Trabajo Fin de Máster
Máster Universitario en Ingeniería Industrial

Programación de la producción para problemas de flowshop con dos conjuntos de trabajos

Autor: Antonio Gutiérrez González
Tutor: Paz Pérez González

Dpto. de Organización Industrial y Gestión de Empresas I
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Máster
Máster Universitario en Ingeniería Industrial

Programación de la producción para problemas de flowshop con dos conjuntos de trabajos

Autor:
Antonio Gutiérrez González

Tutor:
Paz Pérez González
Profesora Titular de Universidad

Dpto. de Organización Industrial y Gestión de Empresas I
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Trabajo Fin de Máster: Programación de la producción para problemas de flowshop con dos conjuntos de trabajos

Autor: Antonio Gutiérrez González

Tutor: Paz Pérez González

El tribunal nombrado para juzgar el Trabajo arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocal/es:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

Resumen

La programación de la producción es un área de interés industrial debido a la gran productividad que se puede llegar a alcanzar en la actividad desarrollada, así como por la reducción de los costes asociados. Se trata de la asignación de recursos en el tiempo para realizar un conjunto de tareas con el fin de optimizar un objetivo específico.

En este Trabajo Fin de Máster se aborda un problema de programación de la producción en un entorno de trabajo de taller de flujo regular (*flowshop*). Consiste en dos máquinas en serie y dos conjuntos de trabajos diferentes, cuyos objetivos son minimizar el tiempo total de finalización de los trabajos del primer conjunto sin tener trabajos retrasados en el segundo conjunto.

Para analizar y resolver el problema se propone un método exacto y otro aproximado. El modelo exacto es un modelo matemático de programación lineal entera, con el objetivo de obtener soluciones óptimas. El método aproximado se basa una técnica metaheurística fundamentada en el algortimo *Simulated Annealing*, para resolver el problema con buena calidad en aquellos tamaños en los que el modelo exacto no puede encontrar solución, ya que se trata un problema de alta complejidad computacional.

Abstract

Production scheduling is an area of industrial interest due to the high productivity that can be achieved in the activity carried out, as well as the reduction in associated costs. It is about the allocation of resources over time to perform a colletion of tasks in order to optimize a specific objective.

This Master's Final Project adresses a production scheduling problem in a flowshop environment. It consists of two machines in series and two different sets of jobs, the objectives of which are to minimize the total completion time of the jobs in the first set without having tardy jobs in the second set.

To analyze and solve the problem, an exact and an approximate method are proposed. The exact method is an integer linear programming mathematical model, with the aim of obtaining optimal solutions. The approximate method is based on a metaheuristic technique grounded in the Simulated Annealing algorithm, to solve the problem with good quality for those sizes in which the exact model cannot find a solution, since it is a problem of high computational complexity.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE GRÁFICAS

Notación

:	Tal que
≠	Diferente de
<	Menor o igual
≤	Menor o igual que
>	Mayor o igual
≥	Mayor o igual que
∑	Sumatorio
∪	Unión
∩	Intersección
⌀	Conjunto vacío
∈	Pertenece a
∀	Para todo
…	Progresión
!	Factorial de
→	Implica
∆	Diferencia
ℝ	Conjunto de los números reales

1 INTRODUCCIÓN

En este Trabajo Fin de Máster se hace un estudio relacionado con la programación de la producción, problema común en muchos sectores industriales, en el que las empresas deben tomar decisiones en cuanto a la asignación de los recursos existentes a una serie de actividades. Dichas actividades deben ser ordenadas en secuencia, ubicándolas en el espacio temporal, determinando cuando empieza y acaba cada actividad, de manera que se optimice uno o varios objetivos. La secuenciación no es una tarea fácil en la práctica, ya que su aplicación se debe adaptar y modelar de manera correcta a los entornos en los que se aplica, como los complejos entornos productivos. Sin embargo, es de gran importancia, pues se trata de optimizar la producción, tratando de lograr altos niveles de productividad mediante la reducción de los costes asociados.

En este enfoque de la programación las actividades se denominan trabajos, que son realizados con los recursos, que se denominan máquinas. Existen multitud de escenarios dentro de la producción, sin embargo, en este documento, se plantea el escenario de un entorno de trabajo de taller de flujo regular de permutación (conocido como *permutation flowshop*), es decir, máquinas en serie en las que todos los trabajos tienen que ser procesados por todas las máquinas en el mismo orden de proceso. En este estudio se parte de la premisa de que los trabajos no se pueden interrumpir, por tanto, una vez que están siendo procesados en una máquina deben terminar en sus correspondientes tiempos de proceso.

En este documento que se desarrolla se plantean dos conjuntos de trabajos distintos a procesar. Desde que Smith y Baker (2003) y Agnetis, Mirchandani, Pacciarelli y Pacifici (2004) introdujeron la programación de la producción con dos conjuntos de trabajos, la atención sobre este tema ha sido creciente (como indican Fan y Cheng 2016). Este problema con varios conjuntos de trabajos (conocido como *multi-agent scheduling problem*) es cada vez más estudiado en la literatura, debido a la semejanza con entornos productivos reales. A diferencia de los problemas clásicos, donde todos los trabajos pertenecen a un único conjunto y contribuyen al mismo objetivo, en los problemas de varios conjuntos existen diferentes clases de trabajos que comparten recursos, teniendo cada uno un objetivo a optimizar asociado.

Ejemplos de situaciones con varios conjuntos de trabajos son (Perez-Gonzalez y Framinan, 2014): la

secuenciación de varias cadenas de suministro, la implantación de secuencias de transmisión en problemas de telecomunicaciones, la actualización de un plan de secuenciación previamente existente para solucionar posibles imprevistos (*rescheduling*) o la consideración simultánea de la producción y el mantenimiento preventivo de la maquinaria.

En este trabajo se analiza la situación en la que existen dos conjuntos de trabajo diferentes, cuyos objetivos son el de minimizar el tiempo total de finalización del primer conjunto sujeto a que no se permiten trabajos retrasados, con respecto a una fecha de entrega, en el segundo conjunto.

1.1 Objetivos del Trabajo

El objetivo principal de este trabajo es el desarrollo de un método aproximado, del tipo metaheurístico, que proporcione buenas soluciones al problema presentado en esta Introducción en un tiempo de cómputo reducido.

También se plantea como objetivo la caracterización de un modelo de programación lineal que resuelva de manera exacta el problema, comprobando hasta qué número de trabajos se puede resolver, sin altos tiempos computacionales. Además, estos resultados pueden servir para la construcción y calibración del método aproximado, calculando desviaciones porcentuales relativas.

Estos objetivos expuestos se pueden desglosar en las siguientes fases:

1. Adaptación de un modelo matemático existente en la literatura al problema propuesto.
2. Programar el modelo para su posterior resolución.
3. Resolver el modelo para un número pequeño de instancias.
4. Validar el modelo.
5. Resolver de nuevo el modelo para una batería de instancias.
6. Evaluación de los resultados.
7. Desarrollo de una técnica metaheurística para la obtención de soluciones aproximadas.
8. Calibración y experimentación del algoritmo.
9. Comparación de resultados.
10. Propuesta de modificaciones.

1.2 Sumario

Este Trabajo Fin de Máster está compuesto por 8 capítulos y diferentes anexos. A continuación, se resumen los diferentes temas tratados en cada uno de ellos:

Este primer capítulo (Introducción) introduce el propósito del trabajo y el problema objeto de estudio, así como la metodología empleada para su solución. El resto de los capítulos se estructuran siguiendo la línea de los objetivos especificados en el apartado 1.1.

El capítulo 2 (Descripción Del Problema de Programación de la Producción) define y explica en detalle la notación y los conceptos básicos necesarios para entender el contexto de estudio, así como la taxonomía de los problemas.

El capítulo 3 (Estado del Arte) presenta una recopilación de los estudios e investigaciones existentes en la literatura que tienen similitud con el problema de estudio y que, por tanto, pueden servir de referencia para el desarrollo de este trabajo.

El capítulo 4 (Modelo Matemático del Problema) describe la metodología empleada para la resolución exacta del problema, presentando un modelo de programación lineal original y su adaptación para cumplir con los objetivos propuestos.

El capítulo 5 (Programación y Análisis de Resultados Modelo Exacto) presenta la implementación del modelo para su resolución y el análisis de los resultados obtenidos, analizando los tiempos de cómputo, así como los casos en los que no es posible encontrar solución exacta.

El capítulo 6 (Metaheurística) presenta la técnica metaheurística elegida para resolver el problema, algoritmo basado en el *Simulated Annealing*, con distintas variantes y propuestas a comparar mediante diferentes experimentos.

El capítulo 7 (Experimentos y Análisis de Resultados Metaheurística) recopila los resultados obtenidos en los diferentes experimentos llevados a cabo, para obtener la propuesta que ofrezca mejores resultados.

El capítulo 8 (Conclusiones) hace un resumen de todos los capítulos anteriores, comentando los resultados más importantes obtenidos en el desarrollo del trabajo.

Por último, se incluyen unos anexos en los que se pueden encontrar los códigos desarrollados en lenguaje MATLAB, así como resultados en bruto, sin procesar. El Anexo A. Modelo Matemático recoge la implementación del modelo de programación lineal en Microsoft Excel, GAMS y MATLAB, así como la batería de problemas para resolver con Gurobi Optimizer de manera exacta, presentando todos los resultados obtenidos. El Anexo B. Código Simulated Annealing detalla el código en MATLAB de la metaheurística empleada para resolver los problemas de manera aproximada, incluyendo todas las propuestas estudiadas.

2 DESCRIPCIÓN DEL PROBLEMA DE PROGRAMACIÓN DE LA PRODUCCIÓN

La programación de la producción es un proceso que ha cobrado mucha importancia en el sector industrial, debido a su influencia en la productividad de la actividad desarrollada, por lo que es un tema de gran relevancia en el área de la investigación operativa.

El objetivo de la programación de la producción se puede entender como la asignación óptima, según un determinado criterio, de los recursos existentes a lo largo de un intervalo temporal para realizar un conjunto de tareas, creando un programa válido. En este capítulo se describen los conceptos básicos, la notación y taxonomía de los problemas.

2.1 Conceptos básicos

En este apartado se explican los conceptos básicos, necesarios para entender los problemas de programación de la producción. En primer lugar, se indican algunos términos y definiciones generales (Framinan, Leinsten y García, 2014):

- Máquina: recurso productivo con capacidad de realizar operaciones de trasformación o transporte de material (abstracción que puede representar distintos tipos de recursos, como un operario, un horno, una carretilla, etc.).

- Trabajo: producto que es objeto de alguna operación de transformación, o de transporte, en alguna de las máquinas del entorno de trabajo.

- Tiempo de proceso: duración temporal que requiere la operación de transformación de un trabajo en una determinada máquina.

La solución a un problema de programación de la producción se conoce como programa (*schedule*), esto es, la asignación en la escala temporal de una serie de trabajos a las máquinas de las que se dispone. Por tanto, el resultado indica qué máquinas procesan cada trabajo y el orden, la secuenciación, con los tiempos de inicio y fin de cada uno de ellos. La finalidad de la programación es encontrar un programa factible, es decir, un programa que cumpla con todas las restricciones del

proceso productivo y minimice uno o varios objetivos, siendo este programa la opción con mejor resultado respecto a dichos criterios establecidos.

En los problemas de programación de la producción se considera que el número de trabajos y de máquinas es finito. Típicamente, los programas se suelen representar en diagramas de Gantt, que son una popular forma de representación gráfica y temporal de los mismos. Estos diagramas se pueden expresar orientados a la máquina, como se enseña en la Figura 2-1, u orientados al trabajo, Figura 2-2, aunque en este documento se hace uso de los del primer tipo.

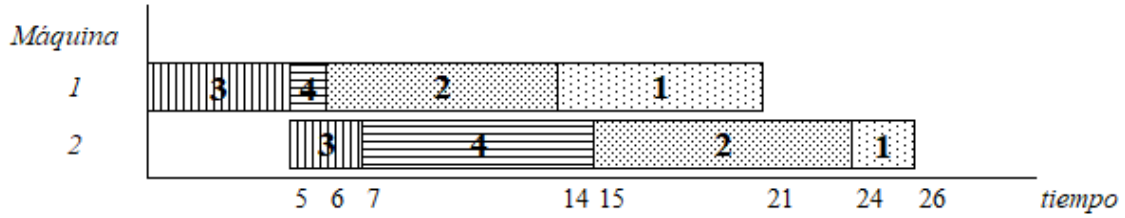


Figura 2-1. Diagrama de Gantt orientado a las máquinas[1].

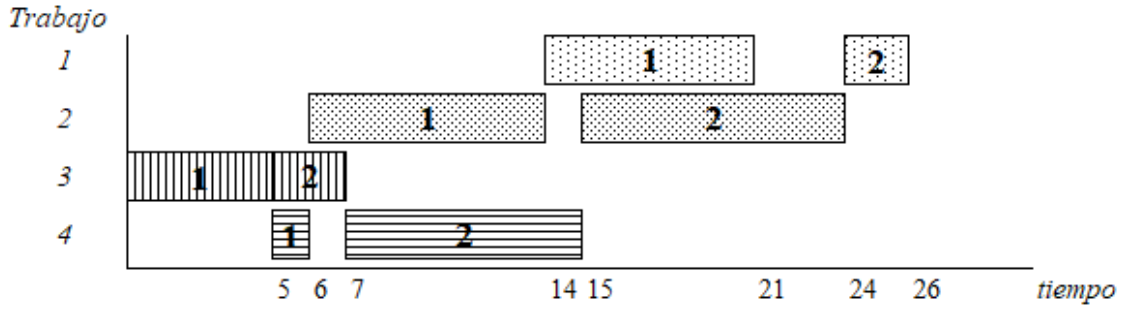


Figura 2-2. Diagrama de Gantt orientado a los trabajos[1].

Observando los diagramas de Gantt anteriormente expuestos en las figuras Figura 2-1 y Figura 2-2 se pueden extraer dos nuevos conceptos importantes:

- Ruta: secuencia predeterminada que indica el orden en que cada trabajo debe ser procesado por cada máquina.

[1] Fuente: Elaboración propia, adaptado de Framinan et al. (2014).

- Secuencia: orden en que cada una de las máquinas recibe los trabajos.

La variable a minimizar, en la mayoría de estos problemas, está relacionada con los tiempos de finalización de los trabajos. Sin embargo, en otros casos se desea optimizar más de un objetivo: son los llamados problemas multicriterio.

Por otra parte, existen problemas en los que se consideran dos o más conjuntos de trabajos, cada uno con sus propios parámetros identificativos y distinta función objetivo (aunque el objetivo de cada conjunto pueda coincidir). Este tipo de problemas son, en general, más complejos de resolver que los de un solo conjunto.

2.2 Notación

En primer lugar, es necesario establecer una notación común para poder describir cualquier problema de secuenciación. La notación usada en este documento se detalla a continuación:

- n: número de trabajos del conjunto.
- m: número de máquinas.
- j: subíndice que identifica a los trabajos del conjunto, $j = 1, \dots, n$.
- i: subíndice que identifica a las máquinas del sistema, $i = 1, \dots, m$.

Si un trabajo requiere una serie de pasos de procesamiento u operaciones, el par (i, j) se refiere a la operación del trabajo j en la máquina i (Pinedo, 2008). Algunos de los parámetros más usuales que pueden definir a los trabajos de un conjunto son los siguientes:

- p_{ij}: tiempo de proceso del trabajo j en la máquina i. Si se omite el subíndice i se supone que se procesa en una sola máquina, o cuando los tiempos de proceso no dependen de las máquinas
- r_j: fecha de llegada del trabajo j, estando disponible para ser procesado. Si $r_j = 0$, el trabajo está disponible desde el tiempo inicial.
- d_j: fecha de entrega del trabajo j, compromiso acordado con el cliente. En el caso de no cumplirse se suele incurrir en una penalización de la función objetivo.

Los problemas de programación se pueden abstraer y simplificar en modelos, los cuales quedan

diferenciados según una notación establecida por Graham, Lawler, Lenstra y Kan (1979). Utilizando la notación de Graham et al. (1979), es posible caracterizar cualquier problema de secuenciación mediante el formato $\alpha|\beta|\gamma$. El campo α describe el sistema de máquinas, y contiene un único parámetro. El campo β indica las características y restricciones del sistema, pudiendo contener una entrada, varias o ninguna. Por último, el campo γ es el que describe la función objetivo a minimizar y, normalmente, contiene una única entrada. A continuación, se explican diferentes casos:

- Entorno (α). Indica el número de máquinas y su disposición en el entorno de trabajo. La disposición se expresa con una letra y el número de máquinas con un subíndice. Los distintos entornos más habituales son los siguientes:
  - 1: *Single machine*. Caso más simple, una sola máquina, en la que cada trabajo debe ser procesado exactamente una vez.
  - P: *Identical parallel machines*. Máquinas idénticas en paralelo. El tiempo de proceso de cada trabajo en cada una de las máquinas no depende de la máquina donde se procese.
  - Q: *Uniform parallel machines*. Máquinas paralelas uniformes. Las máquinas están en paralelo, pero con distintos tiempos de proceso, debido a un nuevo parámetro, la velocidad, siendo una proporción constante para todos los trabajos.
  - R: *Unrelated parallel machines*. Máquinas paralelas no relacionadas. Las máquinas se encuentran en paralelo y tienen diversos tiempos de proceso (las máquinas no son iguales), no habiendo relación de velocidad.
  - F: *Flowshop*. Taller de flujo regular, con máquinas organizadas en serie y con diferentes propósitos, por las cuales pasan los trabajos en el mismo orden especificado. Existe un caso particular denominado taller de flujo regular de permutación (*permutation flowshop*) en el que se tiene la misma secuencia para todas las máquinas.
  - J: *Job shop*. Taller similar a *flowshop*, sin embargo, los trabajos no siguen la misma ruta.

– O: Open shop. Taller abierto. Distribución más general y compleja, con máquinas en serie, no existiendo ruta de procesamiento fija para cada trabajo y siendo posible cualquier combinación.

– H: entorno híbrido. Combinación de las anteriores.

En las Figura 2-3, Figura 2-4 y Figura 2-5, se muestran los esquemas básicos de los entornos de una sola máquina, máquinas en paralelo y máquinas en serie (flujo regular) respectivamente.

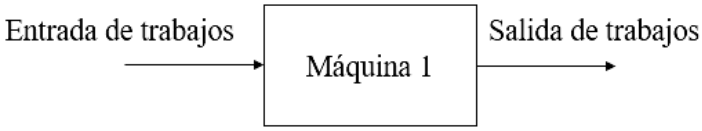


Figura 2-3. Esquema de entorno de una máquina[1].

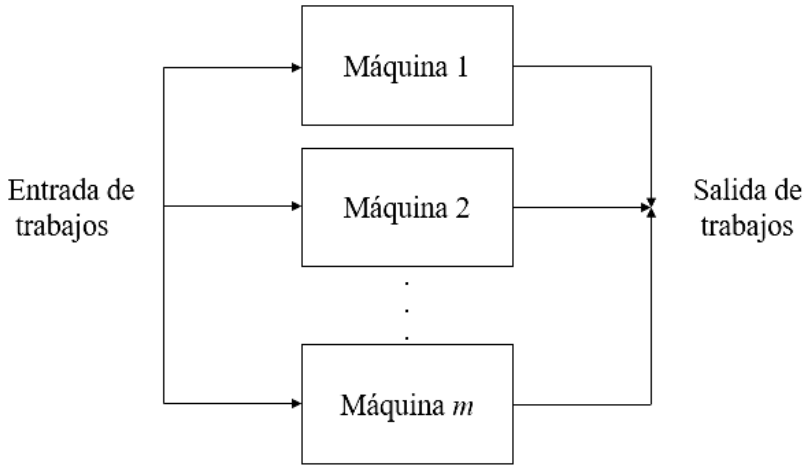


Figura 2-4. Esquema de entorno de máquinas en paralelo[1].

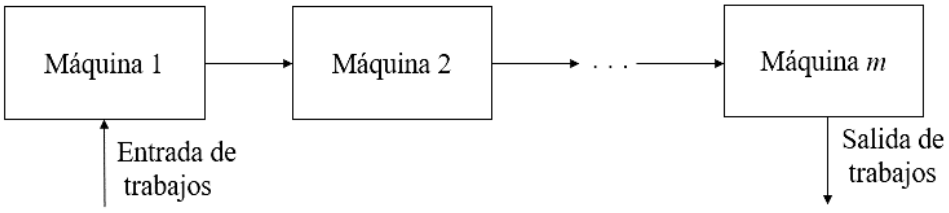


Figura 2-5. Esquema de entorno de taller de flujo regular[1].

- Restricciones (β). Esta notación se usa para explicar las características del proceso. Algunas de las restricciones más comunes son:

  – Fechas de llegada y de entrega: estas fechas marcan el instante a partir del cual el trabajo j está disponible y puede empezar a ser procesado ($\beta = r_j$) y el instante en el que debe estar realizado ($\beta = d_j$).

  – Disponibilidad de las máquinas: las máquinas no suelen estar disponibles siempre, ya que normalmente hay paradas programadas para revisión y mantenimiento preventivo, o para su calibración. Por tanto, existen periodos de indisponibilidad.

  – Precedencia de trabajos ($\beta = prec$): algunos trabajos no se pueden ejecutar sin haber realizado otros previamente. Si no aparece representado el campo no existen relaciones de precedencia.

  – Permutación ($\beta = prmu$): el orden en el que se procesan los trabajos es el mismo en cada máquina (*permutation flowshop*).

  – Los tiempos de cambio o *set up times* (s_{ij}): son tiempos de preparación que necesita la máquina para el siguiente trabajo que se va a llevar a cabo. Si no aparecen representados significa que todos los tiempos de preparación son cero.

  – Interrupciones ($\beta = pmtn$): indica si el trabajo que es interrumpido se pierde completamente y se debe empezar de nuevo al reiniciarse, o si se pierde solo parte del trabajo, y en su caso, de reiniciarse de nuevo, se continua por donde se había dejado en la interrupción.

  – Máquina no ociosa ($\beta = no\text{-}idle$): no están permitidos tiempos ociosos de las máquinas entre los diferentes trabajos, es decir, la máquina no puede parar una vez que empieza a procesar el primer trabajo de su programa.

