Heurísticas de búsqueda local para el problema de programación de tareas

Maria Alejandra Soriano Castañeda

Universidad de los Andes
Cra. 1 #18a-12, Bogotá, Cundinamarca
ma.soriano@uniandes.edu.co

Abstract

This article presents two local search heuristics for the task scheduling problem in order to minimize the total execution time of all tasks. To validate the results obtained by implementing the local search algorithms, the ten instances presented in Taillard (1993) are used as test cases, from the instance tai500_20_1 to tai500_20_10. Subsequently, the results are compared with the upper limit available for the instances of Taillard (1993). The computational results demonstrate the effectiveness of the heuristics and the improvements implemented.

Keywords—scheduling; heuristics; flow shop.

1 Introducción

El problema de programación de tareas, comúnmente referenciado en inglés como Flow Shop, es un problema de optimización combinatoria que puede ser categorizado como un problema de difícil solución y está clasificado técnicamente como de solución en un tiempo no polinomial (NP-Hard) (Najarro et al., 2017). En general, la complejidad del problema radica en la gran cantidad de posibles soluciones. El problema de Flow Shop consiste en asignar tareas o trabajos a un conjunto limitado de recursos disponibles o maquinas durante un intervalo de tiempo, donde se busca optimizar algún criterio relevante como el makespan (el tiempo mínimo para completar todos los trabajos) (Fonseca et al., 2015). El *Flow Shop Scheduling Problem* es un caso particular de los problemas de secuenciación de tareas o problemas de scheduling, ya que el orden en que se programan los trabajos es el mismo para cada máquina pese a que los tiempos de procesamiento de cada trabajo puede ser diferente en cada máquina.

A lo largo de los años se han propuesto y validado diversas heurísticas y meta-heurísticas para hacer frente a los problemas de programación de tareas, puesto que este problema en adición de diversas restricciones funciona como base para problemas más complejos con características más parecidas a las evidenciadas en la industria.

El objetivo de las heurísticas de búsqueda local que se van a presentar es definir una programación factible que programé n trabajos en m maquinas tal que se minimice el tiempo mínimo para procesar todos los trabajos. El problema que se va a abordar se define considerando que ninguna maquina puede procesar más de un trabajo a la vez y que las operaciones en las maquinas no se pueden interrumpir una vez sean iniciadas. Además, se considera que el tiempo de traslado de los trabajos entre maquinas no es significativo, la secuencia en que se programan los trabajos es la misma para todas las máquinas y no hay restricciones de precedencia entre trabajos (Fonseca et al., 2015).

2 Descripción del problema

En el problema de Flow shop que se propone analizar, un conjunto de n trabajos deben ser procesados en m maquinas diferentes en el mismo orden, con tiempos de procesamiento no negativos. De manera general, el problema planteado puede formularse matemáticamente como

sigue (Goggel, 2020).

$$Minimizar\ C_{max} \tag{1}$$

Sujeto a:

$$\sum_{j=1}^{n} X_{jk} = 1, \qquad \forall\, k \in J \tag{2}$$

$$\sum_{k=1}^{n} X_{jk} = 1, \qquad \forall\, j \in J \tag{3}$$

$$\sum_{j=1}^{n} X_{j,k+1}\left(p_{ji}\right) \cdot M_{ji} - \sum_{j=1}^{n} X_{jk}\left(p_{j,i+1}\right) \cdot M_{j,i+1} = 0\,, \quad \forall \mathrm{j} \in \mathrm{J}, \mathrm{k} = 1, \ldots, \mathrm{n}-1; \mathrm{i} = 1, \ldots, \mathrm{m}-1 \tag{4}$$

$$C_{max} = \sum_{i=1}^{m}\sum_{j=1}^{n}\left(p_{ji} \cdot X_{jk}\right) \cdot M_{ji} \tag{5}$$

