Heurística de búsqueda local aplicada al problema de Flow Shop

Oscar Mauricio Cepeda Valero[1]

Universidad de los Andes
Cra 1 Nº 18A - 12, 28922, Bogotá, Colombia
om.cepeda86@uniandes.edu.co

Abstract

El problema de flow shop consiste en la programación de un conjunto de trabajos dados con el mismo orden en todas las máquinas. Cada trabajo puede ser procesado como máximo en una máquina. Mientras tanto, una máquina puede procesar como máximo un trabajo. El objetivo más común para este problema es minimizar la duración de todos los trabajos. Así, en el documento se presenta el diseño y aplicación de un algoritmo de búsqueda local para el problema de flow shop. Para el diseño del algoritmo de búsqueda local se codificó la secuencia de trabajos en un vector, sobre el cual se aplicó un operador de intercambio de trabajos y otro de inversión del vector. La implementación de los métodos de búsqueda presentan una gran simplicidad en su aplicación. Sin embargo, para la obtención de buenos resultados requieren de un gran tiempo de computo. La aplicación se llevó sobre 10 instancias de literatura con 500 trabajos en 20 maquinas. En promedio los resultados aplicados presentan un alejamiento de 5.6% respecto a la mejor solución reportada, con el operador de intercambio para la generación del vecindario.

1 Introducción

En el problema de flow shop, un conjunto de trabajos debe procesarse en un conjunto de máquinas en idéntico orden. El objetivo es determinar la secuencia de trabajos para optimizar una determinada función objetivo. En un momento dado, cada máquina puede procesar como máximo un trabajo, y cada trabajo puede ser gestionado como máximo por una máquina. Además, cada trabajo no puede ser adelantado por otros trabajos. Este tipo de problemas tipo flow shop son comunes en el ámbito industrial. Un ejemplo es el caso del acero, inicialmente se funde en planchas semiacabadas; después de calentarse en el horno de calor, las planchas se laminan en productos en el tren de laminación *[7]. Este caso es un ejemplo de producción de taller de flujo (flow shop). El problema de flow shop puede tener diferentes objetivos, usualmente se centra en minimizar el tiempo total de finalización todos los trabajos, es decir, el makespan. Además, también se consideran objetivos como el tiempo total de flujo, la tardanza y el tiempo de inactividad. [11] Este es un problema tipo NP-hard, por lo tanto es difícil obtener una solución global óptima en tiempo polinómico. Por lo tanto, el estudio de los algoritmos de programación del taller de flujo es muy importante para reducir el tiempo de ejecución y aumentar la productividad.

Aunque existen una diversidad de métodos aplicados al problema de flow shop y los diferentes problemas de scheduling. Este conjunto de problemas continua siendo de alto interés tanto académico como práctico. En este sentido, la evaluación de nuevos métodos son una contribución importante en busca de nuevas formas de abarcar este tipo de problemas.

En la literatura, los problemas de programación de taller de flujo (flow shop) o taller de trabajo (job shop) han sido ampliamente estudiados. De estos mismos existen una cantidad significativa de variaciones del problema [5, 8, 12]

La mayoría de los problemas de programación, como el problema de flujo en talleres son NP-duros, por lo que la heurística es la principal forma de abordar estos problemas. Las estrategias heurísticas sencillas pueden basarse en la aplicación de reglas de reglas de despacho basadas en la prioridad. Las heurísticas más eficaces representan algoritmos específicos que se desarrollados para algún tipo especial de problema. Dado que los problemas de la práctica suelen tener características diferentes, la aplicación de estas heurísticas puede implicar un costoso desarrollo de un algoritmo que solo aplica para para un caso especifico y que puede ser complejo llevarlos a la aplicación en el mundo real. Este importante

problema puede resolverse aplicando metaheurísticas, las cuales son ampliamente genéricas con respecto al tipo de problema y que permiten contemplar aspectos específicos del problema [3]. Diversas metaheurísticas se han aplicado, entre las cuales se encuentran colonia de hormigas [4], algoritmos genéticos [2], búsqueda tabú [10], optimización de enjambre de partículas [1], entre otros.