  – *Preemptions* ($\beta = prmp$): indica que se puede interrumpir el proceso de un trabajo en una máquina para empezar con otro trabajo diferente. Cuando no está incluido no está permitido.

- *Breakdowns* ($\beta = brkdwn$): implica que las máquinas no están continuamente disponibles para procesar trabajos.

Existen algunas suposiciones generales que se consideran sin necesidad de ser indicadas en las restricciones:

- Los trabajos, disponibles al principio del horizonte de programación, se consideran no interrumpibles.
- Una máquina no puede procesar más de un trabajo simultáneamente, al igual que un trabajo no puede ser tratado por más de una máquina en un mismo instante.
- El búfer entre máquinas se supone infinito y el tiempo de transporte despreciable.

- Criterio (γ). Esta notación contiene la información sobre la función objetivo del problema (a minimizar). Hay tres tipos de objetivos comunes en secuenciación y programación de la producción: uso eficiente de los recursos, rápida respuesta a las demandas y estrecha conformidad para cumplir con los plazos (Baker y Trietsch, 2018). Algunos objetivos clásicos son:
  - *Makespan* (C_{max}): es la cantidad total de tiempo requerido para procesar completamente todos los trabajos. Equivalente al tiempo de finalización del último trabajo completado. Se define como $max\ (C_1, \dots, C_n)$.
  - Retraso máximo (L_{max}): mide el trabajo que más se retrasa con respecto a su fecha de entrega, la peor desviación. Definido como $L_{max} = max\ (L_1, \dots, L_n)$.
  - Tiempo total de finalización ($\sum C_j$): es el tiempo total de realización de todos los trabajos. A menudo, la suma de los tiempos de finalización está referida en la literatura como el tiempo del flujo (*flow time*). Se calcula como $\sum_{j=1}^{n} C_j$.
  - Tardanza total ($\sum T_j$): es la suma de las tardanzas de cada trabajo respecto a su fecha de entrega.
  - Número de trabajos retrasados ($\sum U_j$): es la suma de todos los trabajos que han sido retrasados respecto a su fecha de entrega.

En el caso de que los trabajos tuvieran distinta importancia, habría que ponderar las

expresiones anteriores con sus respectivos pesos, w_j.

Los objetivos expuestos anteriormente se calculan con las siguientes medidas:

- C_j: tiempo de finalización del trabajo j (*completion time*). Instante en el que un trabajo finaliza su procesamiento en el entorno.
- L_j: retraso del trabajo j (*lateness*). Mide lo que se retrasa ($L_j \geq 0$) o adelanta ($L_j \leq 0$) un trabajo. $L_j = C_j - d_j$.
- T_j: tardanza del trabajo j (*tardiness*). Cuantifica lo que se retrasa un trabajo respecto a la fecha de entrega. $T_j = max\{0, L_j\}$.
- U_j: trabajo retrasado (*tardy job*). $U_j = 1$ si $T_j \geq 0$, es decir $C_j > d_j$; $U_j = 0$, en caso contrario.

Existen muchos más objetivos, así como diferentes medidas, que se pueden encontrar en la literatura existente sobre esta área de investigación.

Como se ha comentado en capítulo 1 de Introducción, este trabajo está enfocado en los problemas con más de un conjunto de trabajos (también conocidos como *interfering scheduling problems*, *multi-agent scheduling problems*, *competing agents* o *heterogenous-criteria scheduling*), los cuales compiten por el uso de las mismas máquinas, teniendo cada uno su propio objetivo (Perez-Gonzalez y Framinan, 2014).

Para la notación de los problemas de secuenciación con más de un conjunto de trabajo se hace uso del estudio desarrollado por Perez-Gonzalez y Framinan (2014), en el que declaran un marco común para tener una visión unificada del mismo, de acuerdo con los diferentes trabajos existentes en la literatura.

Se considera $k = 2, \dots, K$ conjuntos de n^k trabajos cada uno, denotados como $\zeta^1, \dots, \zeta^K$. Se define el número total de trabajos como $n = \sum_{k=1}^{K} n^k$ y $\zeta = \zeta^1 \cup \dots \cup \zeta^K$. Cada conjunto ζ^k tiene asignado un criterio f^k. Cada trabajo $j \in \zeta^k$ tiene que ser procesado en la máquina i con un tiempo de proceso p_{ij}^k, con $i = 1, \dots, m$.

Una secuencia completa $\sigma = [\sigma_1, \dots, \sigma_n]$ es una permutación de todos los trabajos en ζ, donde cada $\sigma_j, j = 1, \dots, n$, está en algún ζ^k. Dada una secuencia σ, el tiempo de finalización del trabajo σ_j en la

máquina *i* se denota como $C_{ij}(\sigma)$, siendo $C_j(\sigma)$ el tiempo de finalización del trabajo σ_j en la última máquina en la que el trabajo debe ser procesado.

Los enfoques más comunes a la hora de minimizar los problemas de secuenciación son:

- Método de la combinación lineal convexa: se denota $F_l(f^1, \dots, f^k)$ si el objetivo es minimizar una combinación lineal convexa de criterio K, $\sum_{i=1}^{K} \lambda_i f^i$, donde $\sum_{i=1}^{K} \lambda_i = 1$, dependiendo de la importancia de cada objetivo. Por tanto, busca optimizar la suma ponderada de los distintos objetivos de los diferentes conjuntos.

- Método de la restricción-épsilon: se denota $\varepsilon(f^1/f^2, \dots, f^K)$ si el objetivo es minimizar f^1 sujeto a $f^k \leq \varepsilon^k$, $\varepsilon^k \geq 0$ para $k = 2, \dots, K$. Es decir, trata de optimizar el objetivo de uno de los conjuntos sujeto a que los objetivos de los otros conjuntos no superen un cierto límite.

- Método de Pareto: se denota $\#(f^1, \dots, f^K)$ si el objetivo es enumerar todos los Pareto-óptima. En este método, una solución se denomina Pareto-óptima si no es posible disminuir el valor de un objetivo sin aumentar el valor de otro.

2.3 Caracterización del problema de estudio

El problema de estudio en este documento se basa en un entorno *flowshop* con dos máquinas y dos conjuntos de trabajos, A y B, con los objetivos de minimizar el tiempo total de finalización del conjunto de trabajos A con retraso cero para todos los trabajos en el conjunto B.

Utilizando la notación explicada en el apartado 2.1, el problema se podría introducir como: dos conjuntos disjuntos ζ^A y ζ^B ($k = 2, \zeta^A \cap \zeta^B = \emptyset$), de n^A y n^B trabajos respectivamente, tal que $\zeta = \zeta^A \cup \zeta^B$ y $n = n^A + n^B$, y definirlo, usando la notación de Graham et al. (1979) $\alpha|\beta|\gamma$ como:

$$PF2|\sum_{j\epsilon B} T_j^B = 0|\sum_{j\epsilon A} C_{2j}^A$$

A continuación, se hacen algunas aclaraciones con respecto a la notación expuesta en la triple terna de Graham:

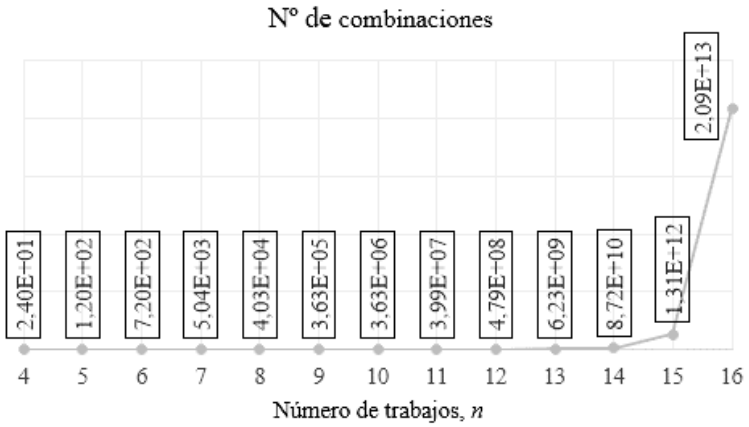
- Entorno: $\alpha = PF2$. Dos máquinas formando un entorno de trabajo de taller de flujo regular de permutación. Se incorpora la letra P para hacer referencia a la permutación (*permutation flowshop*), siguiendo lo descrito por Perez-Gonzalez y Framinan (2014) y

en aras de crear una taxonomía más clara de los problemas con el campo β vacío (en la notación estándar, la condición de permutación se indica en el campo β mediante *prmu*).

- Restricciones: $\beta = \sum_{j\epsilon B} T_j^B = 0$. El problema de dos conjuntos de trabajos planteado se resuelve mediante el enfoque restricción-épsilon, $\varepsilon(f^A/f^B)$, en el que ε toma el valor 0, ya que no puede haber ningún trabajo retrasado en el conjunto B, por lo que se expresa como una restricción en el campo β. Si bien se ha usado $\sum_{j\epsilon B} T_j^B = 0$, también se podría haber expresado como $\sum_{j\epsilon B} U_j^B = 0$ (Lee, Chen, Chen y Wu, 2011) pero, no obstante, la interpretación es la misma. Otra forma de definir el problema, atendiendo a la notación de Graham y al enfoque restricción-épsilon, es $PF2||0(\sum_{j\epsilon A} C_{2j}^A / \sum_{j\epsilon B} T_j^B)$, donde ε se sustituye por su valor 0, dejando el campo β vacío.

- Criterio: $\gamma = \sum_{j\epsilon A} C_{2j}^A$. El objetivo del primer conjunto de trabajos, A, es el de minimizar el tiempo total de finalización de todos los trabajos. Se enfatiza con el subíndice 2 el hecho de que hay dos máquinas y que, por tanto, la segunda máquina es la salida y fin del entorno de trabajo.

La solución óptima del problema expuesto es una de las $(n^A + n^B)! = n!$ combinaciones posibles. Si no se impusiera la restricción de permutación, el problema sería aún más complejo, teniendo un total de $(n!)^m$ combinaciones posibles. En cualquier caso, el número de soluciones posibles crece de manera factorial, por lo que la dificultad del problema crece de manera exponencial al aumentar el número de trabajos n. En la Gráfica 2-1 se puede observar que el número de combinaciones al aumentar n es cada vez mayor, creciendo además con mucha rapidez.



Gráfica 2-1. Complejidad del problema según *n*.

A continuación, se presenta un ejemplo sencillo con el objetivo de entender los diferentes casos de programas que se pueden tener. El ejemplo consta de cuatro trabajos, correspondiendo a cada conjunto de trabajo dos de ellos. En la Tabla 2-1 se detallan los tiempos de proceso y fechas de entrega de cada trabajo en unidades de tiempo.

Ejemplo. Flowshop de 2 máquinas y 2 conjuntos de trabajos

Tabla 2-1. Datos ejemplo numérico.

	A		B	
j	1	2	3	4
p_{1j}^k	7	8	5	1
p_{2j}^k	2	9	2	8
d_j	17	25	10	15

La primera secuencia que se plantea es la secuencia óptima, $\sigma = [3, 4, 1, 2]$. A continuación, en la Figura 2-6, se muestra el correspondiente diagrama de Gantt.

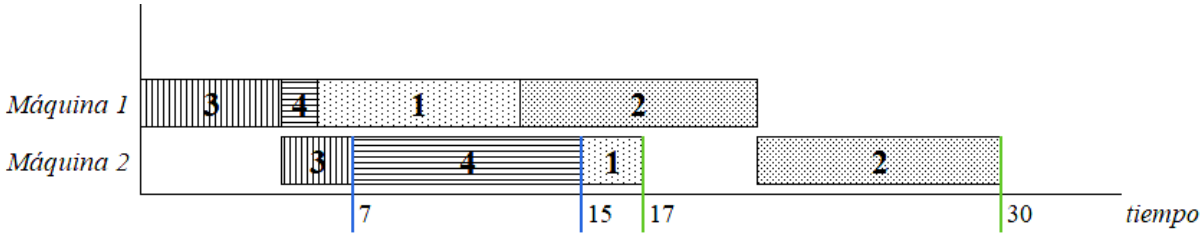


Figura 2-6. Solución óptima del ejemplo numérico.

El objetivo del conjunto A se calcula como

$$\sum_{j=1}^{2} C_j = 17 + 30 = 47.$$

Además, se puede comprobar que el objetivo del conjunto B también se cumple ya que

$$C_3 = 7 < 10 = d_3 \text{ y } C_4 = 15 = d_4,$$

por lo que no hay ningún trabajo retrasado.

La segunda secuencia propuesta es $\sigma = [3, 4, 2, 1]$, cuyo diagrama se Gantt se muestra en la Figura 2-7.

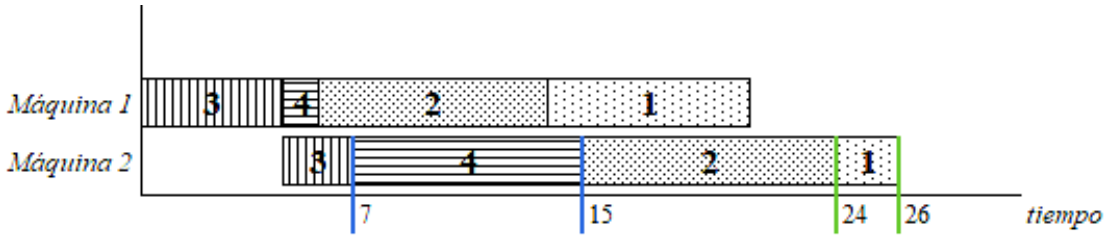


Figura 2-7. Solución factible del ejemplo numérico.

De nuevo, el objetivo, o restricción, impuesta para el conjunto B se cumple (mismos tiempos de finalización, véase Figura 2-6), sin embargo, el objetivo de A es

$$\sum_{j=1}^{2} C_j = 26 + 24 = 50,$$

valor mayor al calculado en la secuecia anterior, por lo que estamos ante una secuencia factible pero no óptima.

Por último, se plantea la secuencia $\sigma = [1, 3, 4, 2]$, con diagrama de Gantt correspondiente al de la Figura 2-8.

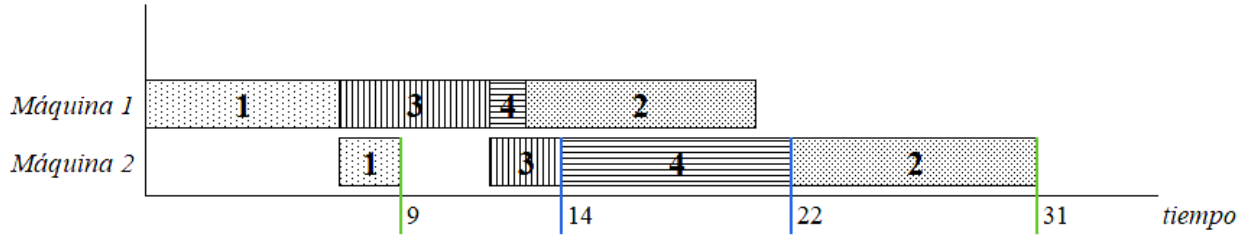


Figura 2-8. Solución no factible del ejemplo numérico.

Como se puede observar en el diagrama de Gantt de la Figura 2-8, aunque el objetivo de A es mejor,

pues

$$\sum_{j=1}^{2} C_j = 9 + 31 = 40,$$

el objetivo de no tener trabajos retrasados en el conjunto B no se cumple, ya que para los dos trabajos del conjunto B se tiene que

$$C_3 = 14 > 10 = d_3 \text{ y } C_4 = 22 > 15 = d_4,$$

por lo que se trata de una secuencia no factible.

2.4 Complejidad computacional

La teoría de complejidad proporciona un marco para clasificar los problemas computacionales de acuerdo con su estructura y dificultad. Los modelos se pueden clasificar como P (modelos con algoritmos polinomiales, "fáciles") y NP (modelos con algoritmos no polinomiales, "difíciles").

La mayoría de los problemas que se plantean de *flowshop* son NP. El problema tratado en este estudio (*flowshop* de permutación con objetivos de tiempo total de finalización y retraso nulo para cada conjunto de trabajos respectivamente) es un problema de optimización catalogado como NP, puesto que el problema base $1||\epsilon(\sum C_j^A \,/\, \sum T_j^B)$ también lo es (Perez-Gonzalez y Framinan, 2014).

Un problema NP-*Hard* se presenta cuando un algoritmo que intenta solucionarlo aumenta su tiempo de ejecución, en el peor de los casos de forma exponencial al tamaño del problema. Por tanto, en los problemas NP-*Hard* de instancias de tamaños elevados, y dependiendo del problema, el objetivo no suele ser buscar la solución óptima, sino buscar soluciones buenas que sean próximas a la óptima, debido a la gran complejidad.

En el caso de los problemas "difíciles", algunas estrategias posibles que se pueden llevar a cabo son:

- Encontrar un algoritmo exacto, con tiempo de ejecución exponencial, para problemas de tamaños moderados.
- Intentar reducir la complejidad del modelo.
- Intentar reducir el potencial del algoritmo (soluciones aproximadas).
- No usar algoritmos, usar reglas de despacho (*dispatching rules*).

2.5 Métodos de programación de la producción

Existen diversos métodos de programación a la hora de resolver un problema de programación de la producción. Estos métodos de programación proporcionan una solución factible a un problema, teniendo, la solución obtenida, que proporcionar unos buenos resultados acorde con un objetivo establecido. Dichos métodos que proporcionan una solución se denominan algoritmos, pudiendo ser de dos tipos (Pinedo, 2008):

- Métodos exactos: proporcionan la solución óptima a un problema concreto, existiendo la garantía de que ningún otro programa factible funciona mejor que el obtenido, con respecto al objetivo buscado. Estas técnicas son rápidas para problemas polinomiales (P) y para algunos problemas no polinomiales (NP) que no sean excesivamente complejos. Los métodos exactos para problemas NP-*Hard* requieren con frecuencia un tiempo de búsqueda para obtener la solución extremadamente grande cuando el número de trabajos se incrementa. Dentro de este tipo se encuentran:

  – Algoritmos constructivos exactos: son algoritmos polinomiales aplicados para resolver problemas polinomiales, aunque existen muy pocos problemas polinomiales en el ámbito de la programación de la producción. Se caracterizan por dar el óptimo en un tiempo reducido independientemente del tamaño de la instancia. Un ejemplo es el algoritmo de Jonhson (Johnson, 1954), que resuelve de forma óptima el problema $F_2||C_{max}$.

  – Algoritmos enumerativos: son algortimos no polinomiales que garantizan obtener el óptimo de cualquier problema, ya que exploran la totalidad del espacio de soluciones. Sin embargo, para instancias grandes los tiempos computacionales son muy altos, por lo que su aplicación en problemas polinomiales no es razonable. Entre ellos se encuentran:

    - Programación matemática: la formulación se realiza mediante la programación lineal entera mixta (MILP) y se resuelve mediante un *solver*.

    - *Branch and Bound*: se utiliza para resolver problemas de optimización combinatoria. Para ello, es necesario definir los nodos, definir la ramificación, calcular la cota y establecer una estrategia de ramificación.

- Programación dinámica: se utiliza para optimizar problemas complejos que pueden ser discretizados y secuenciados.

- Métodos aproximados: proporcionan programas factibles, sin garantizar que sean los óptimos. No es posible predecir el comportamiento de los algoritmos, cuya eficiencia es evaluada mediante experimentos. Se suelen usar para resolver los problemas NP-*Hard* en tiempos pequeños. Entre este tipo de algoritmos se encuentran:

  - Heurísticas constructivas: este tipo de métodos comienzan sin una secuencia inicial y poco a poco construyen una, mediante la adición de un trabajo cada vez. Un ejemplo de heurística constructiva son las reglas de despacho (*dispatching rules*), como por ejemplo FIFO (*First In First Out*), SPT (*Shortest Processing Time first*) o LPT (*Longest Processing Time first*)

  - Metaheurísticas: se basan en procedimientos de búsqueda para mejorar soluciones previamente encontradas. Tratan de optimizar localmente alrededor de una solución. Algunas de las metaheurísticas más usadas son los Algoritmos Genéticos, búsqueda tabú, *Simulated Annealing* o *Iterated Greedy*, entre otros.

3 ESTADO DEL ARTE

En este capítulo se recoge, una vez caracterizado el problema objeto de estudio, una revisión de la literatura existente. El problema se basa en un entorno de trabajo de flujo regular con dos máquinas y dos conjuntos de trabajo. Se trata de minimizar el tiempo total de finalización de los trabajos del primer conjunto, A, sujeto a no tener trabajos retrasados en el segundo conjunto, B, caracterizando el problema según la notación de Graham como: $PF2|\sum_{j\epsilon B} T_j^B = 0|\sum_{j\epsilon A} C_{2j}^A$.

En primer lugar, se revisan artículos que investigan problemas de flujo regular de permutación con varios conjuntos de trabajos y, en segundo lugar, artículos que hacen uso de la metaheurística propuesta para resolver el problema, *Simulated Annealing.*

3.1 Problemas *flowshop* con varios conjuntos de trabajos

El área de investigación sobre programación de la producción en problemas con varios conjuntos de trabajos empezó a estudiarse en profundidad desde Smith y Baker (2003) y Agnetis et al. (2004). A partir de entonces, numerosos artículos se pueden encontrar debido a su especial interés por la aplicación en problemas reales.

Smith y Baker (2003) plantean un ejemplo con dos departamentos diferentes, el departamento de fabricación y el departamento de investigación y desarrollo. Cada departamento tiene su propio conjunto de trabajos, pero comparten el mismo recurso, una máquina. El departamento de fabricación tiene como objetivo finalizar los trabajos antes de sus fechas de entrega, mientras que el objetivo del departamento de investigación y desarrollo es dar respuestas rápidas a los problemas. En este artículo, los autores estudian el problema minimizando la combinación lineal de los objetivos de ambos departamentos y sus conjuntos de trabajos. Por otro lado, Agnetis et al. (2004) hacen uso de los otros dos métodos explicados en el apartado 2.2, restricción-épsilon y optimización de Pareto. Este estudio se aplica sobre diferentes configuraciones de la máquina.

Un documento que estudia un marco común y taxonomía de los problemas de programación de la producción con diferentes conjuntos que compiten por los mismo recursos es Perez-Gonzalez y Framinan (2014). En este documento se presenta la notación más extendida entre los diferentes

estudios existentes en la literatura, y que se usa en este documento. Además, clasifica los problemas según su complejidad, haciendo una revisión extendida de los diferentes tipos de problemas basada en los métodos de resolución empleados.