$$X_{jk} \in \{0,1\}, \qquad \forall i, j \in J \tag{6}$$

$$M_{ji} \in \{0,1\}, \qquad \forall i \in M, j \in J \tag{7}$$

Por un lado, la Ecuación 1 presenta la función objetivo del problema, la cual busca minimizar el *Makespan* (tiempo de terminación del último trabajo procesado en la última máquina). Además, en las Ecuaciones 2-6 se presentan las restricciones que limitan el problema, la segunda restricción asegura que cada posición *k* de la secuencia solo tenga un trabajo *j* asignado, la Restricción 3 asegura a que cada trabajo se encuentre únicamente en una posición de la secuencia, la Restricción 4 establece que la diferencia entre el tiempo cuando el trabajo en la posición *k* en la máquina *i*+1 y el tiempo en el que el *k* termina en la máquina *i.* Por otro lado, la Ecuación 5 establece como calcular el Makespan y, por último, las Restricciones 6 y 7 definen la naturaleza binaria de las variables.

Por otro lado, los parámetros del modelo presentado y la definición de las variables se definen como:

n: Numero de trabajos.
m: Número de máquinas.
i: Índice de máquinas. *i*=1…*m*
j: Índice de los trabajos. *j*=1…*n*
k: Índice de posición de los trabajos. *k*=1…*n*
J: Conjunto de trabajos
M: Conjunto de máquinas.
p_{ij}: Tiempo de procesamiento del trabajo *j* en la maquina *i*
X_{ij}: {1: El trabajo *j* es el k-esimo en la secuencia, 0: d.l.c}
M_{ij}:{1: El trabajo *j* se puede procesar en la máquina *i*, 0: d.l.c}

3 Metodología

Con el objetivo de determinar la calidad de los resultados que se pueden obtener al implementar algoritmos heurísticos en la programación de tareas, se realizaron dos algoritmos basados en algoritmos de búsqueda local, uno utilizando un intercambio denominado swap y el otro basado en la inserción de un trabajo en todas las posiciones posibles de la secuencia. De manera general, los algoritmos de búsqueda local se caracterizan por quedar atrapados fácilmente en óptimos locales y por llegar a soluciones de manera rápida. Sin embargo, son ahorrativos puesto que solo van guardando la mejor solución encontrada y no todas las soluciones evaluadas.

Posteriormente, se realizó un análisis experimental tomando como referencia diez instancias de Taillard (1993) las cuales poseen una dimensión de 500 trabajos y 20 máquinas.

3.1 Intercambio de trabajos programados

Teniendo en cuenta que la solución inicial está dada por el orden lexicográfico de los trabajos, en esta heurística se propone intercambiar las posiciones de los trabajos e ir evaluando que posición de cada trabajo favorece más a la función objetivo para minimizar el *makespan*. Cabe resaltar que conocemos el tiempo de procesamiento de cada trabajo en cada máquina, por lo cual se puede estimar el tiempo de procesamiento de toda la secuencia. Este algoritmo comienza seleccionando los dos primeros elementos de la solución inicial (1, 2, 3, 4, …, 500), es decir, selecciona los trabajos uno y dos e invierte su posición dentro de la secuencia. Posteriormente, el algoritmo calcula y guarda la función objetivo de la nueva secuencia generada, a partir de esto, el algoritmo comienza una nueva iteración seleccionando los siguientes dos trabajos de la mejor secuencia encontrada hasta el momento. Este procedimiento se continua iterativamente hasta evaluar en cada posición un intercambio con el trabajo que se encuentre en la posición a su derecha, de esta manera, en la última iteración del algoritmo se intercambiaría el trabajo ubicado en la posición 499 con el trabajo que se encuentre al final de la secuencia. A modo de ejemplo, si en la primera iteración el menor *Makespan* se obtenía de la secuencia (2, 1, 3, 4, …,500), en la segunda iteración se seleccionarían los siguientes trabajos consecutivos (trabajo uno y trabajo tres) puesto que se toma como referencia la secuencia que presenta un menor valor de *Cmax* seleccionada en la iteración anterior.

