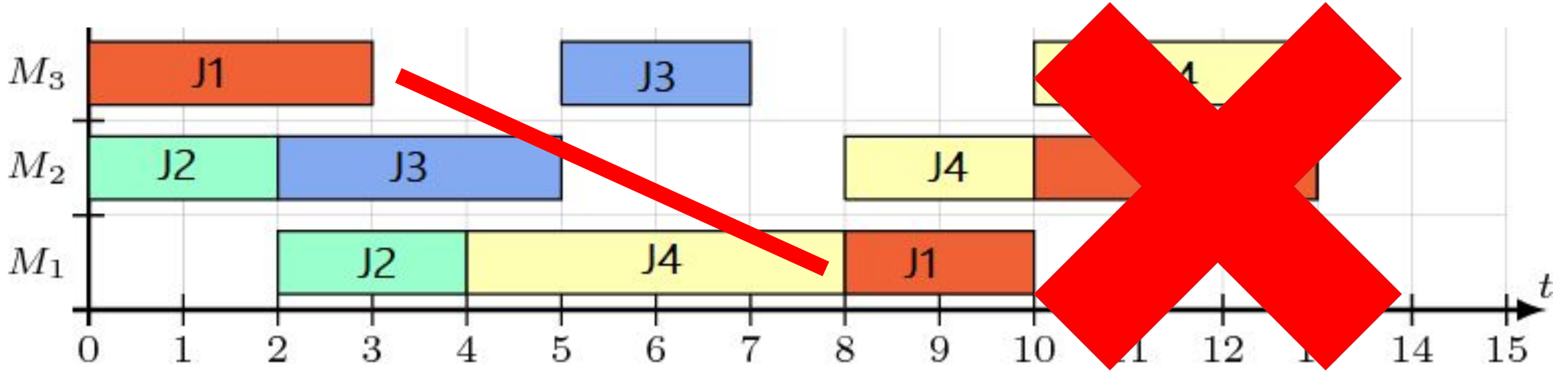
No-Wait Job Shop Scheduling Problem (NWJSSP)

El NWJSSP consiste en la asignación de un conjunto de trabajos a un conjunto de máquinas (recursos), donde cada trabajo tiene un conjunto de operaciones que, una vez iniciadas deben ser procesadas inmediatamente, una tras otra hasta la finalización del trabajo.[1]



Metodología - Representación basada en retardos

Se propone una representación basada en retardos

Un cromosoma es representado de la siguiente manera:

$c = [(d_1,J_1),(d_2,J_2),...,(d_n,J_n)]$, donde d_i es el retardo asociado al trabajo J_i .

Es decir, el tiempo de inicio de la primera operación del trabajo J_i es igual a d_i.