En este apartado, se revisan algunos artículos relacionados con el problema de estudio, particularmente problemas con dos conjuntos disjuntos de trabajos en los que, principalmente, no se imponen condiciones sobre las máquinas (campo β vacío).

Muchos de los estudios con varios conjuntos de trabajo se centran en el entorno de trabajo de una sola máquina, pero en este apartado se reduce la referencia únicamente a aquellos cuyo entorno de trabajo consiste en el taller de flujo regular, haciendo énfasis en aquellos problemas con dos máquinas.

Se puede encontrar en la literatura un par de estudios que desarrollan soluciones al mismo problema propuesto en este documento (véase 2.3 Caracterización del problema de estudio). Lee, Chen y Wu (2010) proponen un algoritmo *Branch&Bound* y diferentes algoritmos heurísticos *Simulated Annealing* para buscar soluciones óptimas y cercanas a las óptimas con los objetivos de minimizar el tiempo total de finalización de un conjunto de trabajos, sin tener trabajos retrasados en el otro conjunto, $PF2|\sum_{j\epsilon B} U_j^B = 0\,|\sum_{j\epsilon A} C_{2j}^A$. Posteriormente, Lee et al. (2011) proponen una nueva versión del estudio, muy similar, añadiendo otras heurísticas para el *Simulated Annealing*. En ambos estudios se muestran los resultados finales y las comparaciones realizadas para sacar conclusiones en cuanto a dominancia de ciertas heurísticas.

Otro estudio similar es el desarrollado por Shiau, Tsai, Lee y Cheng (2015). Los objetivos son minimizar el tiempo total de finalización de un conjunto de trabajos, pero en este caso y a diferencia de Lee et al. (2011), sujeto a que la máxima tardanza del otro conjunto no supere un cierto límite superior, $PF2||\varepsilon(\sum_{j\epsilon A} C_{2j}^A / \sum_{j\epsilon B} T_j^B)$. Para la resolución del problema proponen también un algoritmo *Branch&Bound* y otra metaheurística basada en varias versiones de algoritmo genético para obtener soluciones cercanas a la óptima. Un problema similar se consigue modificando el tiempo total de finalización por el *makespan*. Así lo propone Lei (2015), cuyo artículo se basa en estudiar la factibilidad del problema $PF2||\varepsilon(C_{max}^A / \sum_{j\epsilon B} T_j^B)$. Para su resolución propone un algortimo de búsqueda con vecindad variable (VNS), incorporando un método de aprendizaje para producir nuevas soluciones.

Invirtiendo los objetivos expuestos en Lei (2015), se encuentra el estudio propuesto por Ahmadi-Darani, Moslehi y Reisi-Nafchi (2018). Estudian los objetivos de minimizar la tardanza total del

primer conjunto, sujeto a la condición de que el *makespan* del segundo conjunto debe ser menor a cierto límite superior, $PF2||\varepsilon(\sum_{j\epsilon A} T_j^A / C_{max}^B)$. Para su resolución, proponen un modelo matemático de programación lineal con ciertos teoremas y propiedades para alcanzar optimalidad, hasta un cierto número de trabajos. Además, proponen un algoritmo de búsqueda tabú, cuyos resultados son comparados con los óptimos, logrando gran efectividad.

En referencia a estudios que se centran en la suma ponderada de los objetivos mediante combinación lineal convexa, se encuentra Jeong, Kim y Shim (2020). Consideran el objetivo de minimizar la tardanza total de un conjunto y el tiempo total de finalización del otro conjunto, $PF2||F_l(\sum_{j\epsilon A} T_j^A, \sum_{j\epsilon A} C_{2j}^B)$. Proponen un algoritmo *Branch&Bound* desarrollando límites inferiores y propiedades de dominancia, así como diversas heurísticas para obtener soluciones iniciales.

Por otro lado, Fan y Cheng (2016) abordan el problema de minimizar la suma ponderada del *makespan* de ambos conjuntos, $PF2||F_l(C_{max}^A, C_{max}^B)$. Demuestran que se trata de un problrma NP-*Hard*, y proponen un algoritmo aproximado basado en el algoritmo de Jhonson, considerando además unos casos especiales que se pueden resolver en tiempos polinomiales. Por otro lado, consideran minimizar la suma ponderada del tiempo total de finalización de un conjunto y el *makespan* del otro conjunto, $PF2||F_l(\sum_{j\epsilon A} C_{2j}^A, C_{max}^B)$, para el que plantean un algoritmo aproximado basado en la relajación del problema de programación lineal.

Luo, Chen y Zhang (2012) consideran el problema con dos máquinas $PF2||F_l(C_{max}^A, C_{max}^B)$, a resolver mediante combinación lineal convexa, al igual que Fan y Cheng (2016). En primer lugar, demuestran que se trata de un problema NP-*Hard*, para luego proponer un algoritmo de programación dinámica basado en un esquema de aproximación en tiempo completamente polinomial (FPTAS). Consideran además la resolución del problema mediante el método de la restricción-épsilon, $PF2||\varepsilon(C_{max}^A / C_{max}^B)$, demostrando igualmente que es NP-*Hard*. Para este último, también proponen un algoritmo del tipo FPTAS, distinguiendo diferentes casos según el valor de ε y relajándolo, según convenga, en cierta cantidad.

Un estudio que aplica los tres métodos de resolución explicados (combinación lineal convexa, restricción-epsilón y Pareto) es Yin, Cheng, Wang y Wu (2017). Los objetivos que estudian son en base al método justo a tiempo (*Just in time*: JIT) para los trabajos de ambos conjuntos. Por tanto, se plantean cuatro problemas con las siguientes formas: $PF2||F_l(f^A, f^B)$, $PF2||\varepsilon(f^A/f^B)$, $PF2||\varepsilon(f^B/f^A)$ y $PF2||\#(f^A, f^B)$.

Igualmente, se puede extender el problema añadiendo condiciones a los trabajos o máquinas, es decir, el campo β no está vacío. En Perez-Gonzalez y Framinan (2010) se plantea el problema de *rescheduling*, donde hay que minimizar el *makespan* de un nuevo conjunto de trabajos, cuya restricción es que la máxima tardanza de los trabajos preexistentes debe ser cero. Para resolver el problema proponen una heurística basada en la búsqueda en vecindad variable (VNS).

Siguiendo la misma línea de ideas, se encuentran artículos que extienden el problema a entornos *flowshop* con m máquinas. Ejemplo de ello se expone en Mor y Mosheiov (2014), donde estudian tres casos, variando el objetivo para el primer conjunto: minimizar el máximo coste de todos los trabajos, el tiempo total de finalización y el número de trabajos retrasados del primer conjunto sujeto a no superar un límite superior para el coste de los trabajos del segundo conjunto. Para resolver los problemas proponen algoritmos de tiempo polinomial para cada caso, demostrando que las configuraciones de los casos estudiados se pueden resolver en tiempo polinomial.

3.2 *Simulated Annealing* en problemas de secuenciación

Muchas de las técnicas usadas para resolver los problemas de secuenciación con más de un conjunto de trabajo se basan en técnicas metaheurísticas. En este apartado se hace una revisión de la literatura existente sobre al algoritmo *Simulated Annealing* aplicado a problemas de secuenciación, ya que es el método aproximado elegido para resolver el problema de estudio.

El primer artículo sobre el algoritmo *Simulated Annealing* es desarrollado por Kirkpatrick, Gelatt y Vecchi (1983). En dicho estudio se establecen las bases del algoritmo para ser aplicado a los problemas de optimización combinatoria, y se propone un pre-procesado para obtener la función probabilística de aceptación, base fundamental de la técnica y en la que se basan muchos otros estudios. Esta función, y en consecuencia el algoritmo, surge de los cálculos de ecuaciones de estado planteados por Metropolis, Rosenbluth, Rosenbluth y Teller (1953).

Lundy y Mess (1986) estudian y prueban en su artículo la convergencia del método, obteniendo buenos resultados con una probabilidad cercana a 1. Además, muestran casos en los que la convergencia toma mucho tiempo computacional y, por tanto, no es práctica su aplicación. Una clara analogía del proceso simulado en el algoritmo con los procesos de optimización se detalla en Dowsland (1995), donde también se pone de manifiesto diferentes parámetros de control que se pueden incluir en la función probabilística.

Uno de los primeros artículos que plasman el uso del *Simulated Annealing* aplicado a problemas de

programación de la producción se presenta en Osman y Potts (1989). En este estudio se presenta un entorno *flowshop* de hasta 20 máquinas y 100 trabajos, con el objetivo de minimizar el tiempo total de finalización de todos los trabajos. Una investigación similar en un entorno *flowshop* con n trabajos y m máquinas se desarrolla en Ogbu y Smith (1990).

Dos estudios que aplican el algoritmo *Simulated Annealing* sobre el mismo problema que se estudia en este documentos son los ya mencionados Lee et al. (2010) y Lee et al. (2011). En estos estudios se aplican y comparan diferentes heurísticas. Lo más interesante es la función probabilística usada, basada en la expuesta por Ben-Arieh y Maimon (1992), donde usan el *Simulated Annealing* para la programación de la secuenciación en el ensamblaje de PCBs en dos máquinas en serie.

Karasakal y Köksalan (2000) aplican el algoritmo sobre dos problemas diferentes, basados en el entorno de una sola máquina y con multi criterio. El primer problema es el problema NP-*Hard* de minimizar el tiempo total de flujo y maximizar el adelanto en la finalización de los trabajos. El segundo problema, también NP-*Hard*, consiste en minimizar el tiempo total de flujo y el número de trabajos retrasados. Gallo y Capozzi (2019) tratan el algoritmo en un estudio con m máquinas en serie, donde en primer lugar, estudian el objetivo de minimizar el tiempo total de finalización, y en segundo lugar, estudian minimizar la suma de los tiempos individuales de finalización de cada trabajo. Yazgan y Akkaya (2015) proponen el estudio del algoritmo sobre el primer objetivo estudiado por Gallo y Capozzi (2019).

El algoritmo es aplicado también en otro tipo de entornos, como en Hoon y Pinedo (1997), donde se aplica a dos máquinas paralelas con tiempos de *setup* para minimizar la suma ponderada de la tardanza de los diferentes trabajos. En Fattahi, Saidi y Jolai (2007) también se comparan diferentes heurísticas para resolver un problema *job shop* flexible, entre las que se encuentra el *Simulated Annealing*.

Otros artículos se centran en proponer técnicas de mejora para aplicar a diferentes heurísticas en los problemas de secuenciación. Locatelli (2000) investiga y demuestra la convergencia del algoritmo hacia el óptimo global usando una función basada en la distancia entre las soluciones anzalizadas. Li, Wang y Wu (2009) proponen un método para mejorar los problemas de flujo regular de permutación que tienen como objetivo minimizar el tiempo total del flujo. Este método, *General flowtime computing* (GFC), es presentado en aras de reducir el tiempo computacional, basándose en la similitud de las soluciones generadas. Más tarde, Duan, Yang, Gao, Li y Pan (2013) proponen una mejora sobre el método expuesto por Li et al. (2009).

Hay otros estudios que utilizan este algoritmo, pero que no se aplican en problemas de

secuenciación. Sin embargo, al tratarse de un método de resolución general, utilizan técnicas genéricas que pueden ser aplicada en otros contextos. En el artículo Addou, Serghini y Mermri (2018) se hace un estudio enfocado a los probemas de optimización basados en *spanning tree*, donde proponen una técnica de reinicio para actualizar ciertos parámetros según algunos criterios. Técnicas similares de reinicio se desarrollan en Normasari, Yu, Bachtiyar y Sukoyo (2019), donde usan el algoritmo *Simulated Annealing* para un problema de ruta de vehículos.

4 MODELO MATEMÁTICO DEL PROBLEMA

En este capítulo se exponen:

- Los índices, parámetros (datos) y variables empleados para la construcción del modelo matemático de programación lineal.
- El modelo de programación lineal entera original y su adaptación para resolver el problema propuesto: $PF2|\sum_{j\epsilon B} T_j^B = 0|\sum_{j\epsilon A} C_{2j}^A$.

Una vez que se han definido cada uno de los conceptos básicos de la programación de la producción y ha quedado caracterizado el problema a estudiar, se procede a la exposición y explicación del modelo de programación lineal entera realizado para resolver el problema.

El modelo a seguir es extraído de Stafford, Tseng y Gupta (2005), donde se hace una revisión de los diferentes modelos existentes en la literatura para resolver el problema de *flowshop*. Existen dos familias de modelos: la familia Wagner, cuyos modelos incorporan el problema de asignación, y la familia Manne, cuyos problemas usan un par de restricciones dicotómicas, o sus equivalentes matemáticos, para asiganr las posiciones en la secuencia. Los modelos están planteados para un solo conjunto de trabajos y objetivo *makespan*, por lo que en cualquier caso hay que hacer una adaptación al problema de estudio en este documento.

El modelo elegido para analizar el problema de estudio es el SGST de la familia Manne. Esta familia de modelos presenta menos variables que los correspondientes a la familia Wagner. Además, la sencillez del modelo hace más fácil su adaptación, tanto al objetivo propuesto como al hecho de trabajar con varios conjuntos de trabajos.

4.1 Índices, parámetros y variables empleados

Índices:

i, para las máquinas $(1 \leq i \leq m)$.

j, k, para los trabajos $(\leq j \leq k \leq n)$.

Parámetros:

m, número de máquinas.

n, número de trabajos.

p_{ij}, tiempo de proceso del trabajo j en la máquina i.

d_j, fecha de entrega del trabajo j.

M, representa un valor lo suficientemente grande como para que se cumpla el par de restricciones dicotómicas relacionadas con la precedencia entre los trabajos. El valor de M es para cada problema la suma de los tiempos de proceso de todos los trabajos en todas las máquinas: $M = \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij}$.

Variables:

C_{ij}, tiempo de finalización del trabajo j en la máquina i.

C_{max}, tiempo de finalización del último trabajo en la última máquina.

$\delta_{jk} = 1$, si el trabajo $j$es procesado antes que k; $\delta_{jk} = 0$, en caso contrario.

4.2 Modelo original

Para la resolución óptima se ha optado por un modelo matemático de la familia Manne, el SGST, para posteriormente adaptarlo al problema descrito. El modelo SGST de Manne descrito en Stafford et al. (2005) corresponde al problema $PF_m||C_{max}$. Este modelo emplea las variables descritas en el apartado 4.1, C_{ij}, δ_{jk} y C_{max}, siendo esta última, el tiempo de finalización del último trabajo en la útima máquina o *makespan*, el objetivo original a minimizar.

El modelo SGST usa los tiempos de inicio de los trabajos para desarrollar las restricciones. Ademas, incorpora un par de restricciones dicotómicas, con las cuales se resuelve el problema de asignación de posiciones en la secuencia de procesamiento de los trabajos en las máquinas. El modelo completo se detalla a continuación:

$$min\ C_{max} \tag{4.1}$$

s.a.

$$C_{max} \geq C_{mj} \quad (1 \leq j \leq n) \tag{4.2}$$

$$C_{1j} \geq p_{1j} \quad (1 \leq j \leq n) \tag{4.3}$$

$$C_{ij} - C_{i-1,j} \geq p_{ij} \quad (2 \leq i \leq m;\ 1 \leq j \leq n) \tag{4.4}$$

$$C_{ij} - C_{ik} + M\delta_{jk} \geq p_{ij} \quad (1 \leq i \leq m;\ 1 \leq j \leq k \leq n) \tag{4.5}$$

$$C_{ij} - C_{ik} + M\delta_{jk} \leq M - p_{ik} \quad (1 \leq i \leq m;\ 1 \leq j \leq k \leq n) \tag{4.6}$$

$$C_{ij},\ C_{max} \geq 0;\ 1 \leq i \leq m,\ 1 \leq j \leq n$$

$$\delta_{jk} \in \{0, 1\};\ 1 \leq j \leq k \leq n$$

En este modelo la función objetivo (4.1) coincide con la variable C_{max}, *makespan* del conjunto de trabajos, cuyo valor debe ser igual al tiempo de finalización del último de los trabajos en la secuencia de producción, es decir, en la última máquina, restricción (4.2).

La restricción (4.3) asegura que el tiempo de finalización de los trabajos en la máquina 1 debe ser mayor o igual que el tiempo que llevan ser procesados. El conjunto de restricciones (4.4) asegura que un trabajo no pueda ser procesado en una máquina hasta que no acabe en la anterior $i-1$, garantizando igualmente que un trabajo solo puede ser procesado por una máquina en un instante. El par de restricciones dicotómicas (4.5) y (4.6) relacionan las posiciones de dos trabajos j y k ($j \neq k$), asegurando que cada trabajo j preceda o siga a k en la secuencia, pero no ambas situaciones.

El modelo original consta de un total de $n(n-1)/2$ variables binarias y $m \cdot n$ variables reales, así como de $m \cdot n^2$ restricciones.

4.3 Modelo adaptado

En primer lugar, para adaptar el modelo SGST, hay que tener en cuenta que el número de máquinas es igual a dos, por lo que $m = 2$. En segundo lugar, hay que distinguir que hay dos conjuntos de trabajos diferentes bajo estudio, A y B. Estos están formados por n^A y n^B trabajos respectivamente,

por lo que los n^A primeros trabajos hacen referencia a los trabajos del conjunto A y los n^B siguientes a los del conjunto B ($n = n^A + n^B$). De este modo, el subíndice j de un trabajo lo identifica unívocamente con uno de los dos conjuntos ($j = 1, \dots, n^A \in A \; y \; j = n^A + 1, \dots, n \in B$).

Por otro lado, el modelo modificado no necesita hacer uso de la variable C_{max}, y que por tanto desaparece del modelo, ya que el *makespan* no es objeto de estudio. El modelo SGST adaptado queda como se detalla a continuación:

$$min \sum_{j \epsilon A} C_{2j}^A \tag{4.7}$$

s.a.

$$C_{2j} \leq d_j \quad (n^A + 1 \leq j \leq n) \tag{4.8}$$

$$C_{1j} \geq p_{1j} \quad (1 \leq j \leq n) \tag{4.9}$$

$$C_{2j} - C_{1j} \geq p_{2j} \quad (1 \leq j \leq n) \tag{4.10}$$

$$C_{ij} - C_{ik} + M\delta_{jk} \geq p_{ij} \quad (1 \leq i \leq 2; \; 1 \leq j \leq k \leq n) \tag{4.11}$$

$$C_{ij} - C_{ik} + M\delta_{jk} \leq M - p_{ik} \quad (1 \leq i \leq 2; \; 1 \leq j \leq k \leq n) \tag{4.12}$$

$$C_{ij} \geq 0; \; 1 \leq i \leq m, 1 \leq j \leq n$$

$$\delta_{jk} \in \{0, 1\}; \; 1 \leq j \leq k \leq n$$

Como se puede observar, la función objetivo (4.7) cambia, haciendo ahora referencia a la suma de los tiempos de finalización de todos los trabajos del conjunto A.

La restricción original (4.2), referente al *makespan*, se elimina, al no trabajar con la variable C_{max}. En esta adaptación hay que añadir la restricción (4.8), que es la condición para que el conjunto B no tenga trabajos retrasados (al plantear el problema mediante el método restricción-épsilon, el objetivo de B pasa a ser una restricción en vez de estar en la función objetivo).

El resto de las restricciones tienen el mismo significado que las homólogas del modelo original, pero

teniendo en cuenta que solo hay dos máquinas. La restricción (4.9) asegura que el tiempo de finalización de un trabajo en la máquina 1 es siempre mayor o igual a su tiempo de proceso en esa máquina. El instante de inicio de un trabajo j en la máquina 2 es siempre mayor o igual que el tiempo de finalización de ese trabajo en la máquina 1, restricción (4.10). Por último, el par de restricciones dicotómicas (4.11) y (4.12): si el trabajo k precede al trabajo j en la secuencia, entonces j será procesado en la máquina i en un instante posterior al tiempo de finalización de k en dicha máquina (4.11) y (4.12) hace referencia al caso de que j preceda a k.

Así pues, observando el modelo adaptado, se puede deducir que el problema bajo estudio tiene un total de $2n + n(n-1)/2$ variables, de las cuales $2n$ variables son reales y $n(n-1)/2$ variables binarias, y $n^B + 2n + 2n(n-1)$ restricciones. En la Tabla 4-1 y Tabla 4-2 se detallan las variables y restricciones del modelo respectivamente.

Tabla 4-1. Variables del modelo matemático.

Notación	Cantidad de variables
C_{1j}	n
C_{2j}	n
δ_{jk}	$n(n-1)/2$

Tabla 4-2. Restricciones del modelo matemático.

Conjunto de restricciones	Cantidad de restricciones
(4.8)	n^B
(4.9)	n
(4.10)	n
(4.11)	$n(n-1)$
(4.12)	$n(n-1)$

5 PROGRAMACIÓN Y ANÁLISIS DE RESULTADOS MODELO EXACTO

Una vez planteado el modelo matemático que resuelve el problema de manera exacta, se procede a su implementación en lenguaje MATLAB, para la resolución de una batería de problemas con la que se puedan obtener resultados y sacar conclusiones.

Previamente, se hace una validación del modelo implementado en MATLAB con el objetivo de verificar su correcto funcionamiento.

5.1 Validación del modelo

Una vez desarrollado el modelo matemático y con el objetivo de hacer una programación efectiva en MATLAB, se decide hacer una validación de este usando diferentes *software*. Una vez implementado en MATLAB, tal y como se detalla en los siguientes apartados de este capítulo, se comparan los resultados de diferentes ejemplos sencillos con los obtenidos en Excel y GAMS 28.2 (64 bit).

Con el *solver* de Excel, se introduce explícitamente la matriz de coeficientes tecnológicos de la matriz A, los coeficientes de los objetivos, los recursos (RHS) y los sentidos de las restricciones. Debido a la complejidad del modelo y número de restricciones, se hace una comparación resolviendo un problema con 4 trabajos. Posteriormente, se hace lo mismo con GAMS, para un problema de 8 trabajos. GAMS es un *software* de alto nivel para el modelado de sistemas de optimización algebraica que permite modelar el problema de manera compacta.