Algoritmo constructivo 1

Input: Secuencia inicial y tiempo de procesamiento de cada trabajo en cada máquina
(1) SolucionActual = valor de la función objetivo de la secuencia con orden lexicográfico
(2) **for** i **in** range(0, len(secuencia))
(3) first = Trabajo que se encuentra en la posición i de la secuencia actual
(4) Second = Trabajo en la posición $i + 1$ de la secuencia actual
(5) Intercambiar posiciones de los trabajos first y second:
(6) SolucionesF0={}
(7) F_obj = Función objetivo de la nueva secuencia
(8) SolucionesF0[F_obj] = secuencia probada
(9) **if** min(SolucionesF0) < SolucionActual **then**
(10) SolucionActual = min(Soluciones F0)
(11) **else:**
(12) Intercambiar las posiciones de los trabajos first y second.
(13) **end if**
(14) **Next** i
(15)
Return Mejor secuencia encontrada

3.2 Inserción secuencial

La segunda heurística propuesta, consta de encontrar la mejor posición de un trabajo a partir de una secuencia previamente seleccionada. Este algoritmo se basa en el principio de mejor mejora puesto que para cada trabajo primero calcula la función objetivo en cada posición, guarda los resultados de las 500 iteraciones que se realizó para un trabajo, y posteriormente, selecciona la mejor secuencia encontrada para el trabajo que está siendo evaluado. A partir de la mejor secuencia encontrada para un trabajo, se procede a seleccionar al siguiente trabajo de la secuencia inicial y a evaluar la función objetivo del trabajo en todas las posibles. Una de las ventajas de este algoritmo es que evalúa cada posición de cada trabajo en todo el vecindario de posibles ubicaciones y selecciona la posición que favorece más a la función objetivo. No obstante, esta heurística requiere de mayor esfuerzo computacional.

Después de realizar todas las corridas de ambos algoritmos con las instancias de Taillard, se evidencio que el segundo algoritmo tuvo un mejor desempeño en todos los experimentos realizados, por lo cual se propuso una mejora a este algoritmo incluyendo una modificación en la solución inicial, dado que las soluciones finales de los algoritmos de búsqueda local dependen fuertemente de su solución inicial. De este modo, se incluyeron mejoras al algoritmo de búsqueda local basadas en la heurística NEH, la cual inicia sus iteraciones con una secuencia que depende del tiempo de procesamiento total para cada trabajo, empezando al inicio de la secuencia con el trabajo con mayor tiempo total de procesamiento en todas las maquinas.

Algoritmo constructivo 2

Input: Secuencia inicial y tiempo de procesamiento de cada trabajo en cada máquina
(1) SolucionActual = valor de la función objetivo de la secuencia con orden lexicográfico
(2) **for** x **in** trabajos
(3) SolucionesF0 = {}
(4) Eliminar trabajo x de la secuencia
(5) **for** y **in** range (0, 500)
(6) Insertar el trabajo x en la posición y de la secuencia actual
(7) F_obj = Función objetivo de la nueva secuencia
(8) Eliminar el trabajo x en la posición y de la secuencia actual
(9) **next** y
(10) Secuencia = SolucionesF0[min(F_obj)]
(11) **Next** x
Return Mejor secuencia encontrada

4 Experimentos computacionales y análisis de resultados.

La implementación de las heurísticas presentadas se realizó en el lenguaje de programación Python versión 5.1.5 y el desempeño de los algoritmos fue testeado con las instancias mencionadas interiormente las cuales contemplan: un set de 500 trabajos y 20 máquinas. Los resultados de las diferentes iteraciones para cada algoritmo se pueden observar en la tabla 1. A partir de esta, se puede evidenciar que del segundo método propuesto en el artículo presenta un mejor desempeño que el método basado en el swap sencillo. Con un GAP promedio del 4.15% en comparación con el *BKS* de las instancias de Taillard, se puede afirmar que el segundo algoritmo es una buena alternativa a la hora de resolver problemas de programación de tareas, sin embargo, se recomienda mejorar la solución inicial puesto que de esta manera se comprobó que se pueden obtener mejores resultados a un menor tiempo computacional.