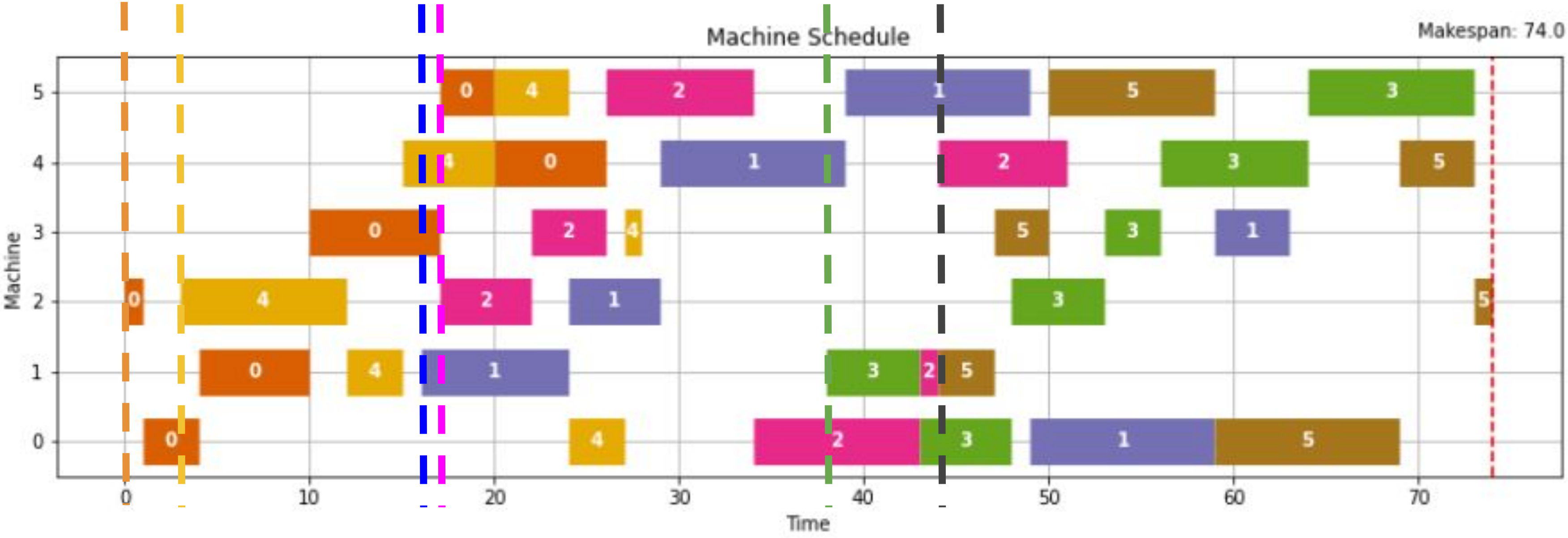
Metodología - Representación basada en retardos

Ejemplo:

c = [(0, 0), (16, 1), (17, 2), (38, 3), (3, 4), (44, 5)]

representa una solución factible a la instancia *ft06* (fisher, 1963).

Metodología - Representación basada en retardos



c = [(0, 0), (16, 1), (17, 2), (38, 3), (3, 4), (44, 5)]

Metodología - FUMDAN (A Feasible UMDAc implementation for NWJSSP)

Se empleó una adaptación al algoritmo UMDAc, denominado FUMDAN (a Feasible UMDAc implementation for NWJSSP).

Esta adaptación, toma el algoritmo base de UMDAc y fue ajustado para el problema de No-wait Job Shop Scheduling Problem, donde se destacan 3 cambios principales:

Metodología - FUMDAN (A Feasible UMDAc implementation for NWJSSP)

1. Los individuos iniciales del algoritmos son generados de manera aleatoria acotada entre 0 y (*Max Start - Job Makespan*$_j$), donde:

   - *Max Start* corresponde a la suma de todas las operaciones de todos los trabajos.

   - *Job Makespan*$_j$ corresponde a la suma de todas las operaciones del trabajo j-ésimo.

Metodología - FUMDAN (A Feasible UMDAc implementation for NWJSSP)

---

2. Se empleó una función auxiliar llamada *MAKE FEASIBLE SOLUTION* la cual dado cualquier cromosoma, regresa un calendario válido, esto lo consigue de la siguiente forma:

- Calendariza el trabajo J_i con su tiempo de retardo d_i si no se genera ninguna colisión en el calendario.

- En caso de que al calendarizar el trabajo J_i genere una colisión, calendariza el trabajo lo más cercano al 0 de tal manera que no genere colisiones.

Metodología - FUMDAN (A Feasible UMDAc implementation for NWJSSP)

---

3. Los individuos al finalizar una generación son reemplazados utilizando una distribución normal truncada

$$TN(\mu_j, \sigma_j^2, 0, \mathit{Max\ Start})$$

acotada entre 0 y *Max Start*, donde *Max Start* corresponde al menor *Makespan* encontrado por el algoritmo.