Con la ayuda de estas dos herramientas, que ofrecen una visión diferente a la hora de plasmar el modelo, se consigue validar la correcta implementación de este en MATLAB, con el que se resuelve la batería de problemas.

En el Anexo A.1.1 Modelo matemático en Excel se puede encontrar la implementación del modelo en Excel y en el Anexo A.1.2 Modelo matemático en GAMS la programación en GAMS.

5.2 Resolución del modelo

El objetivo de obtener un modelo exacto de programación lineal no es otro que el de ver hasta dónde se puede llegar en cuanto al número de trabajos en el problema, así como tener referencias a la hora de construir un modelo aproximado. Como se sabe, es un problema complejo, por lo que se conoce de antemano que no es posible resolverlo de manera exacta para ejemplos grandes.

Una vez validado el problema, se propone resolver una batería de instancias de pequeño tamaño, esto es, para niveles de 4, 8, 12, 16, 20, 24, 28 y 32 trabajos, distribuyéndolos de manera equitativa entre ambos conjuntos de trabajos, tal que $n = n^A + n^B$. Para obtener muestras representativas se generan y resuelven 30 instancias diferentes para cada nivel de trabajo, generando los parámetros p_{ij} y d_j de manera aleatoria según diferentes distribuciones uniformes (Lee et al., 2011).

Los tiempos de proceso vienen dados según la distribución uniforme discreta entre 1 y 100 unidades temporales:

$$p_{ij} \sim U(1, 100).$$

De la misma manera, las fechas de entrega se generan mediante la distribución uniforme:

$$d_j \sim U(P(1 - \tau - R/2), P(1 - \tau + R/2)),$$

donde P se define como la suma de los tiempos de proceso de todos los trabajos en todas las máquinas y los parámetros τ, factor de tardanza, y R, rango de fechas de entrega, parámetros para obtener distintos escenarios de holgura, toman valores de 0,25 y 0,75 respectivamente. En Lee et al. (2011) se estudian diferentes valores y combinaciones para los parámetros τ y R ($\tau = 0{,}25, 0{,}5$ y $R = 0{,}25, 0{,}5, 0{,}75$), pero se concluye que no tiene efecto en el desempeño de los distintos modelos.

El código para la generación de los tiempos de proceso y fechas de entrega, para las 30 instancias de cada nivel de trabajo, se detalla en el Anexo A.1.3. Batería de datos en MATLAB, donde se plasman las distribuciones probabilístcas expuestas.

Debido a la complejidad del problema, el tiempo de cómputo para hallar el óptimo es cada vez mayor, creciendo de forma exponencial, a medida que aumenta el número de trabajos. Por ello, para evitar tiempos de computación excesivamente largos, se limita el tiempo máximo de ejecución a 900 segundos (15 minutos). En el caso de que para un problema se alcance dicho tiempo sin encontrar la solución óptima, se devuelve la mejor solución factible encontrada hasta el momento.

El proceso llevado a cabo para realizar la optimización de los distintos problemas se refleja en la Figura 5-1.

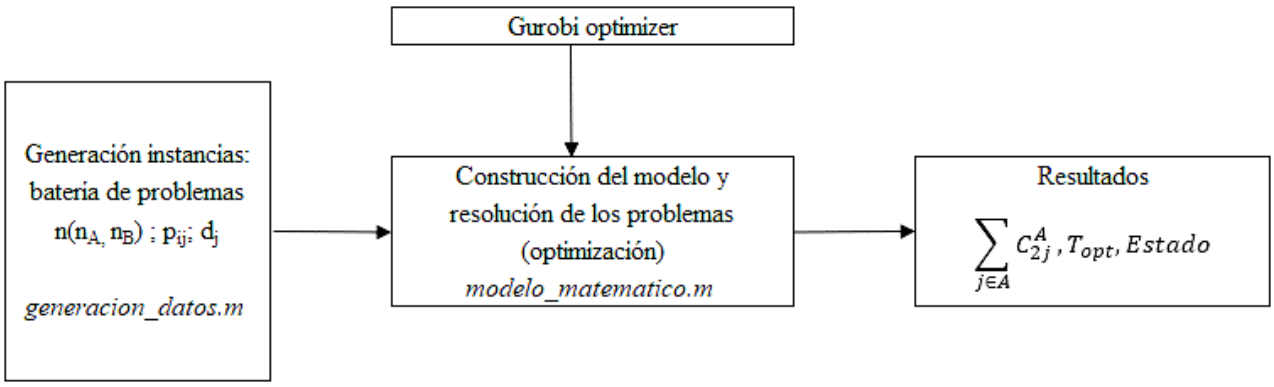


Figura 5-1. Diagrama experimento modelo exacto[2].

Cabe reseñar que para la construcción del modelo de programación lineal en MATLAB es necesario seguir las indicaciones que proporciona Gurobi para que los problemas puedan ser resueltos. Esto consta de una serie de campos y argumentos necesarios, que se pueden encontrar en la documentación del *solver*. Dicho formato tiene la estructura general siguiente:

$$\min(c' \cdot x + x' \cdot Q \cdot x + \alpha)$$

$$s.a$$

$$A \cdot x = b$$

$$l \leq x \leq u$$

$$x' \cdot Q_c \cdot x + q' \cdot x \leq \beta$$

La estructura del modelo debe contener los siguientes campos obligatorios descritos a continuación:

- model.A: la matriz de restricciones lineales del modelo.

[2] Fuente: Elaboración propia, adaptado de Aliaga (2015).

- model.obj: el vector objetivo lineal. Se debe especificar un valor para cada columna de *A*.

- model.sense: el signo de las restricciones lineales. Los valores que se permiten son <, = o >. Se debe especificar un valor para cada fila de *A*, o un único valor para especificar que todas las restricciones tienen el mismo signo. Debe ser un vector de caracteres.

- model.rhs: es el vector del término independiente de las restricciones lineales (*b* en el modelo expuesto anteriormente). Se debe especificar un valor para cada fila de *A*.

Algunos campos opcionales en el modelo son:

- model.vtype: los tipos de variables. Es un vector de caracteres. Los valores que se permiten son: “C” (continua); “B” (binario); “I” (entero); “S” (semi-continuo); “N” (semi-entero). Cuando esté presente, se debe especificar un valor para cada columna de la matriz *A*, o un solo valor para especificar que todas las variables son del mismo tipo. Cuando está ausente, cada variable se trata como una variable continua.

- model.modelsense: el objetivo de optimización. Los valores permitidos son “min” (minimizar) o “max” (maximizar). Cuando no se expresa, el objetivo del modelo por defecto es de minimización.

La estructura del resultado de la optimización del problema contiene el siguiente campo:

- result.status: el estado de la optimización, la devuelve como una cadena. Cuando el resultado deseado es “óptimo”, indica que se encontró la solución óptima para el modelo.

La estructura de los resultados de la optimización puede contener también los siguientes campos opcionales:

- result.objval: valor objetivo de la solución encontrada.

- result.runtime: el tiempo en segundos de la resolución de la optimización.

- result.x: la solución calculada. Este vector contiene un valor para cada columna de la matriz *A*.

Por tanto, el modelo de programación lineal en MATLAB se contruye utilizando estos campos y estructura. En el Anexo A.1.4. Modelo matemático en MATLAB se encuentra el desarrollo del correspondiente código.

Los resultados del experimento se recogen en un archivo *.txt*, para luego ser exportados a un archivo Excel y hacer el análisis de los resultados. Los resultados de salida son: el nombre identificativo de la instancia, la suma de los tiempos de finalización de los trabajos del conjunto *A* (en segundos), el tiempo en encontrar el óptimo o el tiempo límite impuesto (en segundos) y el estado de la solución (óptima, tiempo límite alcanzado o solución no factible, en cuyo caso los dos campos anteriores no aplican, N/A). En la Figura 5-2 se muestra un ejemplo del formato del archivo de salida.

Instancia	SUM_C_2j(A)	T_opt	Estado
4_12	703	0.266	OPTIMAL
4_13	954	0.245	OPTIMAL
4_14	1193	0.265	OPTIMAL
4_15	N/A	N/A	INFEASIBLE
4_16	1234	0.187	OPTIMAL
4_17	N/A	N/A	INFEASIBLE
4_18	1334	0.188	OPTIMAL
4_19	1258	0.203	OPTIMAL
4_20	831	0.251	OPTIMAL

Figura 5-2. Fichero de salida con resultados.

Para la resolución de los distintos problemas se ha trabajado en una plataforma PC con procesador *Intel(R) Core (TM) i7* de 1,80 GHz y sistema operativo de *64 bits Windows 10 Home* de *Microsoft Corporation*. Las versiones de los diferentes *software* son: *MATLAB R2019b* y *Gurobi 9.0.2 (win64)*.

5.3 Análisis estadístico de los resultados

En este apartado se hace un análisis de las 240 instancias resueltas (30 por cada nivel de trabajo), en términos de tiempos de cómputo, así como de optimalidad del modelo estudiado, el SGST correspondiente a la familia Manne propuesto por Stafford et al. (2005) adaptado al problema descrito en el apartado 2.3. Con esto, se puede tener conocimiento de para qué tamaño de problemas es el modelo válido.

En la Tabla 5-1 se expone un resumen de los resultados obtenidos de la resolución de la batería pequeña. Los resultados en bruto, sin procesar, se pueden encontrar en el Anexo A.2. Resultados modelo matemático.

Tabla 5-1. Resumen resultados batería pequeña modelo exacto.

Número de trabajos, *n*	Instancias factibles	Optimalidad*	Tiempo de cómputo [s]		IC95%	
			Media	Desviación estándar	Superior	Inferior
4	67 %	100 %	0,08	0,03	0,09	0,07
8	90 %	100 %	0,15	0,04	0,16	0,13
12	93 %	100 %	1,24	1,59	1,81	0,67
16	100 %	97 %	122,52	206,52	196,42	48,61
20	100 %	10 %	868,44	113,33	908,99	827,89
24	100 %	0 %	900,13	0,13	900,18	900,08
28	100 %	0 %	900,16	0,14	900,21	900,11
32	100 %	0 %	900,22	0,20	900,29	900,15

*Nota: Optimalidad calculada en base a las instancias factibles.

En la Tabla 5-1 se recogen los resultados obtenidos para cada nivel de trabajos estudiado, n. En primer lugar, se presenta el porcentaje de instancias factibles originadas, ya que, en algunos casos, según las fechas de entrega generadas aleatoriamente, se pueden encontrar problemas que no tienen solución al tener fechas muy ajustadas para esos tamaños (véase en el apartado 5.2 la distribución empleada). En segundo lugar, el grado de optimalidad alcanzado en las instancias factibles, viniendo dado por aquellos problemas que se han resuelto en un tiempo de cómputo menor al límite impuesto (umbral temporal máximo de 900 segundos). Por otro lado, se presentan los tiempos de cómputo, tanto la media como la desviación estándar, y los límites del intervalo de confianza al 95 %. Algunas conclusiones que se pueden sacar son las siguientes:

- Como se puede observar en la segunda columna de la Tabla 5-1, en los problemas con menor número de trabajos, $n = 4$, $n = 8$ y $n = 12$, es posible encontrar instancias que no son factibles. Esto es debido a que las fechas de entrega, generadas aleatoriamente según el tamaño del problema, pueden salir muy ajustadas. Sin embargo, es posible encontrar el óptimo de todas las instancias factibles dentro del límite de tiempo impuesto.

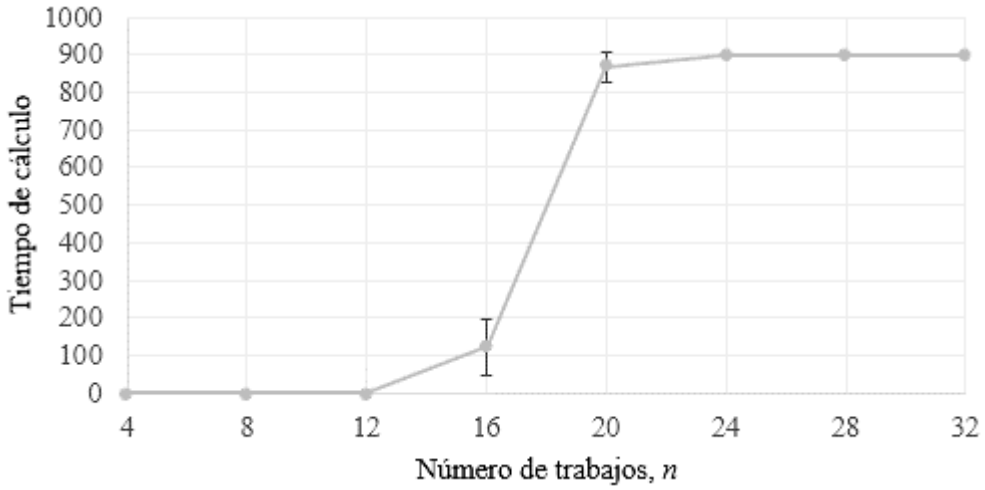
- A partir de $n = 16$ trabajos, todas las instancias que se han generado son factibles. Sin embargo, debido al tamaño de los problemas y al tiempo máximo impuesto, no llega a ser posible encontrar el óptimo en algunos o incluso en todos los casos a medida que se aumenta el número de trabajos, ya que se alcanza el tiempo límite impuesto.

Los análisis de los tiempos de cómputo medios, desviación estándar e intervalos de confianza se realizan omitiendo aquellos casos que no tienen solución factible. Otras conclusiones son:

- Cuando el tamaño de las instancias es pequeño, $n < 16$, el tiempo de resolución requerido por el modelo es muy pequeño, solo superando 1 s de media cuando $n = 12$.

- Sin embargo, para $n \geq 16$, este análisis se ve desvirtuado debido a que se empieza a alcanzar el tiempo máximo de cómputo impuesto, de ahí que la media sea prácticamente igual a 900 s y la desviación estándar muy pequeña en esos casos.

Se puede observar en la Gráfica 5-1 cómo los tiempos de cómputo empiezan a aumentar exponencialmente con n, hasta llegar a $n = 16$, donde a partir de entonces la mayoría de los problemas alcanzan el límite de cómputo, por lo que la gráfica se aplana.



Gráfica 5-1. Tiempo de cálculo e IC95% en función de *n*.

De la misma manera, la desviación estándar o los intervalos de confianza cobran sentido en los niveles de trabajo menores, donde no se alcanza dicho límite. Para analizar estos resultados en detalle se presentan los diagramas de cajas y bigotes.

Los diagramas de cajas y bigotes, Figura 5-3, son gráficos que muestran la distribución estadística de una variable en base a sus cuartiles, de manera que se puede extraer gráficamente algunas conclusiones, como la dispersión o simetría de los datos. Consiste en una caja rectangular dividida por un segmento horizontal, que indica donde se encuentra la mediana y su relación con los cuartiles primero y tercero. El lado inferior del rectángulo hace referencia al primer cuartil y el lado superior al tercer cuartil, por lo que la altura de la caja muestra el recorrido intercuartílico. También se muestra la media de la variable, normalmente con una pequeña aspa.

Esta caja se ubica a escala sobre unas líneas verticales, que se conocen como bigotes, y que tienen como extremos los valores mínimo y máximo del conjunto de datos, siempre y cuando estos valores no difieran de la media más de una vez y media del rango intercuartílico. Estos bigotes tienen un límite de prolongación, delimitado por dos líneas horizontales cortas, de modo que cualquier dato que no se encuentre dentro de este rango es marcado e identificado individualmente. Los datos que se encuentran fuera de los límites suelen llamarse *outliers*, valores atípicos que muestran una gran distancia respecto a la media.

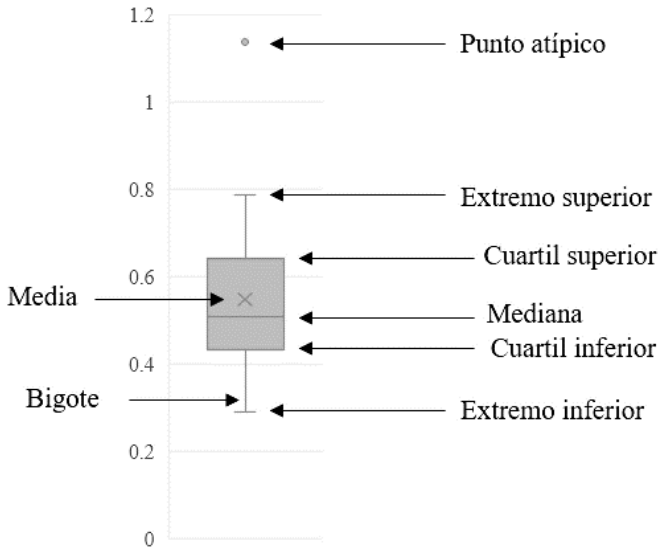


Figura 5-3. Diagrama de cajas y bigotes.

En la Figura 5-4 se pueden observar los diagramas de cajas y bigotes correspondientes a los niveles de trabajo en los que no se alcanza el tiempo límite de cálculo. Se puede observar como para $n = 12$ la dispersión en el tiempo de cálculo es mayor y empiezan a aparecer valores atípicos.

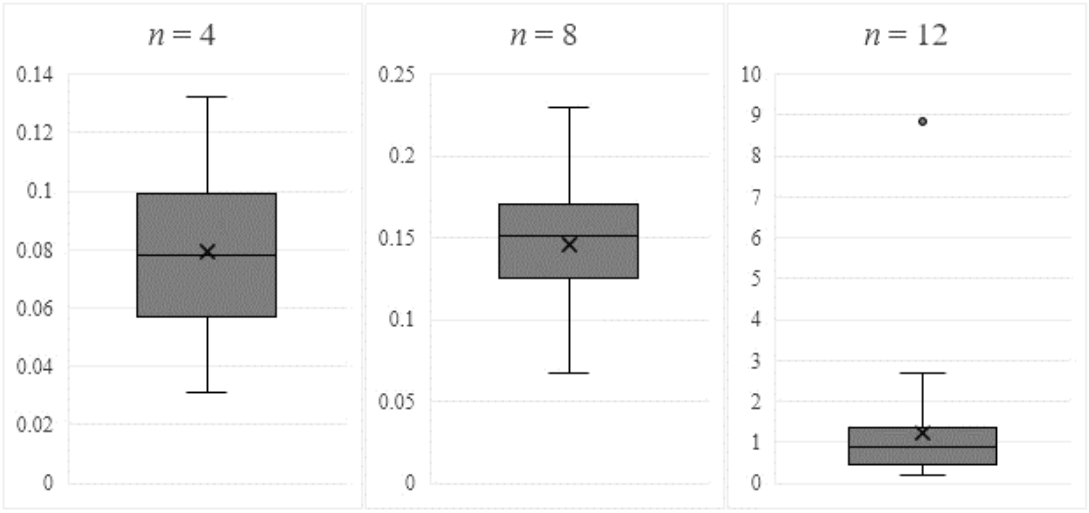


Figura 5-4. Diagrama de cajas y bigotes: tiempo de cálculo según según *n*.

Teniendo en cuenta el grado de optimalidad obtenido en los resultados, se decide reducir la batería pequeña para los próximos experimentos y comparaciones con la metaheurística a los niveles de trabajo $n = 12$ y $n = 16$, con alto grado de optimalidad y $n = 20$ y $n = 24$, cuyas soluciones pueden servir como aceptables límites superiores.

6 METAHEURÍSTICA

Debido a la complejidad computacional, en este capítulo se desarrolla una técnica metaheurística para resolver el problema propuesto de manera aproximada, con la que se puedan resolver problemas grandes en tiempos de cómputo reducidos y con soluciones cercanas a las óptimas. Se comparan diferentes estrategias a la hora de construir el algoritmo para obtener la combinación que ofrezca mejores resultados, desarrollando diferentes experimentos para la comparación y análisis de estos.

En el Anexo B. Código Simulated Annealing se puede encontrar la implementación completa del código en MATLAB, correspondiente a la metaheurística estudiada y las diferentes versiones.

6.1 Algoritmos de mejora iterativa

Como se ha visto en los apartados 2.4 y 2.5, para los problemas del tipo NP-*Hard* no es posible obtener la solución óptima del problema (o los tiempos de cómputo son muy altos) debido a la complejidad y tamaño que pueda tener el mismo (número de trabajos y/o máquinas).

Existen muchos procesos de optimización de interés industrial que involucran funciones que presentan un gran número de soluciones locales y, por tanto, es muy difícil determinar una solución óptima usando técnicas de optimización deterministas (Ponce-Ortega y Hernández-Pérez, 2019). Cualquier aproximación sin garantía formal de desempeño puede ser considerada heurística. Estos enfoques son muy útiles en situaciones prácticas si no hay mejores métodos disponibles (Johns y Brucker, 1996).

A diferencia de otras técnicas heurísticas, las metaheurísticas tratan de no quedarse atrapadas en óptimos locales, por lo que se orienta la búsqueda en cada momento según la evolución del proceso. El método de resolución se basa en una solución inicial factible, a partir de la cual se van generando soluciones parecidas, que son elegidas y sustituidas según vayan mejorando el objetivo a optimizar propuesto. Existen muchas técnicas del tipo metaheurísticas, entre las que destacan los algoritmos genéticos, búsqueda tabú, colonias de hormigas, *Iterated Greedy* o *Simulated Annealing*, entre otros.

Uno de los métodos más exitosos para atajar complicados problemas de optimización combinatoria es el conocido como búsqueda local, metaheurística que proporciona soluciones factibles que no

tienen garantías de ser óptimas, útil a la hora de resolver problemas de optimización discretos (todos los problemas que no permiten interrupciones, *preemptions,* son problemas discretos).

Un problema de optimización discreto puede ser descrito como (Johns y Brucker, 1996): para un conjunto finito S y una función $c: S \to \mathbb{R}$, hay que encontrar una solución $s^* \in S$ con

$$c(s^*) \leq c(s) \; \forall \; s \in S.$$

Búsqueda local es un proceso iterativo que se mueve de una solución en S a otra tanto como sea necesario. Para hacer una búsqueda sistemática en S, los posibles movimientos desde una solución s hacia la siguiente solución deberían estar restringidos de alguna manera. Para describir dicha restricción, se introduce la estructura de vecindad $N: S \to 2^S$ en S. Para cada $s \in S$, $N(s)$ describe el subconjunto de soluciones que pueden ser alcanzadas en un único paso moviéndose desde s. El conjunto $N(s)$ es llamado vecindad de s.