Table 1
Comparación entre resultados de distintos algoritmos para el problema de Flow Shop

Instance (tai500_20)	BKS	Cmax Orden Lexicografico	Swap		Permutación		NEH		GAP*Swap*	GAP-*Perm.*	GAP*NEH*
			Cmax	CPU(min)	Cmax	CPU(min)	Cmax	CPU(min)			
1	26,040	30,121	29,758	0.26	27,319	120.54	26,774	41.55	14.28%	4.91%	2.82%
2	26,520	31,202	30,801	0.28	27,815	139.03	27,215	42.30	16.14%	4.88%	2.62%
3	26,371	30,447	30,210	0.25	27,406	89.30	26,941	30.04	14.56%	3.92%	2.16%
4	26,456	30,355	30,043	0.23	27,372	119.98	26,928	42.72	13.56%	3.46%	1.78%
5	26,334	30,099	29,919	0.36	27,308	90.09	26,928	31.09	13.61%	3.70%	2.26%
6	26,477	30,946	30,676	0.30	27,732	48.51	27,047	31.31	15.86%	4.74%	2.15%
7	26,389	30,792	30,316	0.23	27,249	90.76	26,820	41.49	14.88%	3.26%	1.63%
8	26,560	31,034	30,853	0.27	27,637	125.62	27,230	40.74	16.16%	4.05%	2.52%
9	26,005	30,634	30,259	0.33	27,200	91.77	26,541	41.86	16.36%	4.60%	2.06%
10	26,457	30,148	29,927	0.30	27,499	89.20	27,103	41.33	13.12%	3.94%	2.44%
Average		30,578	30,276	0.28	27,454	100.48	26,953	38.44	14.85%	4.15%	2.25%

5 Conclusión

A partir de los resultados recolectados para cada instancia y para cada algoritmo, podemos concluir que los algoritmos propuestos nos permiten obtener soluciones factibles ante situaciones del mundo real y de grande dimensión. Sin embargo, no se puede asegurar que los valores obtenidos sean óptimos globales, ya que los algoritmos de búsqueda local pueden quedar fácilmente atrapados en óptimos locales.

Adicionalmente, se llega a la conclusión que utilizar la heurística NEH en los casos de alta complejidad presenta grandes ventajas ya que se reducen los costos de computación. Finalmente, se puede afirmar que se cumplió con el objetivo principal del artículo de diseñar dos algoritmos de búsqueda local con el fin de encontrar una solución factible al problema de programación de tareas.

Apendice A. Mejores soluciones encontradas para las instancias comparativas de Taillard