En la implementación del algoritmo de búsqueda local, se logró desarrollar un caso interesante en una búsqueda local. Dónde la generación del vecindario varia según el lugar en el cuál se encuentre la búsqueda. Así, inicialmente se hacen búsquedas en un vecindario muy amplio con un operador y a medida que el algoritmo se empieza a centrar en una zona, se aplica otro operador que permite profundizar en el vecindario.

La estructura de este artículo es la siguiente. La formulación del problema y la metodología para abarcarlo se presenta en la Sección 2. Los resultados de la aplicación del modelo en las 23 instancias se muestran en la Sección 3. La discusión de los resultados se presentan en la sección 4, y la sección 5 presenta las principales conclusiones del estudio.

2 Metodología

El problema de flow shop trata de la asignación de un conjunto de trabajos a ser procesados en un conjunto de maquinas. El objetivo es identificar la secuencia de procesamiento de los trabajos en busca de minimizar el tiempo de procesamiento de todos los trabajos.

En este caso, todos los trabajos pasan por todas la máquinas. Para ello se proponen dos algoritmos de búsqueda local, los cuales son evaluados sobre las instancias $tai500 - 20 - 1$ a $tai500 - 20 - 10$ presentadas en Taillard [9]. Las cuales se encuentran compuestas por 500 trabajos en 20 maquinas.

2.1 Operadores de búsqueda

Inicialmente la representación de las soluciones del problema se estructuró como un vector donde el número de la primera posición corresponderá al primer trabajo a procesar y consecuentemente se procesaran. Un ejemplo se presenta en la Figura 1, donde el primer trabajo a procesar será el 372, seguido por el 365 y continua hasta procesar de ultimas el trabajo 465

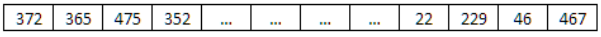

Figure 1: Codificación para solución de problema Flow Shop

A la codificación presentada se le aplicaron los dos operadores propuestos para la búsqueda local. En este caso, se utilizó intercambio e inversión, los cuales se ilustran en la figura 2

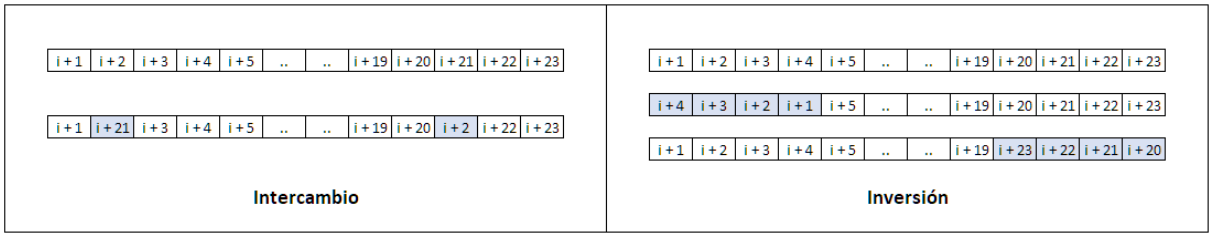

Figure 2: Operadores para generación del vecindario en la búsqueda local

Para la evaluación de los algoritmos El operador de intercambio selecciona dos trabajos aleatorios y los permuta, para la generación de un nuevo vecino. Mientras tanto, el operador de inversión selecciona un punto de corte del vector solución y de forma aleatoria selecciona la inversión de la parte izquierda o derecha del vector.

2.2 Algoritmo

El algoritmo de búsqueda local se implementó en $Python$, el cual se compone de los algoritmos de crear el la instancia, evaluar la función objetivo y construir el vecindario:

Algorithm 1 Búsqueda local

procedure BÚSQUEDA LOCAL (MÁXIMO ITERACIONES)
  $M \leftarrow$ Cargar instancia (matriz de tiempos)
  $So \leftarrow$ Solución actual
  $i \leftarrow 0$
  while $i <$máximo de iteraciones **do**
    $Fbest$ = Encontrar función objetivo FO(So,M)
    Construir vecindario (So, b)
    Evaluar vecindario
    $best$ = mejor punto del vecindario
    if funcion objetivo de FO($best$, M) $<Fbest$ **then**
      Actualizar $Fbest$
      Actualizar So
    else if then
      $i \leftarrow i + 1$.
  return So, $Fbest$
  end

