritmo genético con dicho evaluador, siguiendo el esquema de algoritmo genético de Phanden et al. (2012), constituye el método estudiado en este trabajo.

La combinación de redes de Petri con metaheurísticas ha sido objeto de investigación reciente, con enfoques diversos. Lin et al. (2025) integran redes de Petri temporizadas con un algoritmo genético mejorado para la programación dinámica en talleres flexibles híbridos; He et al. (2026) abordan la programación de sistemas de manufactura flexible mediante redes de Petri temporizadas por lugar y grafos de alcanzabilidad base; Huang et al. (2022) combinan redes de Petri temporizadas con búsqueda A* para la manufactura celular robotizada; y Lefebvre y Basile (2021) y Mušič (2021) emplean redes de Petri temporizadas junto con esquemas de búsqueda en árbol y búsqueda local. En el ámbito de los algoritmos genéticos, Hu et al. (2023) proponen una variante con búsqueda de vecindario dinámico para el JSSP. La mayoría de estos trabajos se orientan a variantes flexibles o dinámicas del problema, utilizan la red de Petri como estructura de análisis o supervisión de la búsqueda, o incorporan mecanismos de hibridación adicionales. A diferencia de ellos, el presente trabajo no propone un nuevo algoritmo ni una nueva hibridación, sino que emplea la ATPPN estrictamente como evaluador exacto del *makespan* dentro de un algoritmo genético estándar, y aporta una caracterización empírica del acoplamiento GA+ATPPN: un estudio paramétrico factorial que determina cómo el tamaño de población y el umbral de reinicio gobiernan el compromiso entre calidad de solución y costo computacional sobre el JSSP clásico, aspecto que no ha sido documentado de forma específica para este acoplamiento.

2. Materiales y métodos

2.1. Definición del problema job-shop

El problema de programación de tareas tipo *job-shop* (*Job-Shop Scheduling Problem*, JSSP) consiste en determinar la secuencia de procesamiento de un conjunto finito de trabajos sobre un conjunto finito de máquinas compartidas, sujeto a restricciones de precedencia y de capacidad (Pinedo, 2008). Sea $J = \{J_1, J_2, \dots, J_n\}$ el conjunto de n trabajos y $M = \{M_1, M_2, \dots, M_m\}$ el conjunto de m máquinas. Cada trabajo J_i está formado por una secuencia ordenada de operaciones; en la formulación clásica, cada trabajo consta de exactamente m operaciones, una por máquina, por lo que el número total de operaciones del sistema es $N = n \times m$ (Xiong et al., 2022).

Cada operación se denota $O_{i,j,k}$ y corresponde a la *j*-ésima operación del trabajo J_i que debe procesarse en la máquina M_k durante un tiempo de procesamiento $p_{i,j,k}$, conocido y constante (Medina-Marín et al., 2014). Las variables de decisión son los tiempos de inicio $s_{i,j,k}$; el tiempo de finalización de cada operación es:

$$C_{i,j,k} = s_{i,j,k} + p_{i,j,k} \tag{1}$$

y el tiempo de finalización del trabajo J_i, denotado C_i, corresponde al de su última operación. El objetivo es minimizar el *makespan*, definido como el instante en que concluye la última operación del sistema:

$$\min C_{\max}, con\ C_{\max} = \max_{1 \le i \le n} \{C_i\} \tag{2}$$

La formulación está sujeta a las siguientes restricciones (Medina-Marín et al., 2014; Phanden et al., 2012): las operaciones de un mismo trabajo deben respetar su secuencia predefinida; cada trabajo visita cada máquina exactamente una vez; cada máquina procesa una sola operación a la vez; y no existen precedencias entre operaciones de trabajos distintos. La restricción de precedencia intra-trabajo se expresa como:

$$s_{i,j,k} + p_{i,j,k} \le s_{i,j+1,k'} \tag{3}$$

donde k y k' son las máquinas asignadas a las operaciones consecutivas $O_{i,j}$ y $O_{i,j+1}$. Adicionalmente, se asume que todos los trabajos están disponibles en el instante inicial, que las operaciones no pueden interrumpirse una vez iniciadas (*non-preemption*), que los tiempos de preparación se incluyen en el de procesamiento y que las máquinas operan sin fallas durante todo el horizonte (Phanden et al., 2012).