Algorithm 1 FUMDAN

$Max\ Start \leftarrow \sum_{j \in J} o_{j\mu_{jn_j}}$ ▷ Suma de todas las operaciones de todos los trabajos

$Job\ Makespan_j \leftarrow \sum o_{j\mu_{jn_j}}$ ▷ Suma de todas las operaciones del trabajo j-esimo

$P \leftarrow TN(\mu_0, \sigma_0^2, 0, Max\ Start - Job\ Makespan_j)$ ▷ TN Distribución Normal Truncada

for generación $i = 1$ hasta $i \leq$ Generaciones **do**

    for Individuo en Población **do**

        MAKE FEASIBLE SOLUTION(*Individuo*)

    end for

    Calcular $Makespan$ de cada individuo en P

    if $Makespan$ del mejor individuo $< Max\ Start$ **then**

        $Max\ Start \leftarrow Makespan$ del mejor individuo

    end if

    Seleccionar n individuos de P utilizando *Selección por torneo*

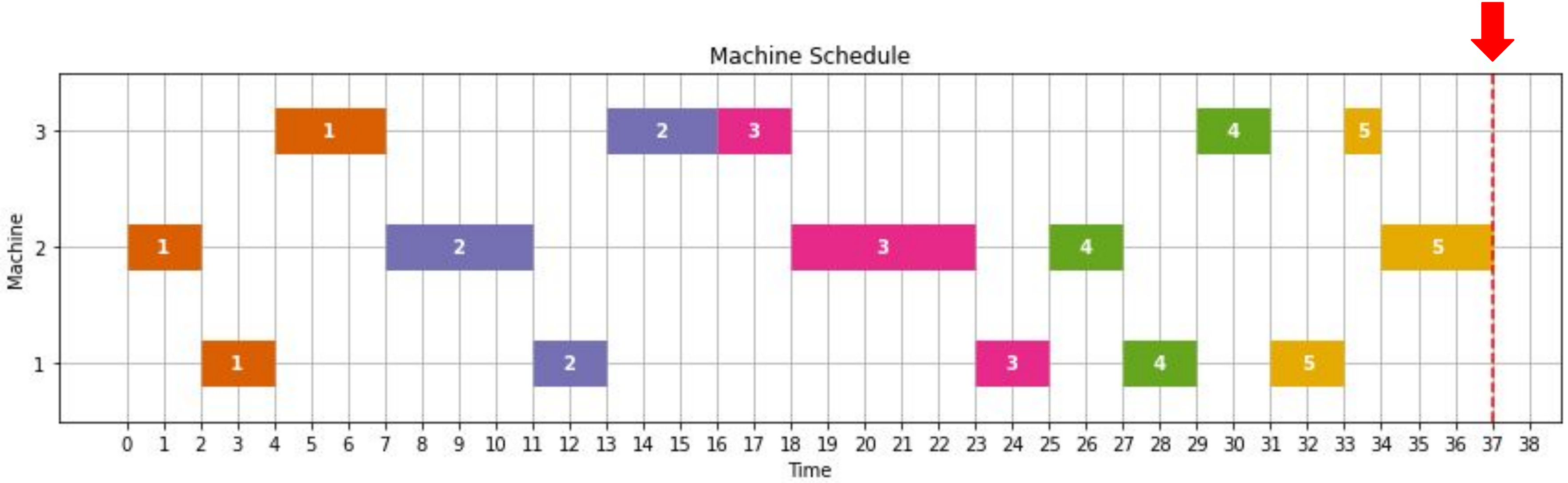
    Estimar μ_j y σ_j^2 para cada tiempo de retardo de cada Trabajo j

    $P \leftarrow TN(\mu_j, \sigma_j^2, 0, Max\ Start)$ ▷ TN Distribución Normal Truncada

end for

Algorithm 1 FUMDAN

1: $Max\ Start \leftarrow \sum_{j \in J} o_{j\mu_{jn_j}}$ ▷ Suma de todas las operaciones de todos los trabajos



Algorithm 1 FUMDAN

$\mathit{Max\ Start} \leftarrow \sum_{j \in J} o_{j\mu_{jn_j}}$ ▷ Suma de todas las operaciones de todos los trabajos

$\mathit{Job\ Makespan}_j \leftarrow \sum o_{j\mu_{jn_j}}$ ▷ Suma de todas las operaciones del trabajo j-esimo