Mejor solución encontrada para ta01:
n = 500, m = 20, Cmax = 26774
Secuencia = [145, 288, 106, 485, 158, 215, 407, 242, 476, 497, 183, 2, 268, 216, 88, 79, 156, 29, 43, 327, 58, 227, 408, 192, 261, 228, 217, 62, 90, 16, 32, 390, 293, 278, 411, 208, 452, 481, 19, 374, 499, 279, 406, 350, 36, 159, 238, 341, 364, 56, 322, 344, 398, 224, 401, 264, 225, 325, 354, 256, 205, 434, 309, 111, 369, 104, 116, 400, 113, 189, 257, 399, 467, 311, 410, 376, 240, 496, 378, 96, 296, 78, 17, 193, 169, 5, 129, 285, 69, 303, 210, 433, 161, 154, 48, 421, 86, 59, 333, 370, 432, 31, 110, 74, 409, 1, 449, 166, 283, 319, 386, 6, 186, 381, 11, 102, 46, 404, 150, 492, 358, 299, 201, 81, 207, 164, 275, 494, 457, 301, 488, 483, 422, 484, 99, 115, 53, 298, 101, 326, 271, 176, 35, 107, 355, 248, 297, 100, 480, 167, 258, 247, 418, 143, 310, 478, 337, 474, 105, 351, 243, 76, 97, 157, 68, 362, 180, 45, 415, 439, 363, 254, 37, 274, 141, 206, 230, 51, 195, 397, 335, 367, 179, 220, 475, 464, 498, 353, 253, 188, 236, 50, 202, 334, 91, 252, 10, 500, 493, 402, 272, 149, 287, 339, 460, 199, 420, 330, 7, 394, 144, 352, 221, 302, 4, 423, 307, 203, 435, 389, 231, 128, 371, 22, 451, 98, 38, 162, 20, 446, 466, 454, 383, 286, 148, 431, 65, 372, 172, 200, 470, 465, 8, 313, 487, 375, 25, 153, 187, 440, 348, 178, 262, 117, 441, 80, 168, 456, 182, 118, 416, 34, 306, 165, 429, 250, 44, 89, 66, 24, 479, 462, 235, 60, 139, 342, 284, 197, 304, 196, 491, 237, 190, 316, 259, 273, 211, 77, 173, 114, 82, 213, 346, 276, 331, 294, 414, 146, 270, 396, 424, 152, 490, 23, 92, 295, 320, 377, 382, 360, 368, 343, 461, 175, 42, 147, 64, 419, 124, 373, 265, 73, 477, 41, 177, 289, 249, 486, 233, 312, 356, 308, 132, 290, 347, 63, 87, 323, 214, 443, 447, 444, 9, 219, 365, 413, 277, 191, 57, 142, 385, 39, 130, 212, 70, 209, 245, 384, 391, 134, 120, 359, 328, 349, 338, 472, 13, 438, 229, 357, 93, 136, 329, 260, 83, 126, 171, 292, 305, 317, 280, 471, 155, 437, 49, 109, 218, 468, 40, 27, 442, 455, 151, 246, 127, 412, 314, 403, 263, 267, 61, 392, 174, 251, 184, 425, 121, 28, 324, 160, 426, 72, 3, 336, 138, 123, 332, 405, 47, 185, 318, 52, 473, 163, 244, 170, 436, 103, 239, 30, 122, 489, 135, 14, 495, 234, 361, 54, 198, 226, 380, 291, 453, 131, 12,

108, 232, 388, 395, 448, 85, 482, 95, 427, 67, 282, 379, 125, 181, 266, 255, 140, 458, 428, 469, 450, 222, 137, 18, 445, 75, 459, 315, 15, 463, 387, 119, 366, 26, 269, 133, 21, 345, 241, 321, 204, 194, 33, 340, 84, 94, 71, 281, 223, 55, 112, 393, 430, 417, 300]