El número de secuencias factibles crece de manera factorial, con una cota del orden de $(n!)^m$, lo que vuelve inviable la enumeración exhaustiva incluso para instancias moderadas. Garey et al. (1976) demostraron que el JSSP con tres o más máquinas pertenece a la clase NP-difícil, lo que justifica el empleo de métodos aproximados, como las metaheurísticas (Medina-Marín et al., 2014).

2.2. Evaluación del makespan mediante una red de Petri temporizada adaptativa

La evaluación del *makespan* de cada secuencia de operaciones se realiza mediante el modelo de red de Petri temporizada adaptativa (*Adaptive Timed Place Petri Net*, ATPPN) propuesto por Medina-Marín et al. (2014), capaz de calcular de manera exacta el *makespan* asociado a una secuencia dada a partir de las herramientas analíticas de la teoría de redes de Petri. Este modelo constituye la función de evaluación que se acopla al ciclo evolutivo del algoritmo genético.

Una red de Petri ordinaria se define como la 5-tupla (P, T, F, W, M_0), donde P es el conjunto finito de lugares, T el de transiciones, F el de arcos, W la función de pesos y M_0 el marcado inicial (Murata, 1989). Su estructura se codifica de forma compacta mediante la matriz de incidencia A, cuyos elementos se obtienen como:

$$a_{i,j} = a_{i,j}^{+} - a_{i,j}^{-} \tag{4}$$

donde $a_{i,j}^{+}$ y $a_{i,j}^{-}$ son, respectivamente, los pesos del arco de salida y de entrada que vinculan la transición t_i con el lugar p_j (Medina-Marín et al., 2014). La evolución dinámica del sistema se describe mediante la ecuación de estado:

$$M_k = M_{k-1} + A^{\mathrm{T}} U_k \tag{5}$$

donde U_k es el vector característico de disparo del paso k, con un único elemento no nulo igual a uno, correspondiente a la transición disparada, y M_k es el marcado resultante (Medina-Marín et al., 2014).

Sobre esta base, una red de Petri temporizada por lugar (*Timed Place Petri Net*, TPPN) extiende la definición a la 6-tupla (P, T, F, W, M_0, D), donde D es el vector de retardos temporales asociado a cada lugar; las transiciones de salida de un lugar se habilitan solo una vez transcurrido dicho retardo, que en el modelo del JSSP corresponde al tiempo de procesamiento de la operación (Medina-Marín et al., 2014). La ATPPN añade un séptimo elemento, Fd, conjunto de arcos dinámicos con $F \cap Fd = \emptyset$, que se incorporan a la red en función del orden de operaciones definido por una secuencia particular; a estos arcos se asigna peso dos, lo que impide que una operación inicie en una máquina hasta que su predecesora en esa misma máquina haya finalizado, garantizando así la exclusión mutua en el uso de los recursos compartidos (Medina-Marín et al., 2014).

A partir de la ATPPN, el *makespan* se calcula mediante un procedimiento iterativo que, en cada paso, identifica las transiciones habilitadas y determina su tiempo de finalización como el máximo entre el instante en que el trabajo queda listo y el instante en que la máquina requerida está disponible, más el tiempo de procesamiento de la operación; el marcado se actualiza con la ecuación (5) hasta procesar todas las operaciones, y el *makespan* corresponde al mayor de los tiempos de finalización registrados (Medina-Marín et al., 2014). Este procedimiento se invoca en cada evaluación de aptitud del algoritmo genético, por lo que su implementación replica de manera fiel el algoritmo original para reducir el costo computacional.

2.3. Algoritmo genético acoplado a la ATPPN

Los algoritmos genéticos son métodos de optimización inspirados en la selección natural, que operan sobre una población de soluciones candidatas modificadas iterativamente mediante operadores de selección, cruce y mutación para mejorar progresivamente una medida de aptitud (Holland, 1992). El algoritmo de este trabajo reproduce fielmente el esquema propuesto por Phanden et al. (2012), adaptado al JSSP clásico y modificado para integrar la ATPPN como función de evaluación, lo que constituye el acoplamiento metodológico central.