En el algoritmo, la construcción de la instancia recibe el valor de la semilla reportada por Taillard [9] y retorna la matriz de tiempos de cada trabajo en cada maquina M. El algoritmo de evaluar la función objetivo recibe un vector solución So y la matriz de tiempos M y retorna el tiempo total de hacer todos los trabajos en la secuencia del vector. Finalmente, el algoritmo de construir el vecindario recibe la solución actual So, un valor b asociado al tamaño del vecindario a generar. Este algoritmo funciona con cualquiera de los dos operadores (intercambio o inversión) y genera una cantidad de soluciones según el tamaño de b

2.3 Calibración

La calibración de los dos algoritmos se baso en identificar el máximo de iteraciones y el tamaño del vecindario más indicado para obtener buenos resultados en un tiempo aceptable. El máximo de iteraciones sin encontrar mejora evaluados fueron $1, 2, 3, 5, 10, 20$, mientras los tamaños de vecindario evaluados fueron $5, 20, 30, 50$. El tamaño de vecindario significa la cantidad de soluciones que se generan a partir de la solución actual. Todos los experimentos computacionales se realizaron en una CPU Intel(R) Core(TM) i7-10610U a 1,80GHz 2,30 GHz (con 16GB de RAM) utilizando Python 3.9.7.

3 Resultados

3.1 Algoritmo

Para la evaluación del algoritmo, inicialmente se desarrollo la calibración tomando de base el problema $tai500 - 20 - 1$ (1368624604) y se utilizó el operador de inversión. En busca de identificar el tamaño del vecindario a utilizar y el número de iteraciones sin mejora para la aplicación del algoritmo. En primer momento se mantuvo el tamaño de vecindario en 20 vecinos y se observó el comportamiento con diferentes máximos de iteraciones sin mejora. Los resultados tanto de la función objetivo como de los tiempos de computo se presentan en la figura 3. Para cada uno de los escenarios evaluados se corrieron 30 replicas.