Usando estas definiciones, un método de búsqueda local puede ser definido como: en cada iteración se empieza con una solución $s \in S$ y se elige $s' \in N(s)$. Según los valores de $c(s)$ y $c(s')$, se elige una solución inicial para la siguiente iteración. Basados en los diferentes criterios usados para la elección de la solución inicial de la siguiente iteración, se obtienen diferentes tipos de técnicas de búsqueda local.

La opción más sencilla es la de tomar la solución con el menor valor de la función objetivo. Esta elección lleva a un conocido método de mejora iterativa que puede ser formulado como se expone a continuación:

Algoritmo de mejora iterativa

1. Elige una solución inicial $s' \in S$;
2. REPITE
3. $\quad s := s'$;
4. $\quad$ Genera la mejor solución $s' \in N(s)$;
5. HASTA $c(s) \geq c(s')$.

Este algoritmo termina con alguna solución s^*. En general, s^* es solo un mínimo local con respecto a su vecindad N (solución tal que ninguna solución vecina es mejor que esta solución) y por tanto puede diferir considerablemente del mínimo global. Dicho lo cual, con este tipo de algoritmo no se evita quedar atrapado en un óptimo local.

6.2 Simulated Annealing

Un método que busca evitar quedar atrapado en un mínimo local es el ya mencionado algoritmo *Simulated Annealing* (SA). Fue desarrollado por varios investigadores en la mitad de los años 80 (Kirkpatrick et al., 1983), siendo esta una buena técnica para muchos problemas complejos de optimización combinatoria y, por tanto, la elegida para estudiar el problema abordado en este trabajo. Sin embargo, SA tiene un famoso linaje, derivando del algoritmo de Metropolis, desarrollado por Metropolis et al. (1953).

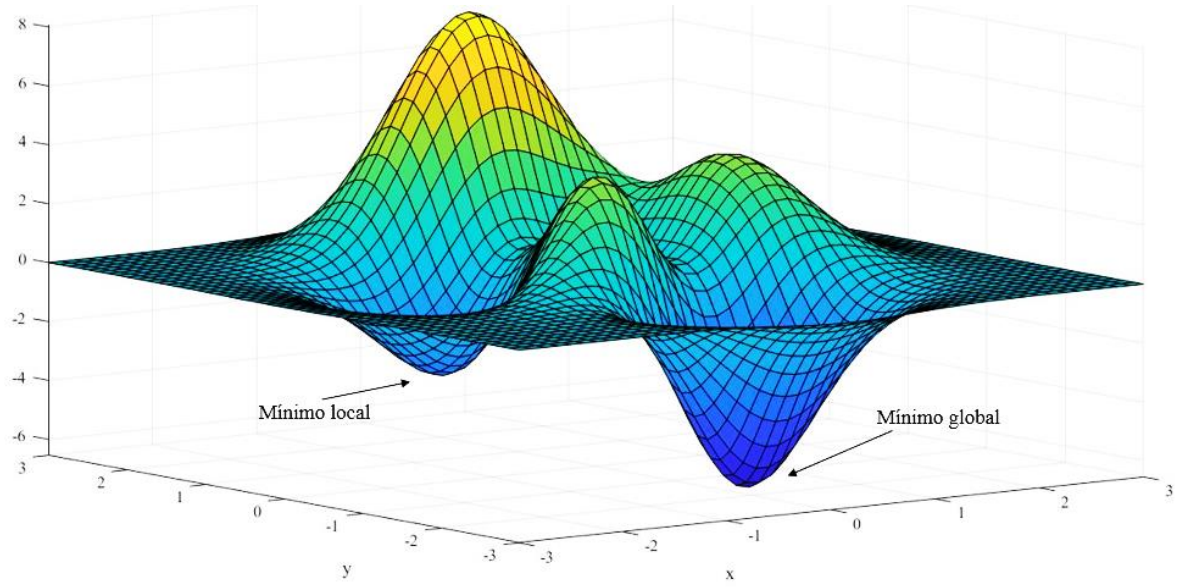


Figura 6-1. Función con mínimo local y global.

SA es una técnica metaheurística con enfoque estocástico, basada en una analogía del recocido del metal u otros sólidos, cuyo objetivo es alcanzar la configuración de energía interna mínima absoluta (global).

Para describir este fenómeno, se considera un sólido con estructura cristalina que es calentado hasta la temperatura de recocido, y luego, es enfriado hasta solidificarse de nuevo para cambiar el material a una estructura deseada. Si la temperatura en el proceso de recocido decrece rápidamente, y no se pasa suficiente tiempo con cada temperatura, el proceso podría quedar atrapado en un mínimo local de estado de energía interna, apareciendo irregularidades en la estructura cristalina del metal enfriado, siendo el nivel de energía del sólido mucho mayor que una estructura cristalina perfecta. Por otro lado, si el material es enfriado lentamente, el nivel de energía sería mínimo. Además, cuando el material está caliente, la estructura molecular es más débil y, por tanto, más susceptible a

cambiar. Cuando el material se enfría, la estructura molecular es más fuerte y menos susceptible a cambiar (Luke, 2011; Ponce-Ortega y Hernández-Pérez, 2019).

El algoritmo de Metropolis simula el material como un sistema de partículas, el cual se somete a un proceso de enfriamiento, disminuyendo gradualmente la temperatura del sistema hasta que converja a un estado estable o de congelamiento. Posteriormente, Kirkpatrick et al. (1983) tomaron la idea del algoritmo de Metropolis y lo aplicaron a problemas de optimización. La idea es usar la simulación del proceso de recocido para buscar soluciones factibles y converger a una solución óptima o cercana a la óptima.

Un esquema simple del SA se puede formular como sigue (adaptando lo propuesto en Johns y Brucker, 1996):

1. Considera una solución inicial s';
2. HASTA que tengas un congelamiento
    a) DESDE $i \coloneqq 1$ hasta el número de vecinos considerados
        i. Sea s' un vecino de s;
        ii. Si $c(s) < c(s')$, entonces $s \coloneqq s'$;
            En otro caso, actualiza $s \coloneqq s'$ con cierta probabilidad;
    b) Enfría la temperatura;
3. DEVUELVE s.

La simulación en el algoritmo calcula un nuevo estado de energía del sistema. Si la energía ha disminuido, el sistema se mueve hacia ese estado, y si la energía ha aumentado, el nuevo estado es aceptado usando cierta probabilidad. Un cierto número de iteraciones se llevan a cabo con cada temperatura para luego disminuirla, aunque también se puede simular disminuyendo la temperatura en cada iteración.

Es un método aleatorio porque:

- s' es elegido aleatoriamente de $N(s)$.
- En la iteración i, s' es elegida con probabilidad

$$min\left\{1, exp\left(-\frac{c(s') - c(s)}{T_i}\right)\right\},$$

donde T_i es una secuencia de parámetros de control positivos, llamada temperatura, con $\lim_{i\to\infty} T_i = 0$.

SA depende de una estrategia aleatoria para diversificar la búsqueda, generando soluciones aleatorias en la vecindad de la solución actual y evaluando la función objetivo, donde se usa el criterio de Metropolis para aceptar un movimiento. En este sentido, la interpretación de esta probabilidad es que los movimientos en los que la función objetivo decrece son siempre aceptados (si $c(s') \leq c(s)$, entonces s es remplazado por s' con probabilidad 1), mientras que los movimientos que incrementan la función objetivo son aceptados pero con cierta probabilidad (si, por otro lado, $c(s') > c(s)$, entonces s es remplazado por s' con probabilidad $exp(-(c(s') - c(s))/T_i))$.

La aceptación de una nueva solución se basa en el valor de la función de densidad de probabilidad de la distribución Bolzman-Gibs. Esta función es interesante en dos sentidos:

- Primero, si s' es mucho peor que s, la fracción es grande, y por tanto la probabilidad es cercana a 0. Si s' es cercano a s, la probabilidad es cercana a 1. Así que, si s' no es mucho peor que s se sigue seleccionando a s' con cierta probabilidad.

- Segundo, se tiene un parámetro ajustable T. Cuando la temperatura es alta, en las primeras iteraciones, muchos movimientos en los que la función objetivo aumenta son aceptados, evitando de este modo quedar atrapado prematuramente en mínimos locales y permitiendo al algoritmo explorar más soluciones en búsqueda del óptimo global. Sin embargo, cuando la temperatura es baja, el parámetro T se acerca a cero (cuando i aumenta) y la probabilidad de aceptar peores soluciones decrece, rechazando casi todas las soluciones peores.

  En la Tabla 6-1 y Tabla 6-2 se muestran unos ejemplos de la probabilidad para aceptar soluciones dadas dos temperaturas diferentes, pero sujetos a los mismos cambios en la función objetivo.

Tabla 6-1. Ejemplo probabilidad de aceptación para temperatura alta[3].

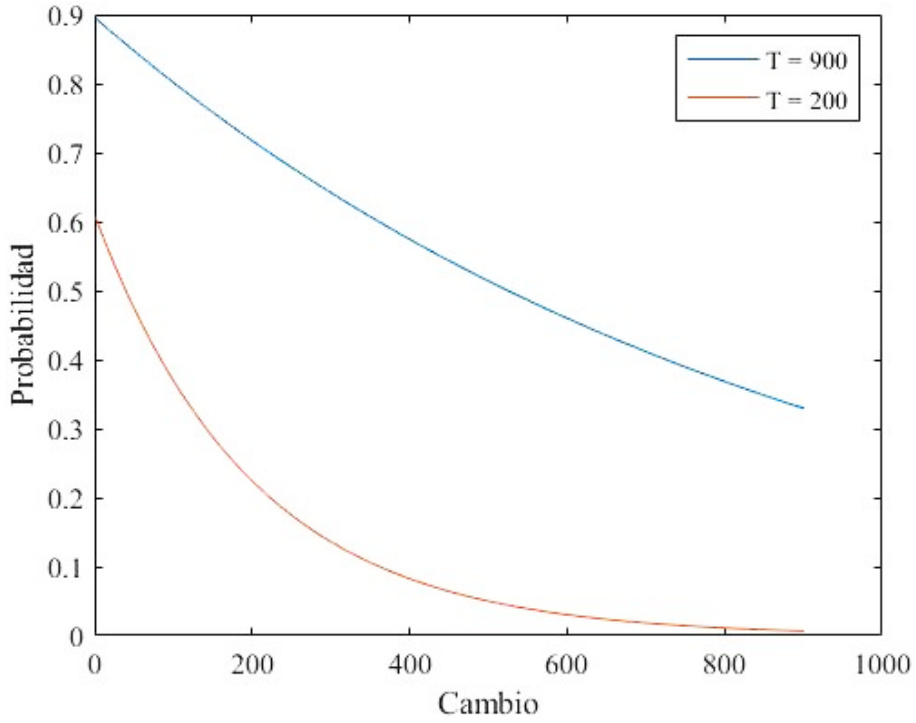
Cambio	Temperatura	Probabilidad
100	900	0,895
300	900	0,716
600	900	0,513
900	900	0,368

[3] Fuente: Elaboración propia, adaptado de Liang (2020).

Tabla 6-2. Ejemplo probabilidad de aceptación para temperatura baja[3].

Cambio	Temperatura	Probabilidad
100	200	0,606
300	200	0,223
600	200	0,050
900	200	0,011

En la Gráfica 6-1. Ejemplo probabilidad de aceptación según temperatura. se puede ver el ejemplo de la función de probabilidad dependiendo de las temperaturas anteriormente propuestas y diferentes cambios en la función objetivo. Se puede observar que al tener una temperatura menor y para el mismo cambio en la función objetivo, se tiene siempre menor probabilidad para aceptar cambios y, además, al haber un cambio mayor, la probabilidad de aceptación se reduce rápidamente, siendo esta muy baja.



Gráfica 6-1. Ejemplo probabilidad de aceptación según temperatura.

Dowsland (1995) presentan una tabla, como la Tabla 6-3. Analogía entre proceso físico de recocido y algoritmo., que muestra cómo el proceso físico de recocido puede ser relacionado con los procesos de optimización y así convertir cualquier problema cuando se pueda cumplir esta analogía.

Tabla 6-3. Analogía entre proceso físico de recocido y algoritmo[4].

Simulación termodinámica	Optimización combinatoria
Estados del sistema	Soluciones factibles
Energía	Valor función objetivo
Cambio de estado	Soluciones vecinas
Temperatura	Parámetro de control
Estado congelado	Solución heurística

A continuación, se detalla el algoritmo, planteando el pseudo-código, donde random [0, 1] denota una función aleatoria uniformemente distribuida entre los valores 0 y 1. Además, la secuencia T_i es creada por una función *g* (conocida como *cooling schedule*), por ejemplo $T_{i+1} = g(T_i)\ \forall\ i$:

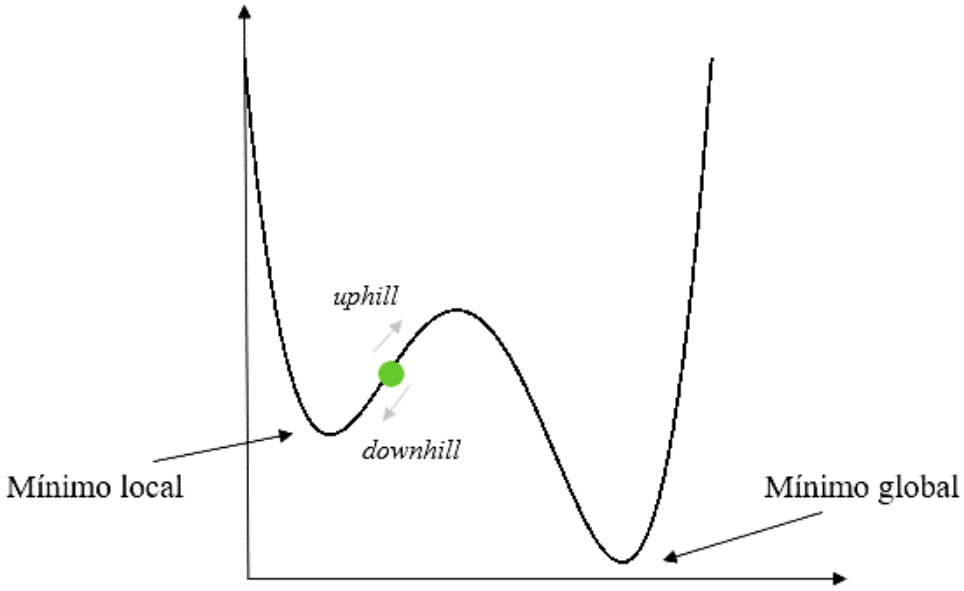
Algoritmo *Simulated Annealing*

1. Considera una temperatura inicial $T_0 > 0$; inicializa un contador $i := 1$;
2. Considera una solución inicial $s \in S$;
3. $s^{best} := s$;
4. $c(s^{best}) := c(s)$;
5. REPITE
6. Genera aleatoriamente una solución $s' \in N(s)$;
7. SI $c(s') < c(s)$ % *downhill*
8. ENTONCES $s := s'$;
9. SI $c(s)$ < c(*best)*
10. ENTONCES $s^{best} := s$;
11. EN OTRO CASO, SI random [0, 1] < exp $(-\frac{c(s')-c(s)}{T_i})$ % *uphill*
12. ENTONCES $s := s'$;
13. $T = g(T_i)$; % reduce Temperatura
14. $i := i + 1$;
15. HASTA criterio de parada
16. DEVUELVE s^{best}, $c(s^{best})$.

En el algoritmo anterior se ha mencionado los conceptos de *downhill* y *uphill*, que en un problema de minimización hacen referencia al proceso de aceptar una mejor solución o una peor respectivamente.

[4] Fuente: Dowsland (1995).

Como se puede observar en la Gráfica 6-2, en la que se representa una función de dos variables, un movimiento *uphill* puede ayudar a evitar un mínimo local y acercar al algoritmo hacia la solución óptima, por lo que la importancia de este tipo de movimientos es crucial en la bondad del algoritmo SA.



Gráfica 6-2. Ejemplo búsqueda *uphill*/*downhill* en SA[5].

6.3 Especificaciones del algoritmo

6.3.1 Solución inicial

Cuando se trabaja con SA es común empezar con una solución inicial totalmente aleatoria y dejar al proceso mejorar sobre ella. De hecho, en algunos documentos teóricos (Lundy y Mess, 1986) se puede interpretar que la elección de una solución inicial no tiene efecto en la calidad de la solución final, ya que la solución converge al óptimo global independientemente de la solución inicial considerada. Sin embargo, también hay excepciones que demuestran que empezar con una solución inicial construida mediante alguna heurística puede ayudar a obtener una mejor solución, ya sea en la calidad de la solución final o en el tiempo de cálculo que se emplea.

En este trabajo se proponen diferentes métodos para generar secuencias iniciales (GSI), usando reglas de despacho, para tratar de obtener mejores soluciones:

[5] Fuente: Elaboración propia, adaptado de Shi, Fernando y Kandoz (2012).

- GSI1: en esta secuencia los trabajos del conjunto B son posicionados primero de acuerdo con la regla de fechas de entrega más tempranas (EDD: *earliest due dates*), seguidos de los trabajos del conjunto A según la regla de tiempos de procesamiento más cortos (SPT: *shortest processing time*) en la máquina 1.

- GSI2: en esta secuencia los trabajos del conjunto B son posicionados primero de acuerdo con la regla de fechas de entrega más tempranas (EDD: *earliest due dates*), seguidos de los trabajos del conjunto A según la regla de tiempos de procesamiento más cortos (SPT: shortest processing time) en la máquina 2.

- GSI3: en esta secuencia los trabajos del conjunto B también son posicionados primero de acuerdo con la regla de fechas de entrega más tempranas (EDD: *earliest due dates*), pero difiere de GSI1 y GSI2 en que los trabajos del conjunto A son ordenados según la regla SPT basada en la suma de los tiempos de procesamiento en ambas máquinas.

En los tres métodos propuestos se posicionan en los primeros lugares los trabajos del conjunto B con el objetivo de partir desde una solución factible, en la que ninguno de los trabajos de dicho conjunto se finalice tras su fecha acordada de entrega, seguidos de los trabajos del conjunto A, como se indica en la Figura 6-2.

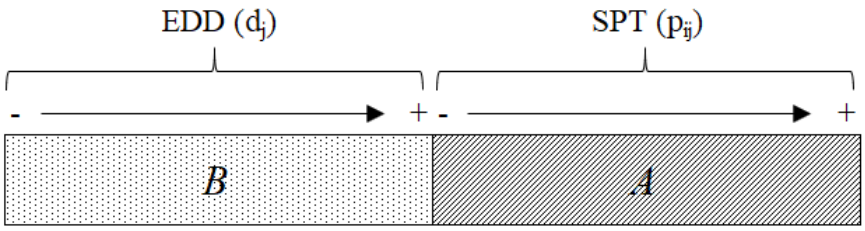


Figura 6-2. Estrategia general para generar soluciones iniciales.

6.3.2 Generación de vecinos

El espacio de soluciones consiste en todas las posibles permutaciones de trabajos ($n!$). El algoritmo SA intenta minimizar la función objetivo examinando parcialmente el espacio de soluciones, haciendo movimientos de permutación en permutación. Una permutación alcanzable con un solo movimiento se conoce como vecino. El modo en que los vecinos son generados tiene un importante efecto en la eficiencia del SA.

Tres modos diferentes de generación de secuencias candidatas (GSC) o vecinas han sido considerados:

- GSC1: intercambio aleatorio por parejas, *random pairwise interchange*.
    1. Genera aleatoriamente dos números enteros j y $k \in \{1, 2, \dots, n\}: j \neq k$.
    2. Define $I = \min(j, k), J = \max(j, k)$.
    3. Intercambia los trabajos en las posiciones I y J en la secuencia de la siguiente manera: sea S'_i el trabajo en la posición i de la nueva secuencia y S_i el trabajo en la posición i de la antigua secuencia.

$$S'_i = S_i \quad i = 1, 2, \dots, I-1; I+1, \dots, J-1; J+1, \dots, n$$

$$S'_J = S_I, S'_I = S_J$$

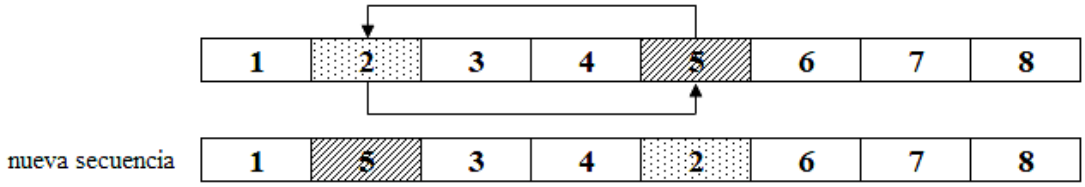


Figura 6-3. Generación nueva secuencia por intercambio de par aleatorio.

- GSC2: intercambio aleatorio de par de trabajos consecutivos, *consecutive pairwise interchange*.
    1. Genera aleatoriamente un número entero $j \in \{1, 2, \dots, n\}$ y sea $k: k = j + 1$.
    2. Intercambia los trabajos en las posiciones I y J en la secuencia de la siguiente manera: sea S'_i el trabajo en la posición i de la nueva secuencia y S_i el trabajo en la posición i de la antigua secuencia.

$$S'_i = S_i \quad i = 1, 2, \dots, I-1; J+1, \dots, n$$

$$S'_J = S_I, S' = S_J$$

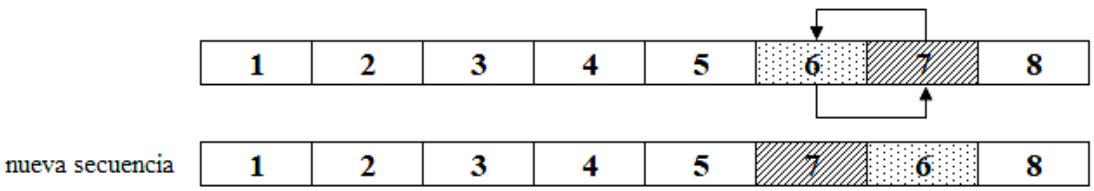


Figura 6-4. Generación nueva secuencia por intercambio de par consecutivo.