Cada cromosoma se codifica mediante la representación basada en la secuencia de operaciones por trabajo (os_jobs): un vector de longitud $N = n \times m$ que contiene una permutación con repeticiones de los identificadores de trabajo, en la que cada trabajo J_i aparece exactamente *m* veces y su *j*-ésima aparición representa su *j*-ésima operación (Escamilla-Serna et al., 2021). Esta representación garantiza que toda permutación con la frecuencia correcta de aparición sea factible, lo que reduce la necesidad de reparación y acota el espacio de búsqueda (Escamilla-Serna et al., 2021). La población inicial se genera de forma aleatoria balanceada. La aptitud de cada cromosoma se obtiene en dos pasos: el vector os_jobs se traduce a una secuencia de operaciones únicas y esta se evalúa con la ATPPN, cuyo *makespan* se utiliza directamente como medida de aptitud; al tratarse de un problema de minimización, valores menores corresponden a mejores soluciones (Medina-Marín et al., 2014).

El ciclo evolutivo emplea cuatro operadores (Phanden et al., 2012). La selección combina elitism; el 10% de los individuos con menor *makespan* pasa intacto a la siguiente generación, con muestreo universal estocástico y clasificación lineal (LR-SUS), basado en Baker (1985), en el que la probabilidad de selección de cada individuo es inversamente proporcional a su *makespan*:

$$\mathrm{prob}_{\mathrm{select}}(n) = \frac{1/C_{\max}(n)}{\sum_{i=1}^{n_{\mathrm{sol}}} 1/C_{\max}(i)} \tag{6}$$

El cruce de dos puntos se aplica con probabilidad $p_c = 0.8$: el segmento central del cromosoma hijo proviene de un progenitor y los extremos del otro. Dado que este operador puede alterar la frecuencia de aparición de los trabajos, un operador de reparación restablece la validez mediante intercambios donador-receptor. La mutación por intercambio recíproco se aplica con probabilidad $p_m = 0.2$ permutando dos posiciones de distinto valor, operación que preserva la factibilidad del cromosoma.

Para evitar la convergencia prematura, el algoritmo incorpora un mecanismo de reinicio por estancamiento, propuesto por Ruiz y Maroto (2006). Un contador registra las iteraciones consecutivas sin mejora del *makespan* mínimo; al alcanzar un umbral it_reinicio, la población se ordena, se preserva el 20% superior, y el 80% restante se regenera en partes iguales mediante mutación de los individuos preservados y mediante generación aleatoria. El criterio de paro es un número máximo de iteraciones igual a $n \times m$ (Phanden et al., 2012).

El tamaño de la población, *n_sol*, y el umbral de reinicio, *it_reinicio*, constituyen las dos variables independientes del estudio paramétrico de este trabajo. La población regula el equilibrio entre diversidad y costo computacional, mientras que el umbral regula la frecuencia de diversificación frente a

la profundidad de explotación. Los niveles evaluados son $n_sol \in \{50,75,100\}$ e $it_reinicio \in \{10,15,20\}$; el resto de los parámetros se mantienen constantes según Phanden et al. (2012).

2.4. Diseño experimental

La evaluación se realizó sobre 18 instancias estándar del JSSP de dimensión 10×10 (10 trabajos, 10 máquinas, 100 operaciones), obtenidas del repositorio público JSPLIB junto con sus mejores soluciones conocidas (Tamy0612, n.d.). La elección de instancias 10×10 obedece a tres razones: su complejidad combinatoria es suficientemente alta para que la enumeración exhaustiva resulte inviable (Medina-Marín et al., 2014); el conjunto es heterogéneo e incluye problemas reconocidos como difíciles; y, por ser instancias estándar, permiten la comparación directa con resultados publicados. Las instancias pertenecen a cuatro familias de referencia: abz, propuesta por Adams et al. (1988); ft10, por Muth y Thompson (1963); la, por Lawrence (1984); y orb, por Applegate y Cook (1991). La Tabla 1 resume las instancias y sus valores de mejor solución conocida (*Best Known Solution*, BKS).