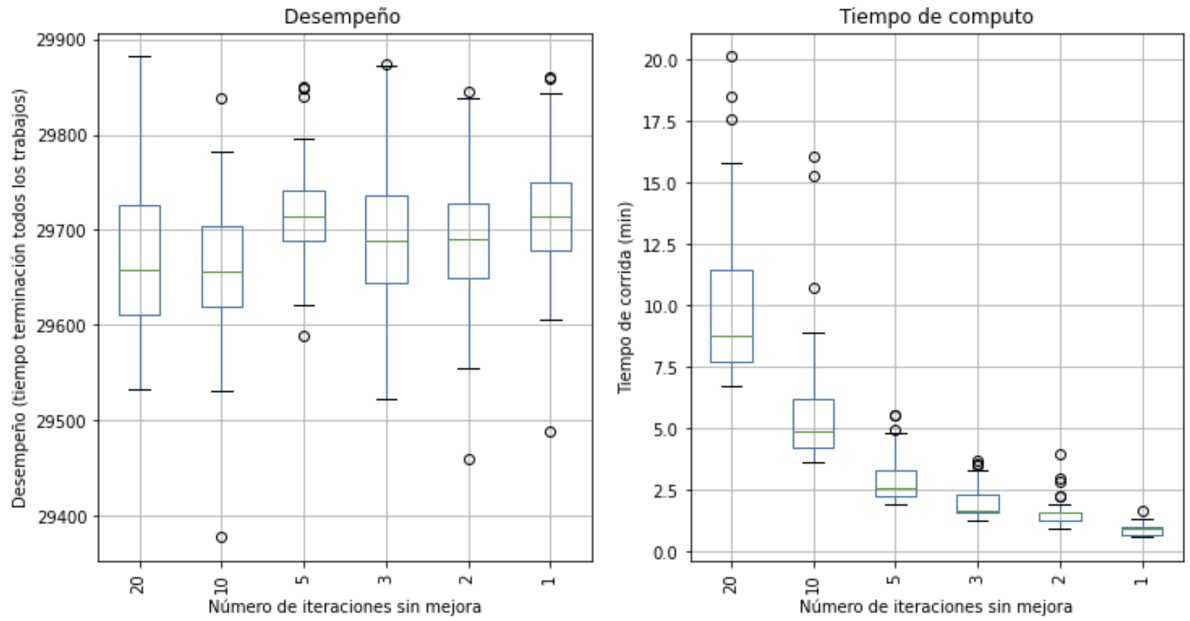

Figure 3: Calibración máximo de iteraciones

De las instancias corridas, se observa el alto impacto en el tiempo de corrida, sin embargo la diferencia en el desempeño entre tener un máximo de iteraciones de 10 y 20 no es tan significativo. Por lo tanto, se seleccionó un máximo de 10 iteraciones sin mejora para evaluar el algoritmo.

Posteriormente, se calibró el tamaño del vecindario. De igual manera, se trabajo con el problema $tai500 - 20 - 1$ (1368624604), el operador de inversión y un máximo de 10 iteraciones sin mejora. Los resultados se presentan en la figura 4

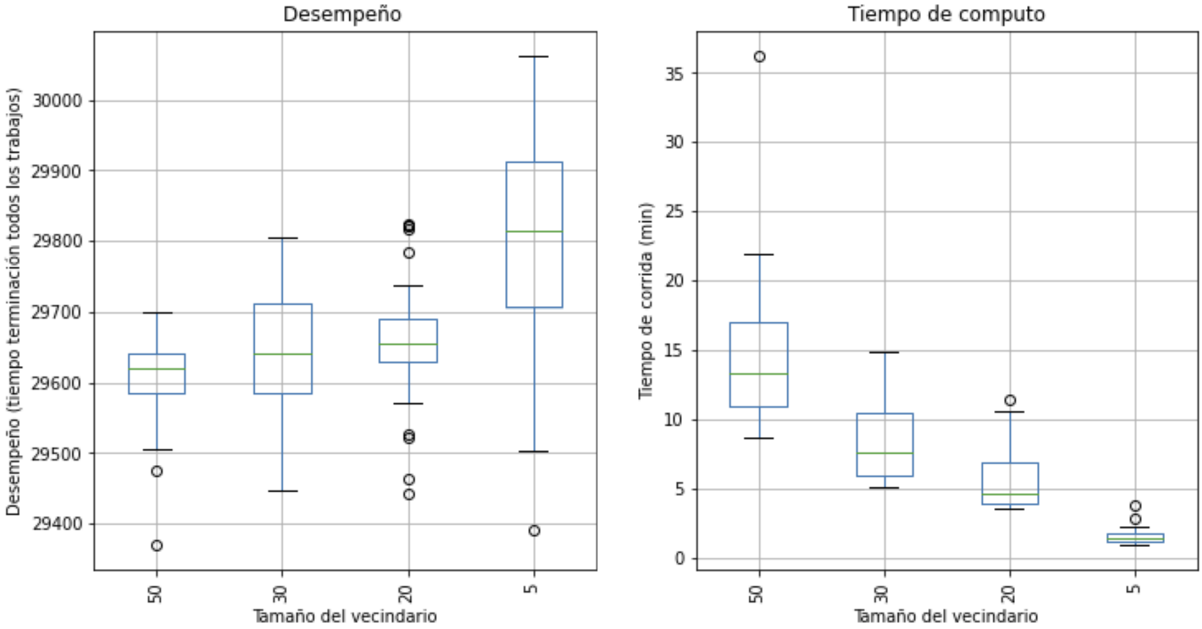

Figure 4: Calibración tamaño del vecindario

Con lo observado en la figura, de igual manera el tamaño del vecindario tiene un alto impacto en el tiempo de computo, al igual que en encontrar una mejor solución. Se seleccionó un tamaño de vecindario

de 20 el cual genera resultados cercanos en comparación con los otros tamaños de vecindario usados. Sin embargo, el tiempo de computo si es significativamente menor.