$P \leftarrow TN(\mu_0, \sigma_0^2, 0, \mathit{Max\ Start}$ - $\mathit{Job\ Makespan}_j)$ TN Distribución Normal Truncada

for generación $i = 1$ hasta $i \leq$ Generaciones **do**

  for Individuo en Población **do**

    MAKE FEASIBLE SOLUTION(*Individuo*)

  end for

  Calcular $\mathit{Makespan}$ de cada individuo en P

  if $\mathit{Makespan}$ del mejor individuo $<$ $\mathit{Max\ Start}$ **then**

    $\mathit{Max\ Start} \leftarrow \mathit{Makespan}$ del mejor individuo

  end if

  Seleccionar n individuos de P utilizando *Selección por torneo*

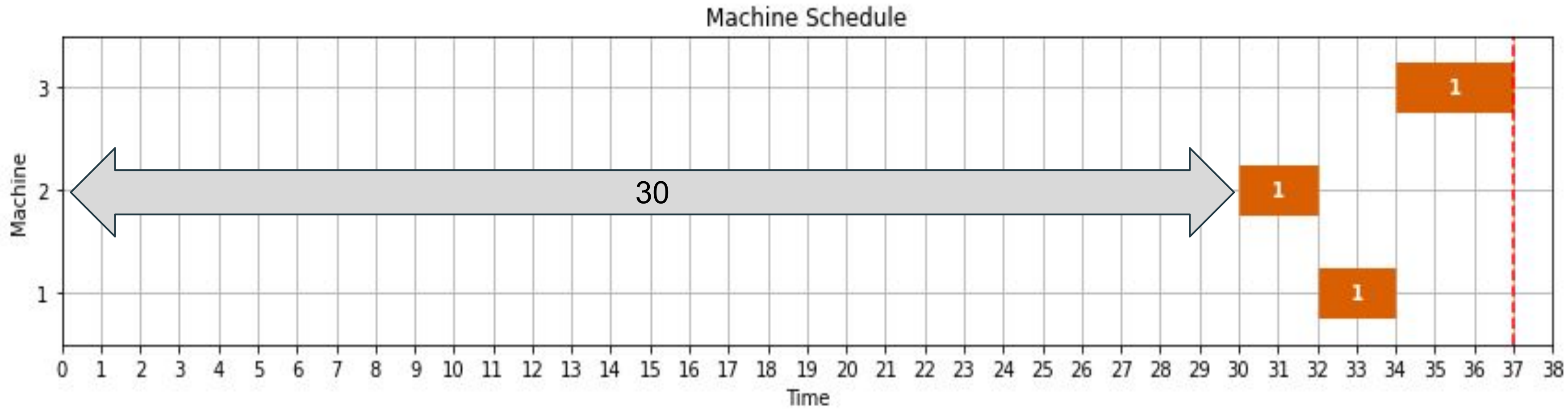
  Estimar μ_j y σ_j^2 para cada tiempo de retardo de cada Trabajo j

  $P \leftarrow TN(\mu_j, \sigma_j^2, 0, \mathit{Max\ Start})$ ▷ TN Distribución Normal Truncada

end for

2: $Job\ Makespan_j \leftarrow \sum o_{j\mu_{jn_j}}$ ▷ Suma de todas las operaciones del trabajo j-esimo

3: $P \leftarrow TN(\mu_0, \sigma_0^2, 0, Max\ Start\ \text{-}\ Job\ Makespan_j)$ ▷ TN Distribución Normal Truncada



Algorithm 1 FUMDAN

$\textit{Max Start} \leftarrow \sum_{j\in J} o_{j\mu_{jn_j}}$ ▷ Suma de todas las operaciones de todos los trabajos
$\textit{Job Makespan}_j \leftarrow \sum o_{j\mu_{jn_j}}$ ▷ Suma de todas las operaciones del trabajo j-esimo
$P \leftarrow TN(\mu_0, \sigma_0^2, 0, \textit{Max Start - Job Makespan}_j)$ ▷ TN Distribución Normal Truncada
for generación $i = 1$ hasta $i \leq$ Generaciones **do**
  for Individuo en Población **do**
    MAKE FEASIBLE SOLUTION(*Individuo*)
  end for
  Calcular *Makespan* de cada individuo en P
  if *Makespan* del mejor individuo $<$ *Max Start* **then**
    Max Start $\leftarrow$ *Makespan* del mejor individuo
  end if
  Seleccionar n individuos de P utilizando *Selección por torneo*
  Estimar μ_j y σ_j^2 para cada tiempo de retardo de cada Trabajo j
  $P \leftarrow TN(\mu_j, \sigma_j^2, 0, \textit{Max Start})$ ▷ TN Distribución Normal Truncada
end for