Tabla 1. Instancias del JSSP utilizadas en la evaluación experimental.

Instancia	Origen	BKS
abz5	Adams et al. (1988)	1234
abz6	Adams et al. (1988)	943
ft10	Muth y Thompson (1963)	930
la16	Lawrence (1984)	945
la17	Lawrence (1984)	784
la18	Lawrence (1984)	848
la19	Lawrence (1984)	842
la20	Lawrence (1984)	902
orb01	Applegate y Cook (1991)	1059
orb02	Applegate y Cook (1991)	888
orb03	Applegate y Cook (1991)	1005
orb04	Applegate y Cook (1991)	1005
orb05	Applegate y Cook (1991)	887
orb06	Applegate y Cook (1991)	1010
orb07	Applegate y Cook (1991)	397
orb08	Applegate y Cook (1991)	899
orb09	Applegate y Cook (1991)	934
orb10	Applegate y Cook (1991)	944

Nota: Dimensión 10×10, 100 operaciones. BKS obtenidas del repositorio JSPLIB (Tamy0612, n.d.).

La implementación se desarrolló en MATLAB R2018a y se ejecutó en un equipo con procesador Intel Xeon W de 24 núcleos a 2.7 GHz y 192 GB de memoria RAM, aprovechando el *Parallel Computing Toolbox* para ejecutar en paralelo las corridas independientes.

La calidad de las soluciones se cuantificó mediante la desviación porcentual relativa (*Relative Percentage Deviation*, RPD) respecto a la BKS, métrica de uso extendido en programación de tareas (Escamilla-Serna et al., 2021):

$$\mathrm{RPD} = \left(\frac{C_{\max}^{\mathrm{obt}} - \mathrm{BKS}}{\mathrm{BKS}}\right) \times 100\% \qquad (7)$$

donde C_{max}^{obt} es el *makespan* obtenido y BKS el mejor valor conocido de la instancia; un RPD igual a cero indica que se alcanzó el óptimo conocido y los valores positivos miden la distancia porcentual respecto a este. Para cada par configuración-instancia se registraron cuatro variables de respuesta sobre diez corridas independientes: el mejor *makespan* (mínimo de las diez corridas), su RPD asociado, la desviación estándar muestral de los diez *makespan*, como medida de consistencia, y el tiempo de cómputo promedio en segundos.

El estudio adoptó un diseño factorial completo 3×3 que combinó los dos parámetros del algoritmo, tamaño de población $n_sol \in \{50,75,100\}$ y umbral de reinicio $it_reinicio \in \{10,15,20\}$, dando lugar a nueve configuraciones (C1–C9). Cada configuración se evaluó sobre las 18 instancias con diez corridas independientes por par, lo que asciende a $9 \times 18 \times 10 = 1620$ corridas.

3. Resultados

Ninguna de las 18 instancias fue resuelta al óptimo conocido por ninguna de las nueve configuraciones. El mejor *makespan* de cada instancia se obtuvo con población de 100 individuos en 13 casos, con población de 75 en 5 casos y en ningún caso con población de 50. El RPD del mejor makespan, considerando todas las combinaciones (instancia, configuración), se distribuyó entre 1.80% (abz6, configuración 75×10) y 22.08% (orb06, 50×15), con media de 10.65%; al retener únicamente la mejor configuración de cada instancia, el rango se estrecha de 1.80% (abz6) a 14.57% (orb08), lo que evidencia que la configuración paramétrica influye de manera no despreciable en la calidad final. El RPD promedio sobre las diez corridas tuvo una media global de 15.91% (rango 8.05%–27.79%), y la desviación estándar del *makespan* una media de 31.98 unidades (rango 10.42–64.39). La Tabla 2 resume, para cada instancia, el mejor *makespan*, su RPD y la configuración con que se obtuvo.

Tabla 2. Mejor *makespan* por instancia, RPD respecto al BKS y configuración (Población×Umbral) con que se alcanzó.