- GSC3: movimiento de inserción, *insert moves*.
    1. Genera aleatoriamente dos números enteros j y $k \in \{1, 2, \dots, n\}: j < k$.
    2. Inserta el trabajo en la posición k en la posición justamente anterior a j. sea S'_i el trabajo en la posición i de la nueva secuencia y S_i el trabajo en la posición i de la antigua secuencia.

$$S'_i = S_i \quad i = 1, 2, \dots, j-1; k+1, \dots, n$$

$$S'_j = S_k$$

$$S'_i = S_{i-1} \quad i = j+1, \dots, k$$

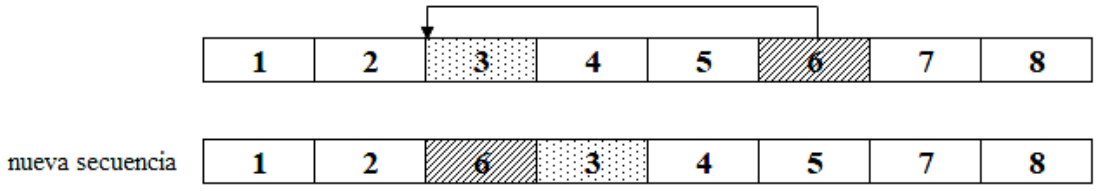


Figura 6-5. Generación nueva secuencia por inserción.

6.3.3 Función de probabilidad y *cooling schedule*:

Es la parte crucial de la técnica, siendo el criterio usado por SA para decidir, en cada una de las iteraciones llevadas a cabo, si aceptar o no una nueva solución encontrada que ofrece peor resultado. Como se comenta al principio del apartado 6.2, esta función tiene la forma

$$P = e^{\left(-\frac{\Delta c}{T_i}\right)},$$

donde Δc es la diferencia en el valor de la función objetivo entre la solución actual y la nueva generada y T_i es la temperatura en la iteración i.

El *cooling schedule* es el conjunto de constantes que va decreciendo periódicamente, haciendo la analogía de disminución de temperatura en el proceso de recocido del metal y sirviendo de parámetro de control. Dicha función, $T_{i+1} = g(T_i)\ \forall\ i$, consiste en cuatro componentes:

1. Temperatura inicial: debe ser lo suficientemente alta para permitir un movimiento a casi cualquier estado vecino, permitiendo al algoritmo explorar una mayor parte del espacio de soluciones. Si no es así, la solución final podría ser la misma, o muy cercana, a la solución inicial. Sin embargo, si la temperatura inicial es demasiado alta, se podría incurrir en una

búsqueda totalmente aleatoria en casi toda la simulación. Posteriormente, va decreciendo, intentando llevar al algoritmo a un área más productiva del espacio de búsqueda, donde es más probable encontrar mejores soluciones.

El problema es encontrar la correcta temperatura inicial, pero no se conoce ningún método para encontrar la temperatura inicial idónea para cualquier tipo de problema.

2. Temperatura final: usualmente se deja decrecer la temperatura hasta cero. Sin embargo, esto podría hacer al algoritmo gastar mucho tiempo computacional, especialmente cuando se utiliza una función del tipo geométrica.

   En la práctica no es necesario dejar a la temperatura alcanzar el cero absoluto, ya que a medida que se va acercando a cero, la probabilidad de aceptar peores movimientos son prácticamente las mismas que si la temperatura fuera cero. Por tanto, se suele usar otro tipo de criterio de finalización, siempre haciendo que la temperatura sea mayor o igual que cero.

3. Decremento de temperatura: una vez que se tiene la temperatura inicial se necesita ir pasando de una a otra, es decir, se necesita disminuir dicha temperatura y que en algún momento se alcance el criterio de parada. El modo en que se hace este decremento es muy importante para el éxito del algoritmo. Normalmente, se pueden hacer un gran número de iteraciones para pocas temperaturas, un número pequeño de iteraciones para muchas temperaturas, o un compromiso entre ambas.

   Algunas funciones sencillas y típicas para disminuir la temperatura son:

   - Reducción lineal: $T = T - \alpha$
   - Reducción geométrica: $T = T \cdot \alpha, \alpha < 1$

4. Iteraciones en cada temperatura: la decisión final es cuántas iteraciones hacer en cada temperatura. Una decisión obvia es la de hacer un número constante de iteraciones en cada temperatura.

   Otro método, propuesto en primer lugar por Lundy y Mess (1989), es el de hacer una sola iteración en cada temperatura, disminuyendo la temperatura lentamente. La función tipo es de la forma $T = T/(1 + \beta T)$, donde β es un valor apropiadamente pequeño.

Teniendo en cuenta los cuatro parámetros de la función de control, *cooling schedule*, se proponen dos funciones probabilísticas (FP). Estas funciones se basan en funciones de enfriamiento

monotónico multiplicativo, en las que la temperatura T en el ciclo i se calcula multiplicando la temperatura inicial T_0 por un factor que decrece con respecto al ciclo i, el cual se decide que corresponda con cada iteración, adoptando la propuesta de Lundy y Mess (1986) de hacer decrementos de temperatura en cada iteración. Se consideran dos variantes: enfriamiento multiplicativo lineal y enfriamiento multiplicativo cuadrático.

- FP1: esta función probabilística se basa en un enfriamiento multiplicativo lineal, cuya estructura es similar a la propuesta por Lundy y Mess (1986) y en la que la temperatura es reducida en cada iteración *i* a razón:

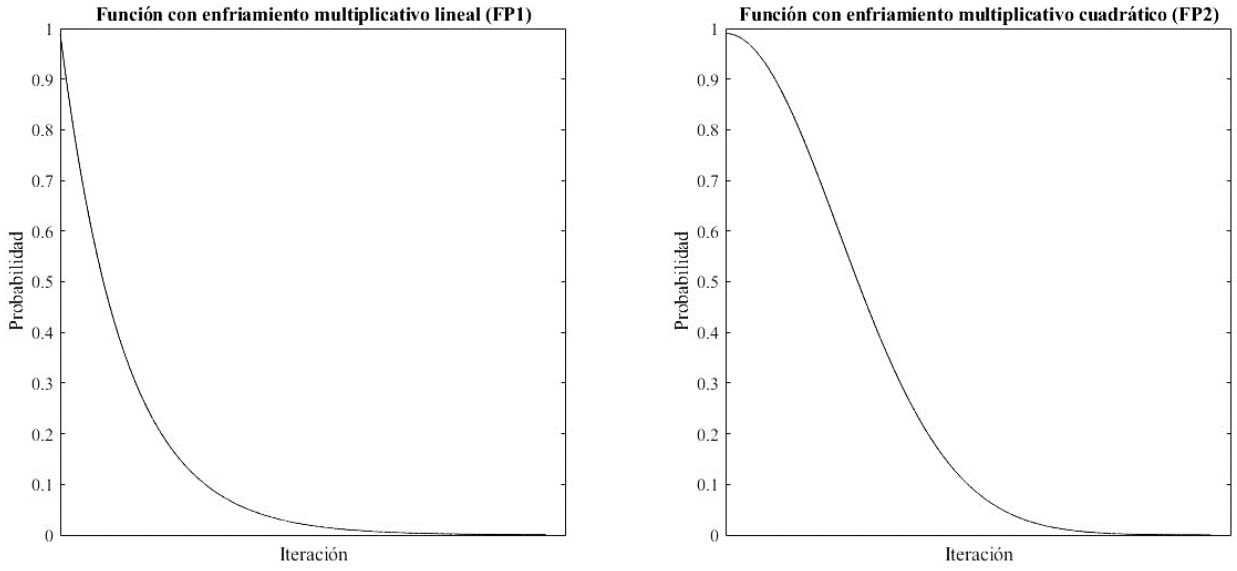
$$T_i = \frac{T_0}{1 + \alpha \cdot i} \quad \alpha > 0.$$

  El decremento de la temperatura se realiza multiplicando la temperatura inicial T_0 por un factor que decrece en proporción inversa al ciclo de temperatura i, que en este caso se corresponde a una iteración.

- FP2: esta función probabilística se basa en un enfriamiento multiplicativo cuadrático, cuya estructura es similar a la anterior, pero en este caso el decremento de temperatura se hace multiplicando la temperatura inicial T_0 por un factor que decrece en proporción inversa al cuadrado de cada ciclo o iteración:

$$T_i = \frac{T_0}{1 + \alpha \cdot i^2} \quad \alpha > 0.$$

En la Gráfica 6-3 se representa la forma general de estas dos funciones, mostrando la evolución según se avanza en el algoritmo y se va reduciendo la temperatura.



Gráfica 6-3. Funciones de enfriamiento multiplicativo.

Para la elección de los parámetros T_0 y α se sigue lo propuesto por Kirkpatrick et al. (1983), donde se sugiere un preprocesado para la elección de los parámetros, de tal manera que la ratio de las soluciones aceptadas y el número total de iteraciones sea igual a un cierto valor.

Análogamente, Karasakal y Köksalan (2000) proponen que la temperatura inicial debe ser elegida de modo que al menos un determinado porcentaje de soluciones propuestas sean aceptadas. Dicho valor es determinado a través de experimentos, en los que se comprueba que las soluciones obtenidas son buenas.

Para poder decidir si las soluciones obtenidas son buenas o no, se hace uso de los resultados exactos obtenidos en la resolución de la batería pequeña de problemas mediante el modelo matemático. Diferentes temperaturas iniciales y parámetros α se prueban en las dos funciones propuestas, resolviendo problemas con diferente número de trabajos. Tras la realización de los experimentos previos, se concluye que un porcentaje entre el 30 % y 35 % de soluciones aceptadas conduce a resultados exitosos del algoritmo.

Siguiendo la línea de los experimentos propuestos, se pretende deducir unas funciones de probabilidad que no dependan del tamaño del problema. Según la fórmula general de dicha función, $P = \exp(-\Delta c / T_i)$, la probabilidad de aceptar un cierto incremento en el objetivo, tiempo total de finalización de los trabajos del conjunto A, es el mismo a una determinada temperatura independientemente del tamaño del problema. Sin embargo, como el tiempo total de este objetivo es

mayor a medida que se aumenta el tamaño del problema, n, la probabilidad de aceptar dicho incremento en la función objetivo debería ser mayor para problemas mayores. Para lograr esto, se escalan dichos parámetros en función del número de trabajos, por lo que, tanto T_0 como α, se calibran en función del número de trabajos, n.

Para determinar T_0 y α se acotan, en primer lugar, sus valores, haciendo una representación gráfica de las funciones y determinando su validez según la forma que tomen las funciones. Posteriormente, en segundo lugar, se prueban valores intermedios de los dos rangos establecidos. En la Tabla 6-4, se muestran los valores finalmente probados.

Tabla 6-4. Valores de T_0 y α probados en las funciones probabilísticas.

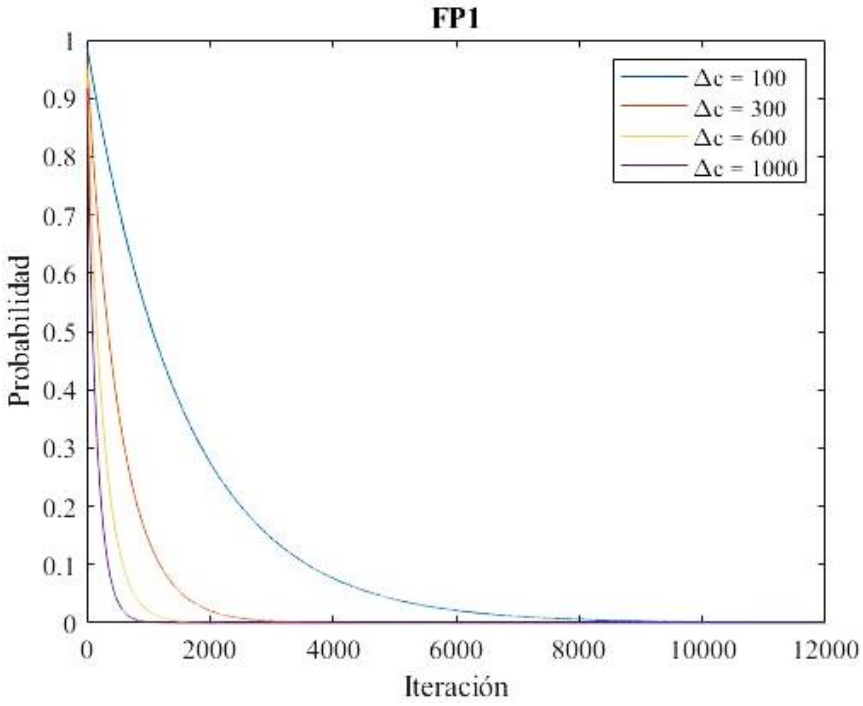
T_0		α	
FP1	FP2	FP1	FP2
$425 \cdot n$		$1{,}08/n$	$1/(10000 \cdot n)$
$500 \cdot n$		$1{,}28/n$	$1/(6250 \cdot n)$
$625 \cdot n$		$1{,}5/n$	$1/(4000 \cdot n)$
$800 \cdot n$		$2/n$	$1/(1000 \cdot n)$

A continuación, se muestran las dos funciones propuestas tras concretrar los correspondientes valores, de acuerdo con los experimentos anteriormente detallados:

- FP1: para la función probabilística con enfriamiento multiplicativo lineal se obtienen los mejores resultados para $T_0 = 625 \cdot n$ y $\alpha = 1{,}28/n$, por lo que finalmente la función queda como

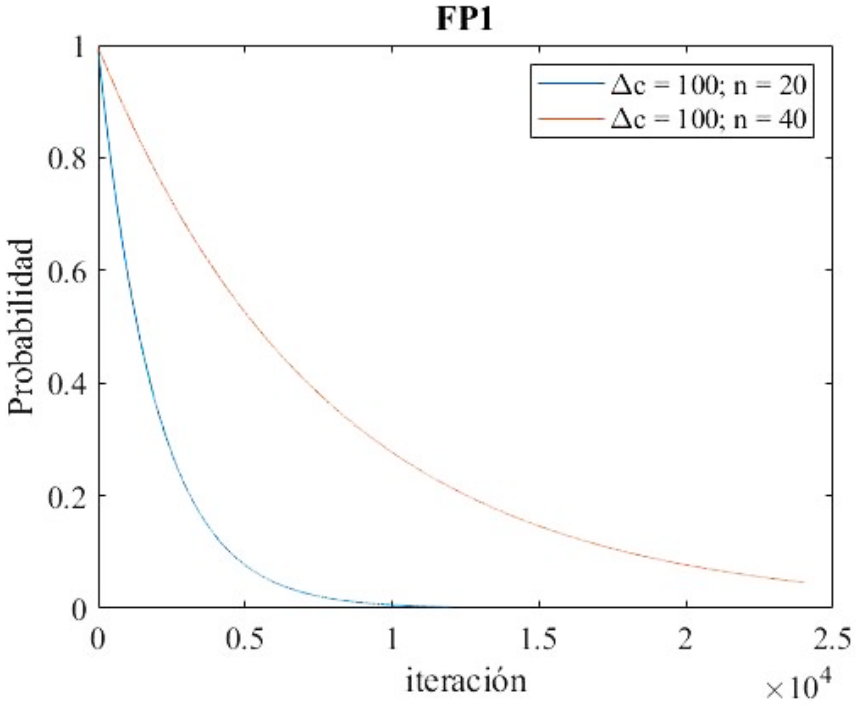
$$P(i) = e^{\left(-\frac{c(s') - c(s)}{\frac{625 \cdot n}{1 + \frac{1{,}28}{n} \cdot i}} \right)}.$$

  En la Gráfica 6-4 se muestra un ejemplo de la función FP1, mostrando diferentes incrementos en el valor del objetivo para un número de trabajos $n = 20$.



Gráfica 6-4. Función FP1 para $n = 20$.

Se puede observar que, para cada temperatura o iteración, la probabilidad de aceptar cambios menores es mayor. Además, como se puede observar en la Gráfica 6-5, si el tamaño del problema aumenta, la probabilidad para un mismo cambio también aumenta.

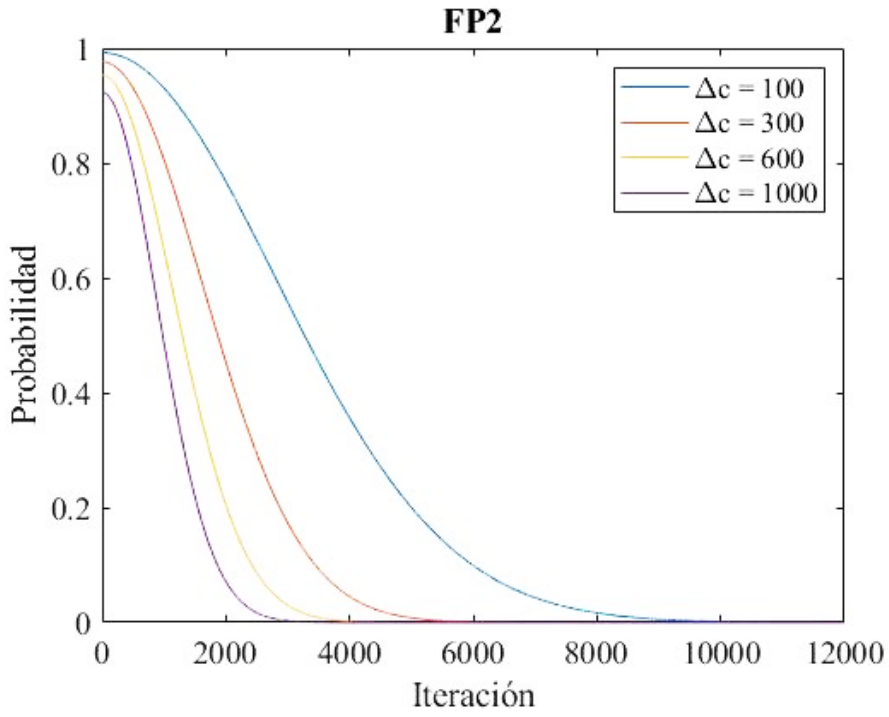


Gráfica 6-5. Comparación función FP1 según *n*.

- FP2: para la función probabilística con enfriamiento multiplicatuvo cuadrático se obtienen los mejores resultados para $T_0 = 625 \cdot n$, donde se ha mantenido la misma temperatura inicial que para FP1, y $\alpha = \frac{1}{6250} \cdot n$, por lo que finalmente la función queda como

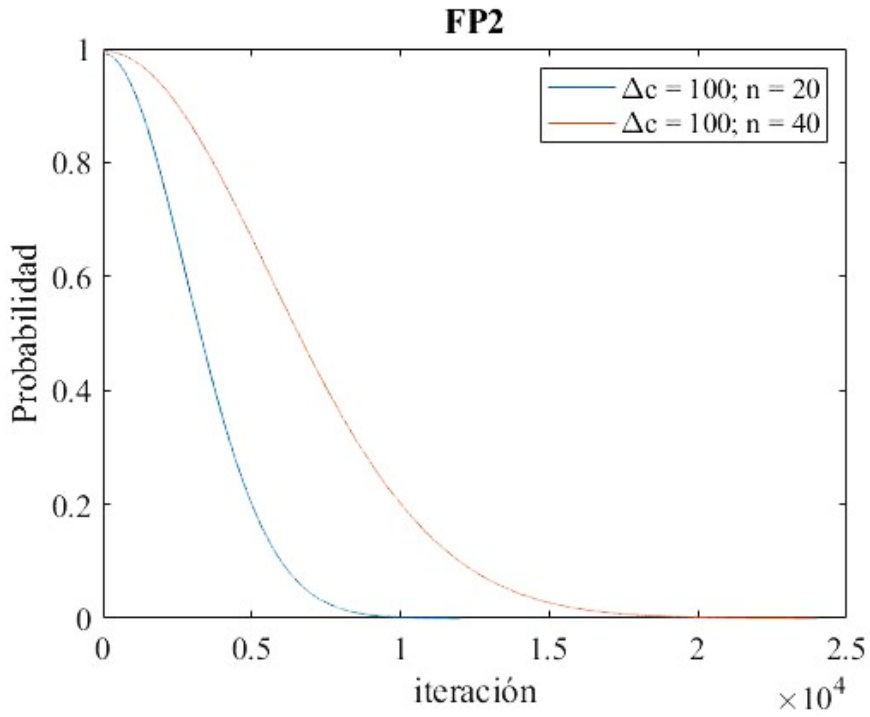
$$P(i) = e^{\left(-\frac{\frac{c(s')-c(s)}{625 \cdot n}}{1+\frac{1}{6250 \cdot n} \cdot i^2}\right)}.$$

En la Gráfica 6-6 se muestra un ejemplo de la función FP2, mostrando diferentes incrementos en el valor del objetivo para un número de trabajos $n = 20$.



Gráfica 6-6. Función FP2 para *n* = 20.

Al igual que para FP1, se puede observar que, para cada temperatura o iteración, la probabilidad de aceptar cambios menores es mayor. Además, como se puede observar en la Gráfica 6-7, si el tamaño del problema aumenta, la probabilidad para un mismo cambio también aumenta, ya que las variables están parametrizadas para adaptarse según el tamaño del problema y que el algoritmo siga funcionando de manera coherente.



Gráfica 6-7. Comparación función FP2 según *n*.

6.3.4 Criterio de parada

Como se comenta en la sección 6.3.4, un criterio de parada puede ser el alcanzar una temperatura final igual a cero, aunque típicamente se opta por otro tipo de criterios basados en un tiempo máximo de computación, en un número máximo de iteraciones o en algún otro criterio que establezca que la solución se encuentra congelada.

Para el criterio de parada se establece el propuesto por Lee et al. (2011), tras comprobar en experimentos preliminares la bondad de las soluciones. Los experimentos indican que tras $600 \cdot n$ iteraciones la calidad de las soluciones que se obtienen es buena, ya que se observa repetidamente la estabilización de estas. Para ello, se ha vuelto a usar los resultados exactos obtenidos con el modelo matemático en la batería pequeña de problemas.

Gracias a que la función de probabilidad, o el *cooling schedule*, se escala para cada tamaño de problema en función de n, se comprueba que la temperatura final después de $600 \cdot n$ iteraciones es siempre mayor que cero.