Algorithm 1 FUMDAN

$\textit{Max Start} \leftarrow \sum_{j \in J} o_{j\mu_{jn_j}}$ ▷ Suma de todas las operaciones de todos los trabajos
$\textit{Job Makespan}_j \leftarrow \sum o_{j\mu_{jn_j}}$ ▷ Suma de todas las operaciones del trabajo j-esimo
$P \leftarrow TN(\mu_0, \sigma_0^2, 0, \textit{Max Start - Job Makespan}_j)$ ▷ TN Distribución Normal Truncada
for generación $i = 1$ hasta $i \leq$ Generaciones **do**
    for Individuo en Población **do**
        MAKE FEASIBLE SOLUTION(*Individuo*)
    end for
    Calcular *Makespan* de cada individuo en P
    if *Makespan* del mejor individuo $<$ *Max Start* **then**
        Max Start $\leftarrow$ *Makespan* del mejor individuo
    end if
    Seleccionar n individuos de P utilizando *Selección por torneo*
    Estimar μ_j y σ_j^2 para cada tiempo de retardo de cada Trabajo j
    $P \leftarrow TN(\mu_j, \sigma_j^2, 0, \textit{Max Start})$ ▷ TN Distribución Normal Truncada
end for

Algorithm 1 FUMDAN

$Max\ Start \leftarrow \sum_{j \in J} o_{j\mu_{jn_j}}$ ▷ Suma de todas las operaciones de todos los trabajos

$Job\ Makespan_j \leftarrow \sum o_{j\mu_{jn_j}}$ ▷ Suma de todas las operaciones del trabajo j-esimo

$P \leftarrow TN(\mu_0, \sigma_0^2, 0, Max\ Start$ - $Job\ Makespan_j)$ ▷ TN Distribución Normal Truncada

for generación $i = 1$ hasta $i \leq$ Generaciones **do**

    for Individuo en Población **do**

        MAKE FEASIBLE SOLUTION(*Individuo*)

    end for

    Calcular $Makespan$ de cada individuo en P

    if $Makespan$ del mejor individuo $<$ $Max\ Start$ **then**

        $Max\ Start \leftarrow Makespan$ del mejor individuo

    end if

    Seleccionar n individuos de P utilizando *Selección por torneo*

    Estimar μ_j y σ_j^2 para cada tiempo de retardo de cada Trabajo j

    $P \leftarrow TN(\mu_j, \sigma_j^2, 0, Max\ Start)$ ▷ TN Distribución Normal Truncada

end for

```
[0    1], [43   2], [45   3], [19   4], [67   5], [27   6]]
[68   1], [52   2], [0    3], [28   4], [5    5], [36   6]]
[25   1], [4    2], [0    3], [68   4], [49   5], [41   6]]
[26   1], [0    2], [69   3], [8    4], [46   5], [61   6]]
[35   1], [61   2], [0    3], [72   4], [49   5], [11   6]]
[82   1], [12   2], [15   3], [62   4], [0    5], [42   6]]
```

μ_1, σ_1^2 μ_2, σ_2^2 μ_3, σ_3^2 μ_4, σ_4^2 μ_5, σ_5^2 μ_6, σ_6^2

Seleccionar n individuos de P utilizando *Selección por torneo*

Estimar μ_j y σ_j^2 para cada tiempo de retardo de cada Trabajo j

$P \leftarrow TN(\mu_j, \sigma_j^2, 0, Max\ Start)$ ▷ TN Distribución Normal Truncada

end for

Algorithm 2 MAKE FEASIBLE SOLUTION

```
Ordenar el cromosoma c = ((d_1, J_1), (d_2, J_2), ... (d_n, J_n)) de acuerdo al tiempo de retardo
    d_i con criterio de desempate J_i
while No se hayan calendarizado todos los trabajos do
    Intentar calendarizar el trabajo J_i con tiempo de retardo d_i y verificar colisiones
    if No existe colisiones then
        Continuar con el trabajo J_{i+1}
    else
        Calendarizar J_i lo mas cercano a 0
        Actualizar d_i para el trabajo J_i
    end if
end while
```