Instancia	BKS	Mejor *Makespan*	RPD (%)	Config.
abz5	1234	1291	4.62	100×20
abz6	943	960	1.80	75×10
ft10	930	1039	11.72	100×15

la16	945	979	3.60	100×20
la17	784	817	4.21	100×20
la18	848	882	4.01	100×10
la19	842	889	5.58	100×15
la20	902	925	2.55	75×10
orb01	1059	1211	14.35	100×10
orb02	888	942	6.08	75×15
orb03	1005	1128	12.24	100×15
orb04	1005	1089	8.36	100×20
orb05	887	940	5.98	100×15
orb06	1010	1147	13.56	75×10
orb07	397	419	5.54	100×10
orb08	899	1030	14.57	100×20
orb09	934	1009	8.03	75×20
orb10	944	1022	8.26	100×15

3.1. Efecto del tamaño de población

El tamaño de población se comportó como el principal factor regulador de la calidad de solución, con una reducción monótona del RPD al aumentar la población (Tabla 3). El RPD del mejor *makespan* disminuyó de 11.70% a 9.62% al pasar de 50 a 100 individuos, y el RPD promedio de 17.44% a 14.79%. El patrón se confirma instancia por instancia: 13 de 18 instancias alcanzaron su mejor RPD del mejor *makespan* con población de 100 y ninguna con población de 50. Este beneficio en calidad debe ponderarse contra el costo computacional, que crece de forma aproximadamente lineal con la población: las razones de tiempo $T(75)/T(50) = 1.478$ y $T(100)/T(50) = 2.000$ coinciden con el escalamiento teórico esperado si el costo está dominado por el número de evaluaciones de aptitud.

Tabla 3. Efecto agregado del tamaño de población (promedio sobre 18 instancias × 3 umbrales).

Población	RPD mejor (%)	RPD prom. (%)	Tiempo medio (s)
50	11.70	17.44	206.92
75	10.64	15.51	305.87
100	9.62	14.79	413.88

3.2. Efecto del umbral de reinicio

El umbral de reinicio tuvo un efecto marginal sobre la calidad y sin tendencia monótona (Tabla 4): la diferencia entre el mejor y el peor nivel fue de apenas 0.58 puntos porcentuales en el RPD del mejor *makespan* y de 0.28 en el RPD promedio. Su incidencia sobre el tiempo de cómputo también fue marginal (variación inferior al 2.4%). El conteo por instancia no reveló un nivel dominante: el mejor RPD del mejor *makespan* se repartió de forma uniforme, con seis instancias por umbral. La configuración agregada de mejor desempeño bajo ambas métricas fue 100×15 (9.18% de RPD del mejor *makespan* y 14.73% de RPD promedio), sin constituir un mínimo aislado, pues las configuraciones vecinas con población 100 mostraron valores muy cercanos.

Tabla 4. Efecto agregado del umbral de reinicio (promedio sobre 18 instancias × 3 poblaciones).

Umbral	RPD mejor (%)	RPD prom. (%)	Tiempo medio (s)
10	10.67	15.90	313.37
15	10.35	16.06	307.22
20	10.93	15.78	306.08

3.3. Comportamiento por familia

El desempeño difirió de forma marcada entre familias de instancias (Tabla 5). Las familias abz y la concentraron las menores desviaciones, con RPD promedio de 3.21% y 3.99% y amplitudes internas reducidas (2.82 y 3.03 puntos porcentuales). En el extremo opuesto, la familia orb presentó el mayor RPD promedio (9.70%) y la mayor heterogeneidad interna (amplitud de 9.03 puntos porcentuales), e incluyó tanto instancias con RPD inferior al 7% (orb02, orb05, orb07) como superiores al 13% (orb01, orb06, orb08). La instancia ft10 se ubicó también en la franja de mayor dificultad (RPD 11.72%), coherente con su reconocida dificultad histórica.

Tabla 5. Desempeño agregado por familia (sobre el menor RPD obtenido por instancia entre las nueve configuraciones).