3.2 Evaluación

Con la calibración del algoritmo se procedió a evaluar cada una de las instancias. Debido a que los dos operadores tienen un componente aleatorio, la evaluación del desempeño del algoritmo también se hizo con 30 replicas. A continuación, en la figura se 5 presentan los resultados para cada una de las instancias con cada uno de los operadores.

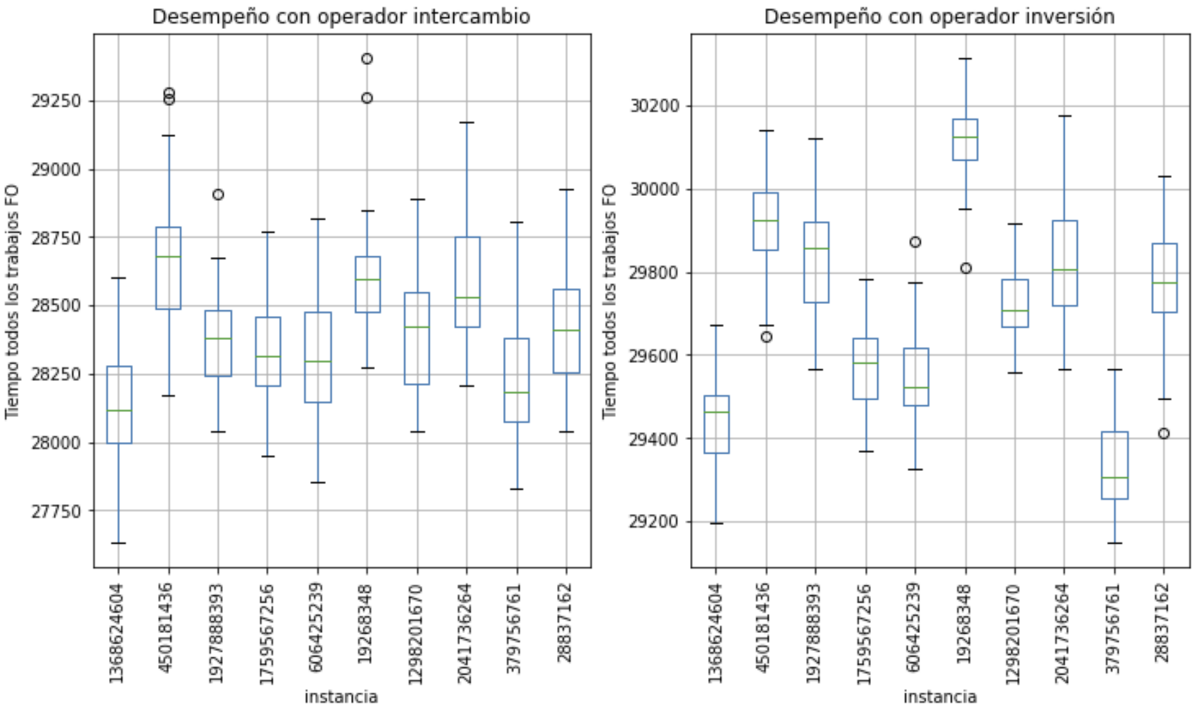

Figure 5: Corridas de las instancias con el algoritmo de búsqueda local

En lo correspondiente a la aleatoriedad, en el operador de intercambio corresponde a la selección de los puntos de intercambio, mientras en el operador de inversión se selecciona aleatoriamente el punto de corte y cual de las dos partes del vector es invertida.

El algoritmo de búsqueda construido logró mejorar todas las instancias con los dos operadores. En los resultados presentados el operador de intercambio genera mejores resultados, sin embargo es mucho más lento.