6.4 Metaheurísticas propuestas

Teniendo en cuenta todas las opciones planteadas en el apartado 6.3, se decide hacer un diseño

factorial completo, esto es, de manera combinatoria cruzar todas las opciones de generación de secuencia inicial (GSI), generación de secuencia candidata (GSC) y función de probabilidad (FP), para estar seguros de cúal es la selección correcta del algoritmo que ofrece mejores resultados.

Por tanto, se tiene un total de 18 algoritmos posibles, del tipo SA, los cuales quedan recogidos y enumerados en la Tabla 6-5.

Tabla 6-5. Metaheurísticas propuestas.

Algoritmo	Secuencia Inicial (SI)	Generación de Secuencia Candidata (GSC)	Función Probabilística (FP)
SA1	SI1	GSC1	FP1
SA2	SI1	GSC1	FP2
SA3	SI1	GSC2	FP1
SA4	SI1	GSC2	FP2
SA5	SI1	GSC3	FP1
SA6	SI1	GSC3	FP2
SA7	SI2	GSC1	FP1
SA8	SI2	GSC1	FP2
SA9	SI2	GSC2	FP1
SA10	SI2	GSC2	FP2
SA11	SI2	GSC3	FP1
SA12	SI2	GSC3	FP2
SA13	SI3	GSC1	FP1
SA14	SI3	GSC1	FP2
SA15	SI3	GSC2	FP1
SA16	SI3	GSC2	FP2
SA17	SI3	GSC3	FP1
SA18	SI3	GSC3	FP2

6.5 Modificaciones

En este apartado se propone una serie de modificaciones con el objetivo de estudiar posibles mejoras en la obtención de soluciones. Estas modificaciones se plantean para implementarlas en el algoritmo que se deleccione de los expuestos en la Tabla 6-5, por ser el que mejor rendimiento general ofrezca de todos los estudiados.

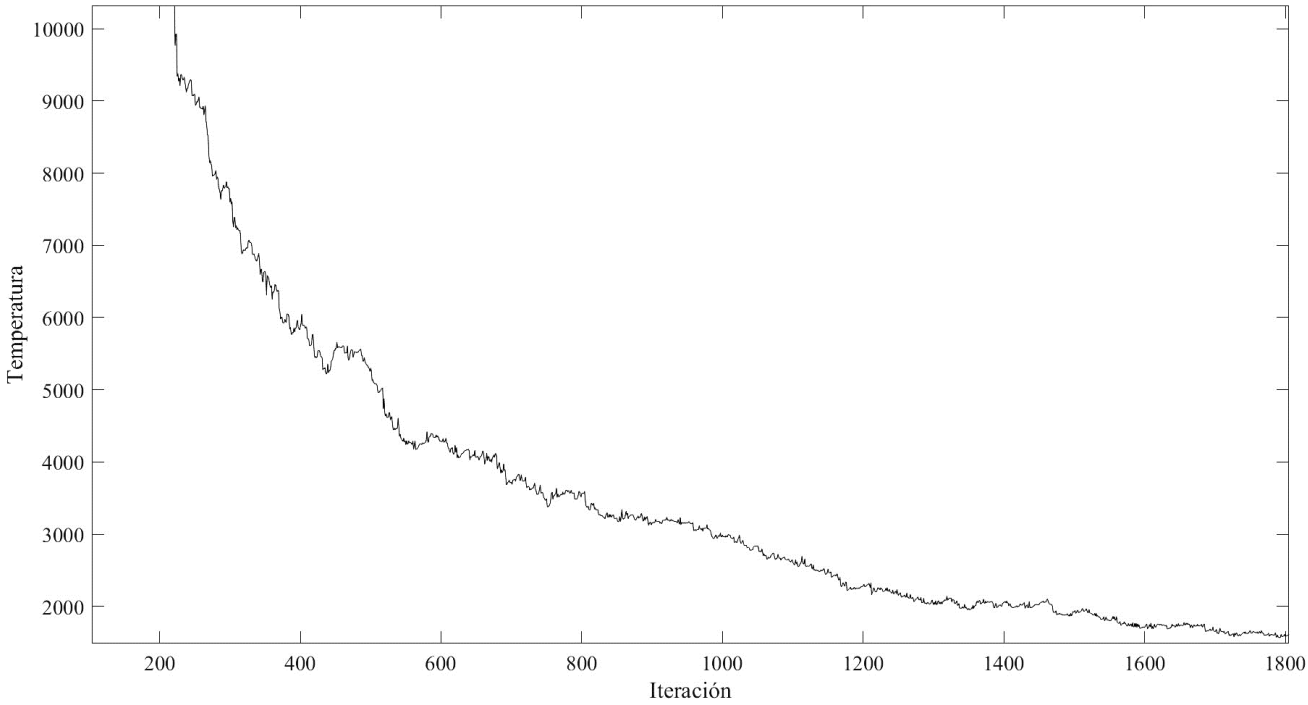
6.5.1 Función de probabilidad adaptativa

A diferencia de las dos funciones propuestas en la sección 6.3.3, FP1 y FP2, ahora se propone usar una función con un enfriamineto adaptativo no monótono. En el enfriamiento adaptativo no

monótono, la temperatura del sistema T en cada transición de estado se calcula multiplicando el valor de la temperatura T_i, obtenido por cualquiera de los criterios anteriores, por un factor adaptativo μ basado en la diferencia entre el valor de la función objetivo de la solución propuesta, $c(s_i')$, y el mejor objetivo logrado hasta ese momento por el algoritmo, $c(s^{best})$. Por tanto, la función de cambio de temperatura o *cooling schedule* queda como

$$T = \mu T_i = \left(1 + \frac{c(s_i') - c(s^{best})}{c(s_i')}\right) T_i.$$

Nótese que se verifica la desigualdad $1 < \mu < 2$. Este factor μ significa que cuanto mayor es la diferencia entre la solución actual propuesta y la mejor solución lograda hasta el momento, mayor es la temperatura y, en consecuencia, los saltos de energía permitidos. Este criterio es una variante del propuesto por Locatelli (2000), y puede usarse en combinación con cualquiera de los criterios anteriores para calcular T_i. Por tanto, la curva de enfriamiento se caracteriza por un comportamiento aleatorio fluctuante comprendido entre la curva definida por T_i y su valor doble $2 \cdot T_i$. En la Gráfica 6-8 se puede ver un ejemplo de las fluctuaciones en la temperatura para adaptarse a las circunstancias de la búsqueda.



Gráfica 6-8. Ejemplo de curva de enfriamiento adaptativa no monotónica.

6.5.2 Reinicio en la búsqueda local

Algunas veces es mejor volver a una solución que fue significativamente mejor en vez de tratar siempre de generar nuevas soluciones desde el estado actual. Este proceso se denomina reinicio (*restart*) del algoritmo *Simulated Annealing* y tiene como objetivo reconducir la búsqueda local. Para hacer eso se establece $s = s^{best}$ y $c(s) = c(s^{best})$, pudiendo en algunos casos reestablecer también la temperatura. La decisión de hacer dicho reinicio puede estar basada en diferentes criterios, entre los que cabe destacar el reinicio basado en un número fijo de iteraciones, en función de si la solución actual es muy grande en comparación con la mejor solución obtenida hasta el momento, reinicio aleatorio, etc.

El mecanismo de reinicio utilizado en este trabajo es muy simple. El algoritmo se reinicia si la mejor solución encontrada hasta el momento, s^{best}, no mejora durante un número fijo de iteraciones, en cuyo caso se establece $s = s^{best}$ y $c(s) = c(s^{best})$, para ayudar al algoritmo a moverse entre soluciones cercanas a la mejor.

Para establecer el número de iteraciones límite se hacen distintas pruebas, modificando dicho valor y los tamaños de los problemas, para finalmente establecer un valor, en función del número de trabajos, de $150 \cdot n$.

Un reinicio parecido al propuesto puede encontrarse en Addou et al. (2018), donde el criterio se basa en un tiempo límite fijo sin mejorar la solución.

7 EXPERIMENTOS Y ANÁLISIS DE RESULTADOS METAHEURÍSTICA

En este capítulo se desarrollan los experimentos llevados a cabo para obtener el algoritmo que mejor respuesta de a la resolución del problema $PF2|\sum_{j\epsilon B} T_j^B = 0|\sum_{j\epsilon A} C_{2j}^A$, en términos de calidad de la solución y tiempo computacional necesario para el cálculo.

En primer lugar, se presenta el estudio factorial completo comparando todas las técnicas propuestas aplicadas a ciertos niveles de trabajos, con el objetivo de extraer conclusiones en cuanto a posible dominancia de ciertas propuestas. Los mejores algoritmos son posteriormente estudiados con la batería pequeña de problemas y finalmente, se estudian con la batería grande aquellos que dan mejor rendimiento.

7.1 Diseño factorial completo

Como se presenta en los apartados 6.3 y 6.4, son 18 las distintas versiones del SA propuestas, tras hacer una combinación de las 3 técnicas de generación de secuencia inicial (GSI), las 3 técnicas de generación de secuencia candidata (GSC) y las 2 funciones probabilísticas (FP) de aceptación.

Para poder concluir posibles patrones de rendimiento entre las técnicas propuestas, se lleva a cabo un primer experimento preliminar en el que se estudian todas las propuestas sobre los niveles de trabajos $n = 16$ y $n = 20$. Estos niveles pertenecen a la batería pequeña y de los cuales ya se tienen los resultados exactos y límites superiores respectivamente, por lo que se hace uso de las mismas instancias generadas para resolver el problema de manera exacta y así poder comparar.

Para hacer el estudio, se compara cada valor obtenido en la resolución de cada algoritmo con su correspondiente valor exacto o aproximado obtenido con Gurobi, usando el porcentaje de la desviación relativa, PDR. Para cada instancia el valor del PDR se cálcula con la fórmula

$$PDR_{inst} = \frac{C_{2j}^{A\ inst} - C_{2j}^{A\ Grb}}{C_{2j}^{A\ Grb}} \cdot 100,$$

donde $C_{2j}^{A\ inst}$ hace referencia, para cada instancia, al valor del tiempo de finalización total del conjunto A obtenido mediante el algoritmo y $C_{2j}^{A\ Grb}$ su correspondiente valor óptimo o aproximado (límite superior) obtenido con Gurobi al resolver el modelo matemático de programación lineal.

Para obtener un valor promedio de las desviaciones porcentuales para cada nivel de tamaño de instancias se utiliza la media de dichos valores, denotada por PDRM, cuya fórmula es

$$PDRM = \frac{\sum PDR_{inst}}{N_{instancias}},$$

donde PDR_{inst} es el valor de la desviación porcentual de cada instancia y $N_{instancias}$ es el número de instancias estudiadas en ese nivel de trabajos, que para todos los tamaños tiene un valor igual a 30.

En la Tabla 7-1 se muestran los resultados finales del diseño factorial completo, mostrando los valores del PDRM para cada algoritmo y nivel de trabajo, así como las desviaciones típicas medias y los tiempos de cálculo medios. Para obtener una visión más global se añaden las dos últimas columnas que muestran el comportamiento medio de los algoritmos en ambos niveles de trabajo.

Tabla 7-1. Resumen resultados: diseño factorial completo.

	$n = 16$			$n = 20$			Total	
	PDRM	$\bar{\sigma}$	Tiempo [s]	PDRM	$\bar{\sigma}$	Tiempo [s]	Media PDRM	Media Tiempo [s]
SA1	0,0013	0,0070	0,3135	-0,0593	1,2586	0,4102	-0,0290	0,3618
SA2	0,0194	0,0975	0,2878	0,0266	1,1889	0,3619	0,0230	0,3249
SA3	6,6757	7,9295	0,1434	8,1931	9,0964	0,1742	7,4344	0,1588
SA4	2,8509	3,8190	0,1426	5,6417	6,9963	0,1653	4,2463	0,1540
SA5	5,8406	6,2365	0,6371	4,6518	5,9974	0,9843	5,2462	0,8107
SA6	5,1954	5,7624	0,5711	4,9097	6,6162	0,7178	5,0525	0,6444
SA7	0,0181	0,0975	0,2917	0,0464	1,1062	0,4198	0,0323	0,3558
SA8	0,0234	0,0905	0,2808	0,0610	1,2121	0,3667	0,0422	0,3237
SA9	5,8673	8,6556	0,2631	8,9654	8,1103	0,3534	7,4163	0,3083
SA10	5,6416	9,0207	0,1505	7,0289	6,9526	0,1766	6,3353	0,1635
SA11	4,3357	5,6740	0,6229	5,0436	6,1043	0,8689	4,6897	0,7459
SA12	3,2321	4,1307	0,5877	4,7562	6,3548	0,7856	3,9941	0,6867
SA13	0,0370	0,1273	0,3112	0,0577	1,2597	0,3693	0,0473	0,3403
SA14	0,0228	0,0985	0,2929	-0,0704	1,1296	0,3492	-0,0238	0,3211
SA15	8,9322	10,8016	0,2752	7,0984	8,4739	0,3712	8,0153	0,3232
SA16	5,1258	6,6481	0,1452	8,7818	9,4210	0,1851	6,9538	0,1651
SA17	6,0193	7,1550	0,6412	4,1098	4,7853	0,8685	5,0646	0,7549
SA18	3,2579	4,0294	0,5482	5,4331	6,4435	0,8228	4,3455	0,6855

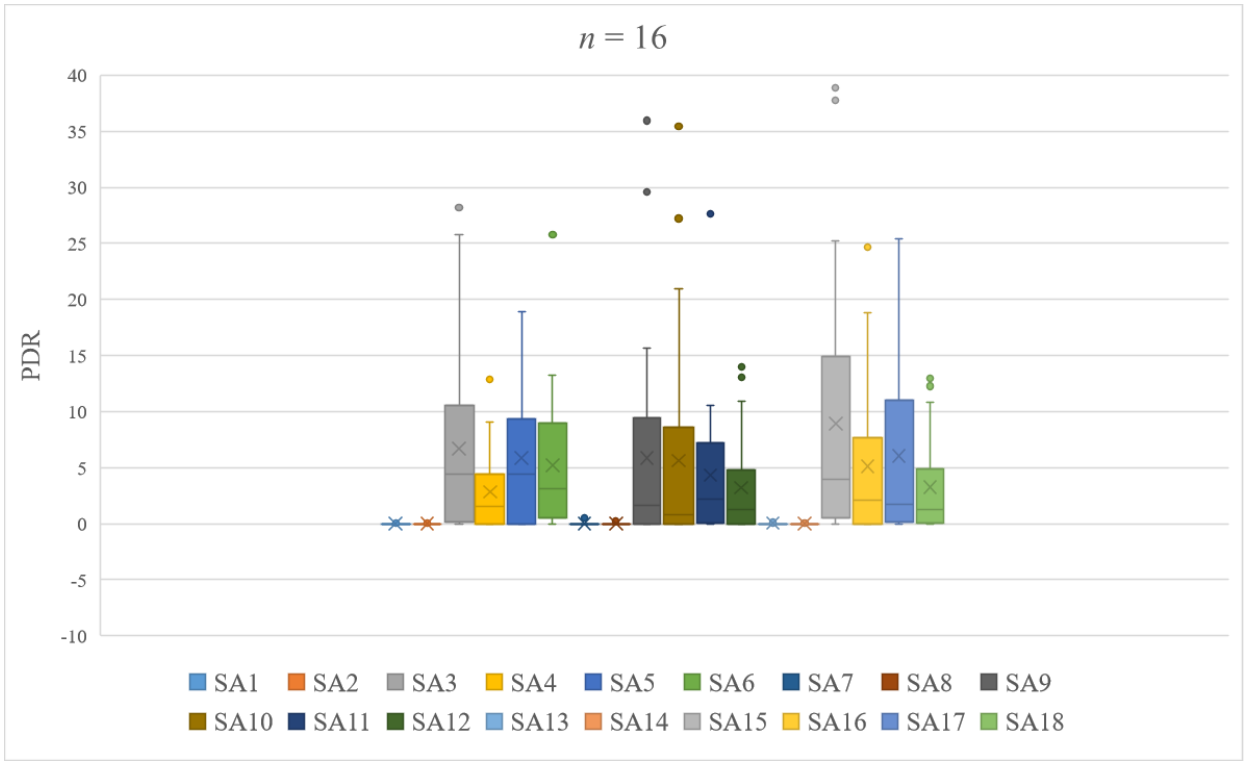
Las dos últimas columnas de la Tabla 7-1 muestra la media del PDRM de los dos tamaños de problema analizados y de los tiempos medios de cómputo. De la columna que muestra la media de las desviaciones porcentuales, PDRM, para los dos niveles de trabajo se puede extraer un comportamiento de dominancia de aquellos algoritmos que utilizan la técnica de generación de secuencias candidatas GS1, es decir SA1, SA2, SA7, SA8, SA13 y SA14. Se puede observar como las desviaciones son prácticamente cero en estos seis algoritmos e incluso para el nivel de $n = 20$, del cual se tiene un límite superior, se consiguen mejores soluciones en algunos casos (PDRM negativo). El resto de los algoritmos presentan unos resultados con desviaciones significativas, comparadas con los anteriormente citados.

En cuanto al rendimiento en términos de tiempo de cálculo, los algoritmos que usan la técnica GS1 vuelven a mostrar resultados muy competitivos con respecto a los otros que, si bien no son los más rápidos en resolver, están más cercanos a los mejores tiempos que a los peores.

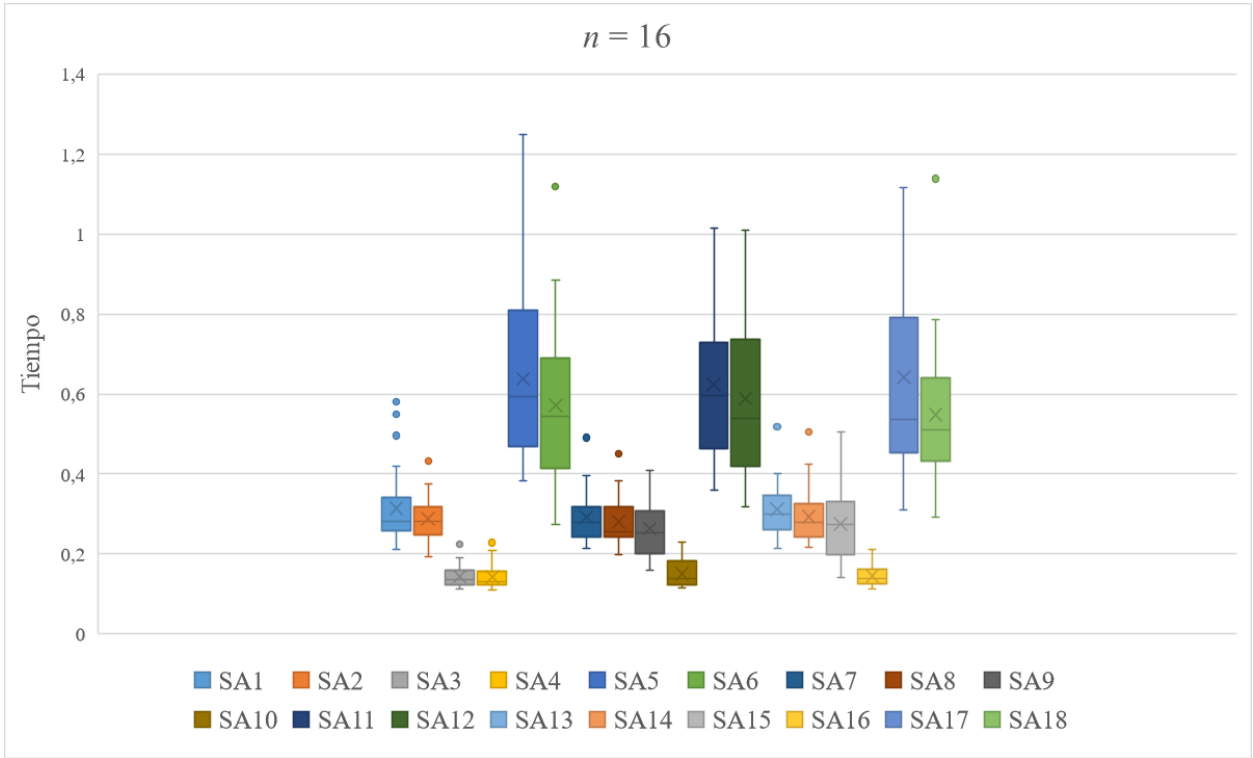
A continuación, en las gráficas Gráfica 7-1, Gráfica 7-2, Gráfica 7-3 y Gráfica 7-4, se presentan los resultados en diagramas de cajas y bigotes para poder ver los resultados y comparativas de manera más gráfica:

- En las gráficas Gráfica 7-1 y Gráfica 7-3 se puede apreciar como los algortimos con GS1 dominan claramente sobre los otros en términos del PDR, ofreciendo también mucho mejores resultados en cuanto a desviaciones y valores atípicos, pudiendo observar en los otros algoritmos medias de desviaciones desde 2,8 % a 8,9 % en el caso de $n = 16$ y desde 4,1 % a 8,9 % en el caso de $n = 20$.