Familia	Instancias	RPD mín. (%)	RPD máx. (%)	RPD prom. (%)
abz	2	1.80	4.62	3.21
ft	1	11.72	11.72	11.72
la	5	2.55	5.58	3.99
orb	10	5.54	14.57	9.70

4. Discusión

El estudio paramétrico permite caracterizar empíricamente cómo se reparte el balance entre exploración y explotación entre las dos variables evaluadas. El tamaño de población actuó como el principal factor regulador de este balance: al pasar de 50 a 100 individuos, el RPD agregado del mejor *makespan* se redujo en 2.08 puntos porcentuales, mientras que la variación del umbral de reinicio entre 10 y 20 generaciones produjo diferencias inferiores a 0.6 puntos porcentuales y sin tendencia monótona. Este patrón es coherente con los mecanismos del algoritmo: una población mayor sostiene una diversidad genética más amplia y por más generaciones, lo que amplía simultáneamente la capacidad exploratoria, al mantener soluciones distintas, y la explotadora, al disponer de más material genético sobre el cual aplicar el cruce.

El umbral de reinicio, en cambio, opera como exploración correctiva ante el estancamiento; su efecto reducido sugiere que, en el rango evaluado, las tres frecuencias de reinicio resultan funcionalmente equivalentes, ya sea porque el algoritmo se estanca de forma similar bajo los tres niveles o porque los demás operadores, mutación recíproca y elitismo, mantienen una diversidad poblacional suficiente. Es plausible, además, que el rango 10-20 sea estrecho; umbrales mucho menores (3 a 5 generaciones) podrían comprometer la explotación, y umbrales mucho mayores (40 a 50) retrasar en exceso la reintroducción de diversidad.

Los resultados deben interpretarse considerando varias limitaciones que delimitan su alcance. Primero, la evaluación se restringió a 18 instancias de dimensión 10×10, por lo que no es posible extrapolar el comportamiento a instancias de mayor tamaño donde el balance entre el costo de la evaluación mediante la red de Petri y la velocidad de convergencia del algoritmo podría manifestarse de manera distinta. Segundo, el estudio varió únicamente dos parámetros en tres niveles cada uno, manteniendo el resto fijo conforme a Phanden et al. (2012); en consecuencia, las conclusiones son específicas para esa región del espacio paramétrico. Tercero, las diez corridas por configuración sustentan una caracterización descriptiva, pero no afirmaciones de significancia estadística en sentido inferencial, que requerirían un mayor número de corridas y pruebas formales de hipótesis. Cuarto, ninguna instancia se resolvió al óptimo conocido; la aproximación más cercana fue 1.80% en abz6, lo que indica margen de mejora mediante operadores de búsqueda local o presupuestos computacionales mayores. Quinto, el RPD es una métrica relativa cuya interpretación depende de la calidad del BKS de referencia, bien establecido para estas familias por trabajos posteriores a su publicación original (Carlier y Pinson, 1989; Brucker et al., 1994).

5. Conclusiones

Este trabajo evaluó, mediante un estudio paramétrico factorial completo 3×3, el efecto del tamaño de población y del umbral de reinicio por estancamiento sobre un algoritmo genético acoplado a una red de Petri temporizada adaptativa como evaluador del *makespan*, sobre 18 instancias estándar de dimensión 10×10 y un total de 1 620 corridas. El tamaño de población se identificó como el principal factor regulador de la calidad de la solución: su incremento de 50 a 100 individuos redujo de forma monótona el RPD agregado del mejor *makespan* de 11.70% a 9.62%, a costa de un tiempo de cómputo que escala de manera aproximadamente lineal con la población. El umbral de reinicio, en cambio, mostró un efecto marginal y sin tendencia monótona dentro del rango evaluado, con diferencias inferiores a 0.6 puntos porcentuales. La configuración de mejor desempeño agregado fue 100×15. El análisis por familia reveló un comportamiento diferenciado: las instancias de las familias abz y la resultaron las más tratables (RPD promedio de 3.21% y 3.99%), mientras que la familia orb y la instancia ft10 concentraron la mayor dificultad.