4 Discusión

De los resultados obtenidos en cada una de las instancias se compararon con los limites reportados por Taillard [9]. La comparación de los resultados se presenta en la tabla 1

En la tabla 1 se observa que la búsqueda local utilizando el operador de intercambio genera mejores resultados, con un GAP promedio de 5.6%, mientras tanto al aplicar el operador de inversión tiene un GAP promedio de 10.4%. En este mismo orden de ideas, mientras la búsqueda local con el operador de intercambio mejora la solución inicial alrededor de un 7%, el operador de inversión tan solo la mejora con un promedio de 2.8%. Actualmente, entre los mejores resultados obtenidos para los problemas de 500 trabajos y 20 máquinas, los mejores resultados se han logrado con mezclas de metaheurísticas como

Time	UB	Inicial	Intercambio	Inversión	Mejora 1	Mejora 2	GAP1	GAP1
1368624604	26699	30121	28150	29440	6.5%	2.3%	5.4%	10.3%
450181436	27303	31202	28661	29912	8.1%	4.1%	5.0%	9.6%
1927888393	26928	30447	28383	29840	6.8%	2.0%	5.4%	10.8%
1759567256	27009	30355	28335	29572	6.7%	2.6%	4.9%	9.5%
606425239	26771	30099	28319	29548	5.9%	1.8%	5.8%	10.4%
19268348	26959	30946	28604	30117	7.6%	2.7%	6.1%	11.7%
1298201670	26870	30792	28406	29723	7.7%	3.5%	5.7%	10.6%
2041736264	27104	31034	28598	29830	7.8%	3.9%	5.5%	10.1%
379756761	26586	30634	28239	29334	7.8%	4.2%	6.2%	10.3%
28837162	26910	30148	28436	29772	5.7%	1.2%	5.7%	10.6%

Table 1: Resultados aplicación algoritmo de búsqueda local

estrategias evolutivas híbridas y simulado templado [6]. En este caso, la solución obtenida por el método de búsqueda local planteado presenta un alejamiento mayor con GAPs cercanos al 7%.

Con el anterior análisis la búsqueda local con el operador de intercambio parece más efectiva, sin embargo, con el análisis de tiempos el operador de intercambio genera que el proceso sea demasiado lento. Con tiempos computacionales alrededor de 25 minutos, no obstante el operador de inversión demora alrededor de 7 minutos en llegar a estabilizarse.

Basado en los resultados, se propone una combinación de los operadores en la búsqueda local. Esto permitiría encontrar soluciones de una forma más rápida, debido a que con la inversión se pueden encontrar fracciones o secuencias de soluciones largas dentro del vector solución, mientras el operador de intercambio genera cambios pequeños que permiten una exploración más profunda del vecindario. De esta manera, las instancias fueron evaluadas con la mezcla de operadores, con un vecindario de tamaño 20 y un máximo de iteraciones de 5. Aunque los resultados fueron buenos en eficiencia con un alejamiento promedio de 7%, los tiempos de computo están alrededor de 25 minutos, los cuales no presentan una gran diferencia con el operador de intercambio. La aplicación sobre las instancias se presenta en la figura 6

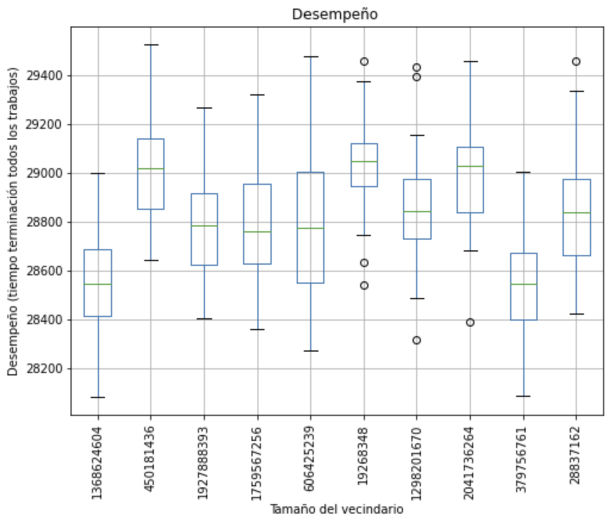

Figure 6: Corridas de las instancias con búsqueda local con mezcla de operadores de generación de vecindario

5 Conclusiones

En este artículo se estudio el problema de flow shop, para el cual se construyó un algoritmo de búsqueda local. El algoritmo se aplico con dos operadores para la generación del vecindario. Un operador de intercambio, el cual cambia dos trabajos aleatorios dentro del vector solución, y un operador de inversión que selecciona un punto de corte del vector y de las dos partes aleatoriamente se selecciona una para invertir.