Mejor solución encontrada para ta02:
n = 500, m = 20, Cmax = 27,215
Secuencia = [8, 326, 97, 159, 148, 475, 244, 305, 47, 348, 378, 282, 178, 230, 55, 4, 62, 365, 127, 446, 343, 350, 150, 311, 302, 472, 253, 254, 162, 88, 429, 362, 186, 412, 93, 358, 452, 114, 204, 49, 101, 490, 468, 79, 403, 117, 64, 12, 74, 442, 349, 438, 415, 314, 105, 356, 424, 90, 10, 152, 409, 145, 227, 81, 481, 116, 375, 480, 265, 199, 65, 188, 189, 291, 153, 318, 416, 498, 335, 243, 85, 120, 308, 324, 347, 134, 187, 386, 132, 427, 405, 224, 258, 164, 397, 171, 432, 173, 495, 59, 196, 119, 325, 6, 206, 455, 166, 376, 421, 198, 133, 341, 331, 236, 297, 2, 462, 185, 1, 453, 366, 278, 264, 183, 379, 385, 285, 422, 338, 313, 321, 181, 9, 240, 315, 100, 72, 404, 137, 24, 209, 292, 272, 340, 361, 497, 207, 393, 156, 135, 66, 256, 96, 154, 235, 22, 15, 417, 459, 214, 369, 448, 402, 492, 456, 21, 469, 293, 149, 83, 69, 300, 320, 111, 478, 355, 57, 410, 494, 306, 172, 330, 487, 363, 158, 91, 140, 389, 238, 203, 61, 239, 319, 426, 43, 342, 45, 450, 109, 71, 377, 112, 222, 384, 157, 428, 192, 277, 329, 274, 491, 217, 275, 263, 303, 131, 470, 268, 216, 5, 195, 344, 25, 367, 234, 37, 368, 107, 225, 296, 14, 476, 289, 345, 400, 200, 486, 23, 170, 124, 328, 249, 370, 353, 283, 466, 460, 380, 165, 394, 184, 92, 290, 219, 304, 3, 294, 94, 142, 68, 182, 113, 451, 146, 346, 484, 161, 261, 322, 18, 42, 287, 7, 471, 334, 215, 226, 425, 435, 250, 168, 401, 80, 474, 391, 383, 418, 118, 257, 84, 309, 441, 359, 374, 449, 208, 115, 167, 151, 259, 110, 396, 266, 41, 489, 220, 440, 201, 499, 122, 58, 103, 500, 210, 463, 390, 372, 248, 273, 477, 228, 98, 352, 251, 357, 245, 179, 11, 479, 298, 485, 13, 246, 136, 317, 255, 286, 310, 174, 155, 52, 288, 78, 223, 29, 16, 336, 339, 382, 121, 99, 247, 295, 337, 381, 407, 233, 276, 312, 260, 281, 431, 104, 89, 445, 301, 67, 364, 123, 461, 46, 413, 169, 467, 465, 241, 194, 419, 423, 191, 176, 17, 354, 360, 130, 70, 60, 40, 180, 54, 31, 126, 106, 30, 51, 392, 125, 458, 163, 50, 128, 190, 411, 437, 252, 483, 284, 433, 414, 33, 76, 129, 270, 351, 213, 269, 175, 73, 39, 327, 218, 373, 139, 323, 86, 77, 202, 177, 436, 493, 496, 399, 229, 457, 443, 87, 464, 279, 430, 473, 19, 212, 138, 231, 20, 262, 388, 35, 108, 205, 434, 144, 488, 395, 34, 439, 147, 221, 53, 398, 232, 267, 387, 447, 38, 141, 454, 211, 444, 32, 271, 26, 482, 197, 299, 408, 333, 27, 28, 143, 95, 160, 82, 56, 102, 307, 332, 237, 36, 316, 280, 371, 75, 420, 48, 242, 44, 406, 193, 63]