Estos resultados sugieren que, en el método GA+ATPPN y dentro de los rangos estudiados, conviene priorizar el incremento del tamaño de población por encima del ajuste fino del umbral de reinicio, ponderando siempre el costo computacional asociado; para las instancias estructuralmente más tratables, una población moderada con un umbral reducido puede bastar para alcanzar soluciones de buena calidad. Dado que ninguna de las 18 instancias se resolvió al óptimo conocido, la mejor aproximación fue de 1.80% en la instancia abz6, existe margen de mejora que orienta el trabajo futuro hacia la evaluación sobre instancias de mayor dimensión, como las de 15×15 o 20×20 propuestas por Taillard (1993); la incorporación de operadores de búsqueda local que complementen la exploración del algoritmo genético; y la adopción de un marco inferencial con pruebas estadísticas formales que permita establecer la significancia de los efectos observados.

Declaración sobre el uso de Inteligencia Artificial generativa y tecnologías asistidas por IA

Durante la preparación del presente manuscrito, los autores utilizaron la plataforma de Language Tool llevar a cabo la corrección gramatical y mejora de redacción. Todo el contenido generado o sugerido por la herramienta fue revisado y editado por los autores, quienes asumen plena responsabilidad por la exactitud, originalidad, integridad y rigor científico del contenido final del manuscrito.

Agradecimientos

Se agradece a la Secretaría de Ciencia, Humanidades, Tecnología e Innovación (SECIHTI) por el financiamiento y apoyo proporcionado mediante la beca de maestría 2052597, que hizo posible la realización de los estudios y actividades de investigación que dieron origen al presente trabajo.

Referencias

Adams, J., Balas, E., & Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. Management Science, 34(3), 391–401. https://doi.org/10.1287/mnsc.34.3.391

Applegate, D., & Cook, W. (1991). A computational study of the job-shop scheduling problem. ORSA Journal on Computing, 3(2), 149–156. https://doi.org/10.1287/ijoc.3.2.149

Baker, J. E. (1985). Adaptive selection methods for genetic algorithms. In J. J. Grefenstette (Ed.), Proceedings of the First International Conference on Genetic Algorithms (pp. 101–111). Lawrence Erlbaum Associates.

Brucker, P., Jurisch, B., & Sievers, B. (1994). A branch and bound algorithm for the job-shop scheduling problem. Discrete Applied Mathematics, 49, 107–127. https://doi.org/10.1016/0166-218X(94)90204-6

Carlier, J., & Pinson, E. (1989). An algorithm for solving the job-shop problem. Management Science, 35(2), 164–176. https://doi.org/10.1287/mnsc.35.2.164

Cheng, R., Gen, M., & Tsujimura, Y. (1996). A tutorial survey of job-shop scheduling problems using genetic algorithms—I. Representation. Computers & Industrial Engineering, 30(4), 983–997. https://doi.org/10.1016/0360-8352(96)00047-2

Cheng, R., Gen, M., & Tsujimura, Y. (1999). A tutorial survey of job-shop scheduling problems using genetic algorithms: Part II. Hybrid genetic search strategies. Computers & Industrial Engineering, 37, 51–55. https://doi.org/10.1016/S0360-8352(99)00022-4

Da Col, G., & Teppan, E. C. (2022). Industrial-size job shop scheduling with constraint programming. Operations Research Perspectives, 9, Article 100249. https://doi.org/10.1016/j.orp.2022.100249

Dauzère-Pérès, S., Ding, J., Shen, L., & Tamssaouet, K. (2024). The flexible job shop scheduling problem: A review. European Journal of Operational Research, 314(2), 409–432. https://doi.org/10.1016/j.ejor.2023.05.017

Escamilla-Serna, N. J., Seck-Tuoh-Mora, J. C., Medina-Marín, J., Hernández-Romero, N., Barragán-Vite, I., & Corona-Armenta, J. R. (2021). A global-local neighborhood search algorithm and tabu search for flexible job shop scheduling problem. PeerJ Computer Science, 7, Article e574. https://doi.org/10.7717/peerj-cs.574

Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research, 1(2), 117–129. https://doi.org/10.1287/moor.1.2.117

He, Z., Li, N., Ran, N., & Li, L. (2026). Scheduling of flexible manufacturing systems based on place-timed Petri nets and basis reachability graphs. IEEE Transactions on Control Systems Technology. Advance online publication. https://doi.org/10.1109/TCST.2026.3664466

Holland, J. H. (1992). Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence. MIT Press.