El algoritmo de búsqueda construido tiene dos parámetros, el tamaño del vecindario y el número de iteraciones sin generar mejora en la solución. Estos dos parámetros fueron calibrados y se encontró que el mejor desempeño del algoritmo es cuando el vecindario es de 20 y el máximo numero de iteraciones sin mejora es de 10.

El algoritmo fue aplicado sobre 10 instancias de literatura de 500 trabajos en 20 maquinas. Para estas instancias se aplicaron ambos operadores y se encontró una mayor eficiencia cuando el algoritmo se utiliza con el operador de intercambio, generando diferencias con la mejor solución reportada en literatura de 5.6%. Sin embargo, este operador es bastante lento dado que para cada corrida del algoritmo toma alrededor de 25 minutos.

Finalmente, se identificaron los beneficios de los dos operadores aplicados. El operador de inversión es bastante bueno para dirigir rápidamente la solución a regiones prometedoras, mientras tanto el operador de intercambio permite profundizar en regiones buenas. Esto permitió generar una heurística de búsqueda local interesante, la cual permite explorar vecindarios amplios en el inicio y profundizar cuando se tienen regiones prometedoras.

References

[1] Junwen Ding, Sven Schulz, Liji Shen, Udo Buscher, and Zhipeng Lü. Energy aware scheduling in flexible flow shops with hybrid particle swarm optimization. *Computers & Operations Research*, 125:105088, 2021.

[2] O Etiler, Bilal Toklu, M Atak, and J Wilson. A genetic algorithm for flow shop scheduling problems. *Journal of the Operational Research Society*, 55(8):830–835, 2004.

[3] Andreas Fink and Stefan Voß. Solving the continuous flow-shop scheduling problem by metaheuristics. *European Journal of Operational Research*, 151(2):400–414, 2003.

[4] Dunwei Gong, Yuyan Han, and Jianyong Sun. A novel hybrid multi-objective artificial bee colony algorithm for blocking lot-streaming flow shop scheduling problems. *Knowledge-Based Systems*, 148:115–130, 2018.

[5] Yuyan Han, Dunwei Gong, Junqing Li, and Yong Zhang. Solving the blocking flow shop scheduling problem with makespan using a modified fruit fly optimisation algorithm. *International Journal of Production Research*, 54(22):6782–6797, 2016.

[6] Bilal Khurshid, Shahid Maqsood, Muhammad Omair, Biswajit Sarkar, Imran Ahmad, and Khan Muhammad. An improved evolution strategy hybridization with simulated annealing for permutation flow shop scheduling problems. *IEEE Access*, 9:94505–94522, 2021.

[7] Tao Ren, Meiting Guo, Lin Lin, and Yunhui Miao. A local search algorithm for the flow shop scheduling problem with release dates. *Discrete Dynamics in Nature and Society*, 2015, 2015.

[8] Yi Sun, Chaoyong Zhang, Liang Gao, and Xiaojuan Wang. Multi-objective optimization algorithms for flow shop scheduling problem: a review and prospects. *The International Journal of Advanced Manufacturing Technology*, 55(5):723–739, 2011.

[9] Eric Taillard. Benchmarks for basic scheduling problems. *european journal of operational research*, 64(2):278–285, 1993.

[10] Moch Saiful Umam, Mustafid Mustafid, and Suryono Suryono. A hybrid genetic algorithm and tabu search for minimizing makespan in flow shop scheduling problem. *Journal of King Saud University-Computer and Information Sciences*, 2021.

[11] Betul Yagmahan and Mehmet Mutlu Yenisey. Ant colony optimization for multi-objective flow shop scheduling problem. *Computers & Industrial Engineering*, 54(3):411–420, 2008.

[12] Mehmet Mutlu Yenisey and Betul Yagmahan. Multi-objective permutation flow shop scheduling problem: Literature review, classification and current trends. *Omega*, 45:119–135, 2014.