Configuración Experimental

Lenguaje de programación: Python

Versión: 3.10.4

Sistema operativo: Arch Linux

Kernel: 5.17.1

Procesador: Ryzen 5 5600g, 3.9 GHz

Memoria RAM: 16GB DDR4

Configuración Experimental

El algoritmo fue evaluado en 5 instancias:

- ft06 (6 x 6)
- ft10(10 x 10)
- la05 (10 x 5)
- la33 (30 x 10)
- la40 (15 x 15)

de Lawrence (1984)[2] y Fisherman & Thompson (1963)[3]. Para cada instancia se realizaron 20 ejecuciones.

Configuración Experimental

El rendimiento del algoritmo se midió utilizando:

- Desviación relativa porcentual PRD (Percentage Relative Deviation)

$$PRD = \frac{Best_{alg} - BKS}{BKS} \times 100\%$$

- Desviación Relativa Porcentual Promedio APRD (Average Percentage Relative Deviation)

$$APRD = \frac{Avg_{alg} - BKS}{BKS} \times 100\%$$

Configuración Experimental

Donde:

- BKS , es la mejor solución conocida del problema.

- $Best_{alg}$, corresponde a la mejor solución generada por el algoritmo de todas las ejecuciones.

- Avg_{alg} , corresponde al promedio de las mejores soluciones generadas por el algoritmo.

Resultados

Nombre	(n,m)	BKS	Avg Value	Std Value	Avg Time	Std Time	Best	PRD	APRD
ft06	(6,6)	73 [4]	73.45	1.56	33.76s	3.58s	73	0.00	0.62
la05	(10,5)	777 [4]	995.40	20.63	69.42s	3.01s	954	22.78	28.11
ft10	(10,10)	1607 [4]	1871.15	41.35	139.26s	3.37s	1814	12.88	16.44
la40	(15,15)	2580 [5]	3719.40	170.14	417.52s	64.77s	3444	33.49	44.16
la33	(30,10)	3413 [5]	12639.05	439.02	331.22s	25.36s	11733	243.77	270.32

Tabla 1. Resultados obtenidos por el algoritmo

Conclusión

El algoritmo parece funcionar en instancias pequeñas como se pudo observar al probarlo en la instancia “ft06” de tamaño 6×6, donde además se pudo observar que requirió un tamaño de población pequeño y a su vez pocas generaciones, lo cual se traduce a pocas llamadas de la función de evaluación de aptitud. Sin embargo en problemas medianos y grandes los resultados obtenidos no fueron los esperados, obteniendo valores de PRD entre 12 % y 33 % para problemas medianos, e inclusive llegando a valores por encima de los 240 % en instancias grandes, lo cual se traduce que el algoritmo obtuvo resultados significativamente lejanos a la mejor solución conocida.

Referencias

[1] V. M. Valenzuela-Alcaraz, M. Cosío-León, A. D. Romero-Ocaño, and C. A. Brizuela, “A cooperative coevolutionary algorithm approach to the no-wait job shop scheduling problem,” Expert Systems with Applications, vol. 194, p. 116498, 2022.

[2] S. Lawrence, “An experimental investigation of heuristic scheduling techniques (supplement),” Graduate School of Industrial Administration, Carnegie-Mellon University, 1984.

[3] H. Fisher and G. L. Thompson, “Probabilistic learning combinations of local job-shop scheduling rules,” Industrial Scheduling, pp. 225-251, 1963.

[4] A. Mascis and D. Pacciarelli, “Job-shop scheduling with blocking and no-wait constraints,” European Journal of Operational Research, vol. 143, pp. 498–517, 2002.

[5] K.-C. Ying and S.-W. Lin, “Solving no-wait job-shop scheduling problems using a multi-start simulated annealing with bi-directional shift timetabling algorithm,” Computers and Industrial Engineering, vol. 146, pp. 498–517, 2020.