Mejor solución encontrada para ta03:
n = 500, m = 20, Cmax = 26,941
Secuencia = [431, 31, 38, 162, 443, 99, 486, 144, 396, 389, 74, 50, 347, 310, 210, 434, 66, 487, 364, 207, 2, 316, 69, 159, 90, 238, 425, 372, 253, 303, 409, 403, 464, 46, 342, 64, 197, 170, 56, 186, 483, 493, 473, 88, 245, 225, 344, 433, 91, 435, 222, 48, 460, 73, 499, 19, 76, 3, 5, 376, 477, 87, 124, 97, 147, 286, 388, 96, 21, 67, 44, 284, 163, 81, 184, 317, 339, 229, 395, 59, 369, 30, 446, 249, 275, 368, 204, 118, 274, 309, 177, 34, 70, 498, 14, 390, 436, 423, 463, 354, 108, 78, 45, 181, 155, 426, 392, 256, 116, 114, 333, 444, 47, 424, 154, 53, 157, 92, 102, 417, 86, 227, 252, 294, 109, 406, 328, 40, 8, 107, 399, 105, 470, 117, 129, 187, 180, 234, 194, 438, 241, 89, 393, 308, 1, 137, 200, 54, 305, 335, 250, 278, 130, 236, 468, 254, 18, 240, 322, 429, 191, 306, 359, 320, 226, 161, 265, 500, 179, 299, 271, 482, 318, 215, 246, 471, 314, 62, 82, 134, 408, 136, 290, 411, 110, 447, 302, 221, 211, 33, 55, 490, 93, 23, 458, 270, 79, 450, 13, 385, 25, 476, 15, 156, 104, 264, 174, 304, 334, 382, 462, 83, 484, 412, 404, 379, 172, 42, 216, 267, 479, 242, 61, 327, 415, 330, 398, 288, 37, 292, 280, 387, 17, 375, 496, 24, 152, 217, 171, 173, 321, 11, 331, 223, 378, 123, 338, 237, 75, 355, 65, 449, 402, 282, 313, 356, 400, 349, 337, 228, 374, 39, 175, 10, 407, 20, 139, 132, 68, 195, 220, 455, 495, 283, 113, 183, 261, 149, 219, 4, 345, 98, 340, 80, 100, 125, 488, 203, 420, 326, 121, 262, 301, 285, 336, 135, 348, 272, 214, 251, 148, 9, 414, 494, 485, 358, 383, 341, 26, 346, 111, 206, 319, 370, 289, 94, 381, 258, 36, 489, 384, 95, 189, 218, 231, 480, 244, 115, 131, 138, 212, 377, 421, 351, 352, 150, 60, 72, 268, 185, 307, 332, 146, 416, 213, 141, 325, 401, 371, 419, 439, 58, 63, 491, 119, 276, 202, 291, 427, 224, 142, 35, 394, 160, 452, 367, 410, 492, 7, 466, 128, 188, 430, 43, 297, 182, 478, 422, 442, 311, 405, 178, 167, 465, 437, 413, 353, 397, 239, 247, 166, 440, 235, 84, 298, 248, 192, 22, 169, 453, 461, 168, 474, 199, 122, 365, 49, 266, 300, 312, 230, 151, 106, 428, 357, 243, 27, 51, 361, 16, 193, 497, 281, 260, 343, 391, 451, 85, 269, 363, 324, 366, 127, 287, 448, 373, 126, 6, 380, 441, 360, 205, 259, 257, 457, 467, 295, 153, 315, 323, 273, 209, 362, 481, 164, 140, 293, 350, 469, 29, 386, 198, 233, 158, 432, 143, 145, 165, 296, 277, 329, 232, 176, 190, 52, 57, 120, 12, 208, 112, 28, 133, 32, 459, 103, 71, 41, 77, 445, 196, 454, 456, 418, 279, 263, 201, 472, 101, 255, 475]

Mejor solución encontrada para ta04:
n = 500, m = 20, Cmax = 26,928
Secuencia = [151, 1, 165, 191, 402, 342, 142, 442, 117, 437, 126, 289, 426, 240, 325, 413, 171, 443, 273, 98, 113, 148, 383, 203, 234, 11, 272, 30, 391, 278, 276, 244, 283, 327, 268, 375, 392, 57, 193, 172, 470, 219, 216, 471, 468, 41, 336, 68, 388, 485, 40, 58, 367, 67, 267, 223, 479, 185, 431, 356, 102, 226, 386, 296, 277, 133, 213, 270, 149, 103, 366, 231, 264, 154, 180, 44, 333, 419, 90, 127, 398, 21, 297, 300, 174, 347, 448, 49, 309, 331, 416, 73, 224, 258, 131, 395, 99, 66, 161, 135, 56, 254, 45, 408, 110, 146, 245, 280, 415, 348, 410, 457, 473, 369,