Hu, K., Wang, L., Cai, J., & Cheng, L. (2023). An improved genetic algorithm with dynamic neighborhood search for job shop scheduling problem. Mathematical Biosciences and Engineering, 20(9), 17407–17427. https://doi.org/10.3934/mbe.2023774

Huang, B., Zhou, M., Abusorrah, A., & Sedraoui, K. (2022). Scheduling robotic cellular manufacturing systems with timed Petri net, A* search, and admissible heuristic function. IEEE Transactions on Automation Science and Engineering, 19(1), 243–250. https://doi.org/10.1109/TASE.2020.3026351

Jain, A. S., & Meeran, S. (1999). Deterministic job-shop scheduling: Past, present and future. European Journal of Operational Research, 113(2), 390–434. https://doi.org/10.1016/S0377-2217(98)00113-1

Lawrence, S. (1984). Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques (Supplement). Graduate School of Industrial Administration, Carnegie-Mellon University.

Lefebvre, D., & Basile, F. (2021). An approach based on timed Petri nets and tree encoding to implement search algorithms for a class of scheduling problems. Information Sciences, 559, 314–335. https://doi.org/10.1016/j.ins.2020.12.087

Lin, X., Xu, Z., Xie, S., Yang, F., Wu, J., & Li, D. (2025). Symmetry-aware dynamic scheduling optimization in hybrid manufacturing flexible job shops using a time Petri nets improved genetic algorithm. Symmetry, 17(6), Article 907. https://doi.org/10.3390/sym17060907

Medina-Marín, J., Seck-Tuoh-Mora, J. C., Hernández-Romero, N., & Escamilla-Serna, N. J. (2014). Applying an adaptive Petri net to calculate the makespan in the job shop scheduling problem. In A. Affenzeller, A. Bruzzone, E. Jiménez, F. Longo, Y. Merkuryev, & M. Zhang (Eds.), Proceedings of the European Modeling and Simulation Symposium (pp. 463–468).

Murata, T. (1989). Petri nets: Properties, analysis and applications. Proceedings of the IEEE, 77(4), 541–580. https://doi.org/10.1109/5.24143

Mušič, G. (2021). Petri net based solution supervision and local search for job shop scheduling. IFAC-PapersOnLine, 54(1), 665–670. https://doi.org/10.1016/j.ifacol.2021.08.177

Muth, J. F., & Thompson, G. L. (Eds.). (1963). Industrial scheduling. Prentice-Hall.

Phanden, R. K., Jain, A., & Verma, R. (2012). A genetic algorithm-based approach for job shop scheduling. Journal of Manufacturing Technology Management, 23(7), 937–946. https://doi.org/10.1108/17410381211267745

Pinedo, M. L. (2008). Scheduling: Theory, algorithms, and systems (3rd ed.). Springer.

Ruiz, R., & Maroto, C. (2006). A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. European Journal of Operational Research, 169(3), 781–800. https://doi.org/10.1016/j.ejor.2004.06.038

Taillard, E. (1993). Benchmarks for basic scheduling problems. European Journal of Operational Research, 64, 278–285. https://doi.org/10.1016/0377-2217(93)90182-M

Tamy0612. (n.d.). JSPLIB: Benchmark instances for the job-shop scheduling problem [Data set]. GitHub. Recuperado el 7 de mayo de 2026, de https://github.com/tamy0612/JSPLIB

Xiong, H., Shi, S., Ren, D., & Hu, J. (2022). A survey of job shop scheduling problem: The types and models. Computers & Operations Research, 142, Article 105731. https://doi.org/10.1016/j.cor.2022.105731

Zhou, M., & Venkatesh, K. (1999). Modeling, simulation, and control of flexible manufacturing systems: A Petri net approach. World Scientific.