28, 212, 412, 433, 326, 308, 303, 424, 128, 316, 25, 423, 436, 184, 116, 136, 200, 338, 20, 210, 418, 156, 387, 261, 455, 169, 50, 92, 157, 445, 225, 257, 18, 241, 72, 214, 9, 164, 306, 259, 250, 4, 441, 464, 314, 459, 199, 78, 343, 488, 33, 42, 81, 332, 384, 290, 466, 162, 218, 59, 170, 75, 238, 321, 301, 322, 202, 472, 47, 34, 134, 496, 36, 310, 10, 192, 353, 409, 292, 489, 208, 247, 450, 12, 69, 6, 153, 233, 469, 494, 467, 499, 315, 371, 77, 404, 251, 80, 70, 84, 344, 425, 446, 483, 281, 43, 215, 205, 262, 462, 125, 376, 377, 63, 31, 330, 204, 189, 167, 340, 123, 201, 222, 232, 452, 358, 122, 237, 65, 188, 256, 320, 482, 83, 13, 405, 465, 107, 221, 24, 337, 27, 105, 145, 130, 177, 401, 119, 378, 380, 319, 451, 269, 129, 477, 229, 406, 288, 14, 246, 100, 242, 94, 178, 108, 421, 287, 64, 486, 249, 137, 48, 2, 350, 323, 91, 114, 54, 444, 252, 96, 87, 447, 282, 143, 481, 417, 411, 286, 285, 394, 368, 88, 374, 243, 275, 60, 430, 349, 351, 230, 357, 46, 198, 422, 155, 32, 176, 16, 86, 112, 381, 85, 393, 197, 435, 52, 181, 354, 235, 71, 305, 37, 195, 372, 389, 440, 147, 284, 341, 255, 352, 313, 179, 209, 55, 438, 217, 493, 109, 293, 360, 62, 355, 475, 271, 396, 152, 329, 7, 498, 239, 492, 478, 76, 168, 407, 23, 490, 173, 236, 104, 484, 304, 474, 196, 295, 362, 382, 101, 328, 19, 390, 175, 497, 97, 463, 144, 397, 79, 253, 53, 111, 274, 29, 429, 365, 324, 187, 302, 400, 95, 211, 476, 298, 82, 279, 115, 339, 312, 89, 220, 491, 454, 150, 379, 346, 183, 140, 8, 432, 139, 158, 370, 39, 74, 265, 118, 311, 439, 124, 335, 93, 399, 318, 363, 420, 453, 449, 263, 359, 364, 317, 207, 15, 294, 138, 248, 385, 166, 334, 35, 17, 227, 427, 121, 194, 291, 26, 500, 487, 141, 182, 106, 260, 480, 403, 206, 228, 3, 120, 159, 434, 132, 461, 38, 458, 460, 186, 299, 61, 307, 5, 266, 361, 22, 51, 373, 414, 456, 160, 428, 163, 495, 190, 345]

6 References

Najarro, R., López, R., Racines, R. E., & Puris, A. (2017). Un Algoritmo Genético Híbrido para la Optimización del Flow Shop Scheduling bajo Restricciones de Entornos Reales. *Enfoque UTE*, *8*(5), 14–25. https://doi.org/10.29019/ENFOQUEUTE.V8N5.176

Fonseca, Y., Martínez, M., Bermúdez, J., y Méndez, B. (2015). A REINFORCEMENT LEARNING APPROACH FOR SCHEDULING PROBLEMS. Investigación Operacional, 36(3), 225–231. http://www.invoperacional.uh.cu/index.php/InvOp/article/view/482

Goggel, D. (2020). SOLUCIÓN AL PROBLEMA DE LA PROGRAMACIÓN DE LA PRODUCCIÓN EN UNA PLANTA COMPUESTA POR ÁREAS INTERRELACIONADAS.