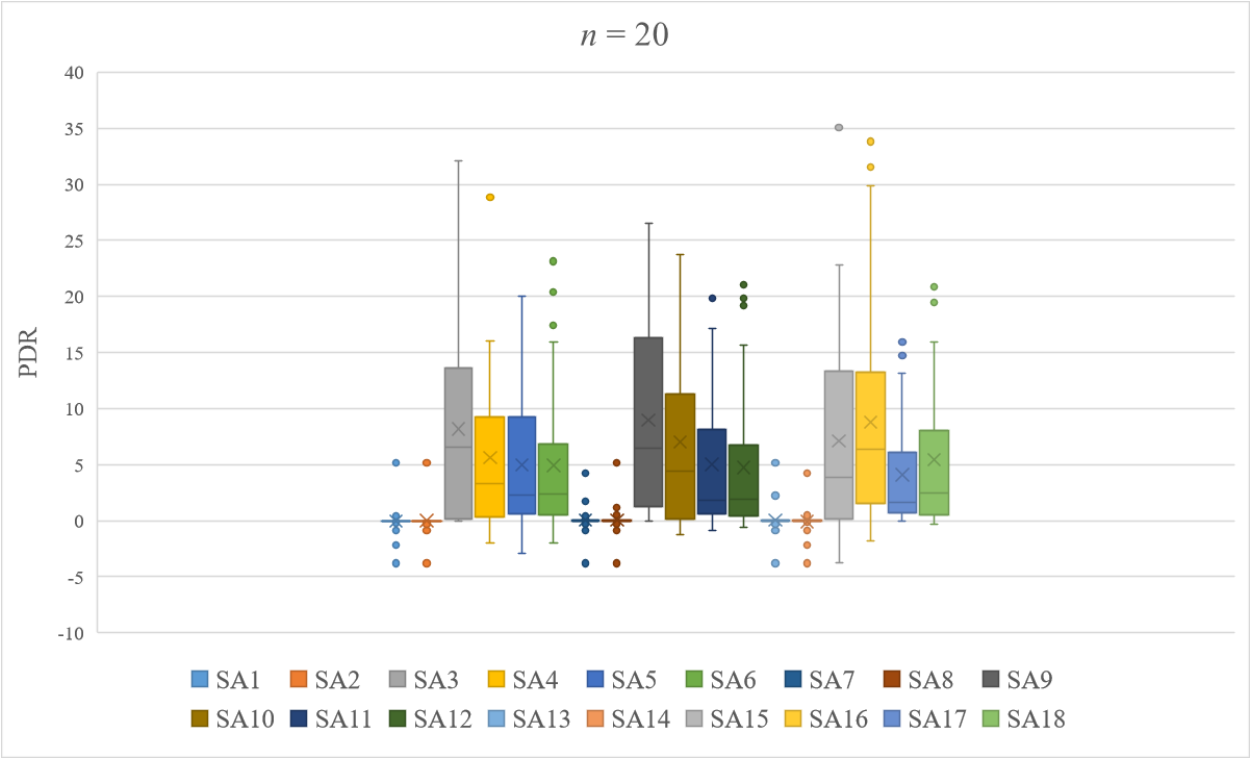
- En las gráficas Gráfica 7-2 y Gráfica 7-4 se puede observar como los algoritmos que utilizan GSC2, es decir SA3, SA4, SA9, SA10, SA15 y SA16 se resuelven en tiempos algo menores que los que usan GS1 y que los tiempos para los algoritmos con GS3, es decir, SA5, SA6, SA11, SA12, SA17 y SA18, son mucho mayores y con mayores desviaciones.



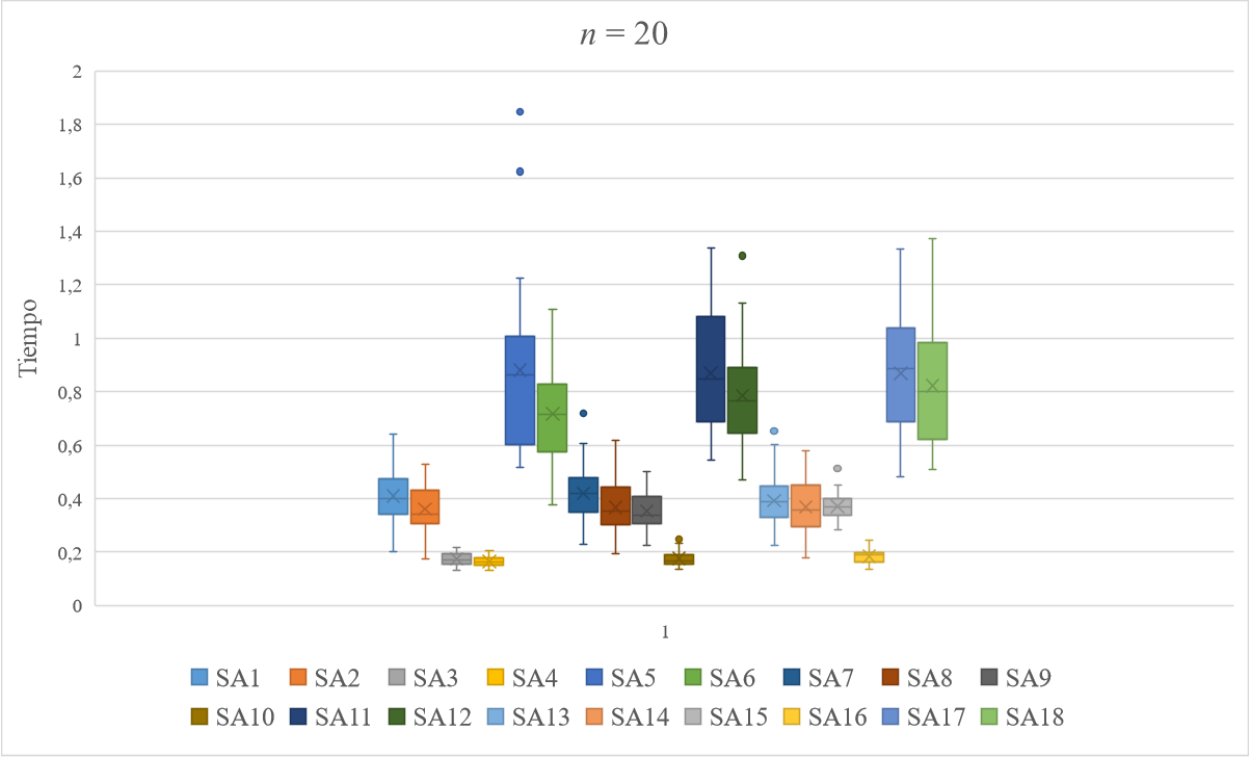
Gráfica 7-1. Diagrama de cajas y bigotes: PDR para $n = 16$.



Gráfica 7-2. Diagrama de cajas y bigotes: tiempos para $n = 16$.



Gráfica 7-3. Diagrama de cajas y bigotes: PDR para $n = 20$.



Gráfica 7-4. Diagrama de cajas y bigotes: tiempos para $n = 20$.

7.2 Análisis sobre batería pequeña

Tras extraer las primeras conclusiones sobre dominancia de algunas heurísticas, se lleva acabo el mismo experimento, pero extendiéndolo a todos los niveles de la batería pequeña y solo sobre los algoritmos que presentan mejores resultados en el experimento del diseño factorial completo; esto es, aquellos que incluyen la generación de secuencia candidata del tipo GSC1: SA1, SA2, SA7, SA8, SA13 y SA14. Analizando la batería pequeña al completo se pueden obtener unos resultados más íntegros, pues se incluye un nivel de trabajo más en el que se tienen las soluciones exactas, $n = 12$, y otro nivel, el cual presenta unas soluciones que son límites superiores, $n = 24$.

Al igual que en el apartado 7.1, se presenta en la Tabla 7-2 un resumen de los resultados obtenidos, expresando los mismos parámetros.

Tabla 7-2. Resumen resultados: análisis sobre batería pequeña.

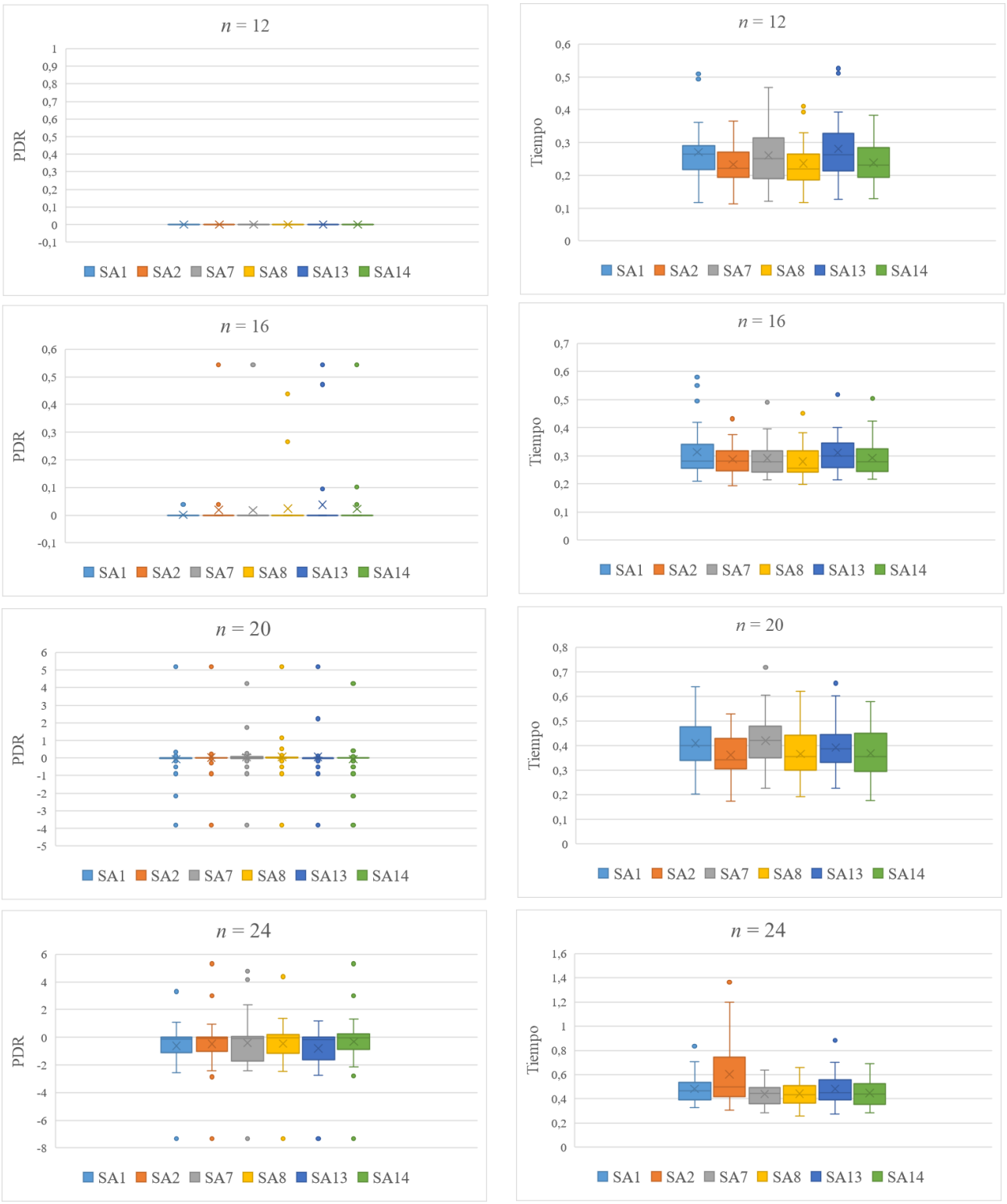
	n = 12		*n* = 16		*n* = 20		*n* = 24		Total	
	PDRM	Tiempo [s]	PDRM	Tiempo [s]	PDRM	Tiempo [s]	PDRM	Tiempo [s]	Media PDRM	Media Tiempo [s]
SA1	0,000	0,271	0,001	0,314	-0,059	0,410	-0,614	0,483	-0,168	0,369
SA2	0,000	0,233	0,019	0,288	0,027	0,362	-0,473	0,602	-0,107	0,371
SA7	0,000	0,261	0,018	0,292	0,046	0,420	-0,406	0,439	-0,085	0,353
SA8	0,000	0,236	0,023	0,281	0,061	0,367	-0,467	0,442	-0,096	0,331
SA13	0,000	0,280	0,037	0,311	0,058	0,369	-0,799	0,483	-0,176	0,361
SA14	0,000	0,239	0,023	0,293	-0,070	0,349	-0,327	0,448	-0,094	0,332

Ademas, en la Gráfica 7-5 se presenta un conjunto de diagramas de cajas y bigotes donde se puede ver y comparar de forma más gráfica las soluciones. Algunas conclusiones que se pueden extraer son:

- Los resultados obtenidos en este experimento son bastante satisfactorios. Los 6 algoritmos estudiados son capaces de obtener el óptimo para el nivel de trabajos $n = 12$ y soluciones muy cercanas a las óptimas para $n = 16$, siendo la mayor desviación porcentual media de 0,037 %. Además, para los niveles $n = 20$ y $n = 24$, se obtienen en casi todos los casos soluciones mejores que los límites superiores encontrados.

- En lo que respecta al tiempo computacional, todos los algoritmos presentan tiempos muy similares para cada nivel de trabajo, siendo estos muy reducidos (véase la Tabla 5-1 para los tiempos de cálculo del modelo matemático en los problemas de la batería pequeña)

Con el objetivo de seguir evaluando el rendimiento de las distintas propuestas, se va a continuar un estudio con la batería grande de problemas. Para ello, se eligen los dos algoritmos que más han destacado de manera general. Aunque para estos casos no hay ningún algortimo que domine sobre otro y las diferencias son muy pequeñas, los algoritmos SA1 y SA13 ofrecen resultados medios algo mejores que el resto. Estos dos algoritmos, además de llevar la técnica GSC1, utilizan la función FP1, difiriendo ambos en la manera de proponer la secuencia inicial.



Gráfica 7-5. Diagrama de cajas y bigotes: análisis sobre batería pequeña.

7.3 Análisis sobre batería grande

Finalmente se extiende el estudio a la batería grande, realizando un último experimento sobre las dos combinaciones que mejores rendimientos han ofrecido en el análisis sobre la batería pequeña. El objetivo es ver si para tamaños grandes hay alguno de los algoritmos que funciona mejor o si por el contrario siguen obteniéndose resultados muy similares.

En este caso, al no haber una referencia de solución óptima, el porcentaje relativo de desviación (PDR) se calcula como

$$\frac{SA_i - SA^*}{SA^*} \cdot 100,$$

donde SA_i es el valor de la función objetivo generado por la heurística i y SA^* es el menor valor de los obtenidos por ambas heurísticas: $SA^* = \min\{SA_i,\ i = 1,\ 13\}$.

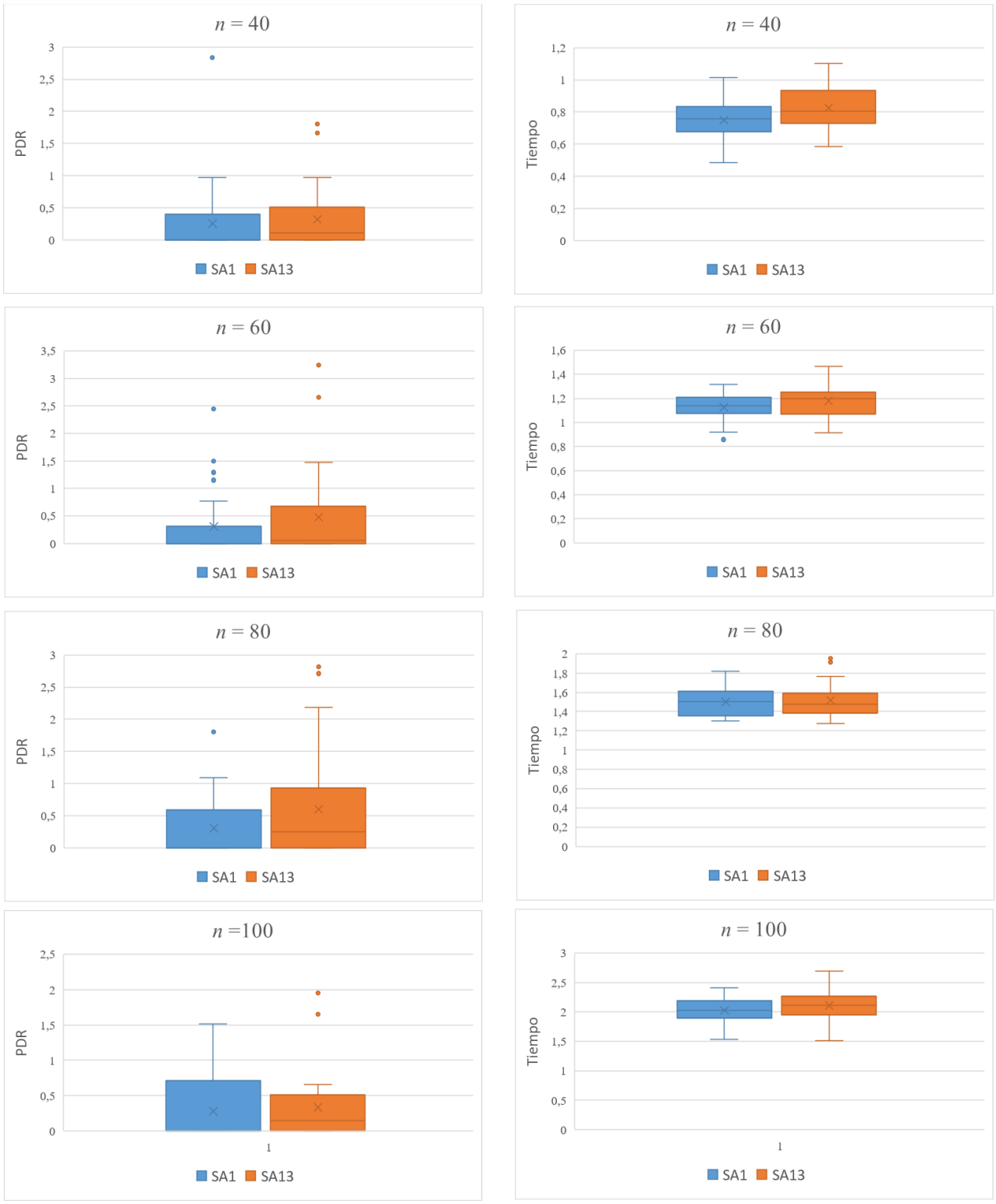
En la Tabla 7-3 se muestra el resumen de los resultados, como en los experimentos anteriores.

Tabla 7-3. Resumen resultados: análisis sobre batería grande.

	n = 40		n = 60		n = 80		n = 100		Total	
	PDRM	Tiempo [s]	PDRM	Tiempo [s]	PDRM	Tiempo [s]	PDRM	Tiempo [s]	Media PDRM	Media Tiempo [s]
SA1	0,250	0,750	0,308	1,125	0,303	1,501	0,281	2,027	0,285	1,351
SA13	0,317	0,827	0,474	1,179	0,600	1,518	0,334	2,109	0,431	1,408

Ademas, la Gráfica 7-6 presenta un conjunto de diagramas de cajas y bigotes donde se puede ver y comparar de forma más gráfica las soluciones. Algunas conclusiones que se pueden extraer son:

- El algoritmo SA1 domina en los 4 casos estudiados para tamaños grandes sobre el algoritmo SA13, tanto en la media de las desviaciones relativas, como en la de los tiempos de cómputo. También se puede apreciar en los diagramas de cajas y bigotes que hay menos dispersión en los resultados obtenidos.
- Aunque las diferencias siguen siendo muy ajustadas, la diferencia porcentual aumenta con resptecto al análisis sobre la batería pequeña y ahora si se puede observar una tendencia dominante de una heurística sobre otra, si bien es cierto que ambas propuestas son muy similares.



Gráfica 7-6. Diagrama de cajas y bigotes: análisis sobre batería grande.

7.4 Análisis de las modificaciones

Tras la realización de los experimentos para seleccionar el algoritmo SA con las diferentes estrategias que proporcionan mejores resultados, se propone incluir algunas modificaciones para intentar afinar aún más el algoritmo desarrollado. Estas mejoras son descritas en el apartado 6.5 de este documento.

Así pues, se implementan estas mejoras sobre el algoritmo SA1, el más eficiente de los propuestos, de manera individual, y se comparan los resultados con los obtenidos sin implementar dichas medidas. El análisis se hace sobre la batería grande, que es donde se pueden obtener diferencias más notables.

En primer lugar, se implementa el reinicio sobre la solución actual cuando no se mejora la mejor solución obtenida hasta el momento en un número de $150 \cdot n$ iteraciones. En la Tabla 7-4 se resumen los resultados obtenidos, de los que se puede extraer que de media se obtienen mejores resutados con el algoritmo mejorado, pero no se puede asegurar una dominancia de la técncia en cada uno de los niveles. En cuanto al tiempo de computación, la mejora del algoritmo incurre en un pequeño aumento de los tiempos de computación.

Tabla 7-4. Resumen resultados: reinicio.

	n = 40		*n* = 60		*n* = 80		*n* = 100		Total	
	PDRM	Tiempo [s]	PDRM	Tiempo [s]	PDRM	Tiempo [s]	PDRM	Tiempo [s]	Media PDRM	Media Tiempo [s]
SA1	0,637	0,750	0,524	1,125	0,272	1,501	0,333	2,027	0,441	1,351
SA1'	0,150	0,786	0,536	1,203	0,460	1,647	0,319	2,379	0,366	1,504

La segunda mejora propuesta es el uso de una función probabilística con enfriamiento adaptativo, con la que en cada iteración la temperatura se corrige según los valores de la solución nueva que se propone y la mejor solución encontrada hasta el momento. Al igual que con la técnica anterior, y como se puede ver en la Tabla 7-5, la media de los resultados es algo mejor que el algoritmo original, pero no tanto como con la primera mejora, y de nuevo no se puede asegurar el dominio de la técncia en todos los niveles de trabajo. Sin embargo, en cuanto a los tiempos de computación, si se observa en este caso una mejora significativa para todos los niveles de trabajo analizados.

Tabla 7-5. Resumen resultados: enfriamiento adaptativo.

	n = 40		*n* = 60		*n* = 80		*n* = 100		Total	
	PDRM	Tiempo [s]	PDRM	Tiempo [s]	PDRM	Tiempo [s]	PDRM	Tiempo [s]	Media PDRM	Media Tiempo [s]
SA1	0,344	0,750	0,548	1,125	0,439	1,501	0,341	2,027	0,418	1,351
SA1"	0,179	0,501	0,535	0,871	0,403	1,209	0,465	1,658	0,396	1,060

Con la referencia de los resultados obtenidos, se propone implementar ambas técnicas a la vez con el objetivo de mejorar los resultados, tanto a nivel de objetivo como en tiempos de cálculo. Como se observa en la Tabla 7-6 la media de las desviaciones ofrece unos resultados mejores que usando las técnicas por separado, manteniendo una pequeña reducción en los tiempos de cáculo.

Tabla 7-6. Resumen resultados: reinicio y enfriamiento adaptativo.

	n = 40		*n* = 60		*n* = 80		*n* = 100		Total	
	PDRM	Tiempo [s]	PDRM	Tiempo [s]	PDRM	Tiempo [s]	PDRM	Tiempo [s]	Media PDRM	Media Tiempo [s]
SA1	0,604	0,750	0,244	1,125	0,273	1,501	0,319	2,027	0,360	1,351
SA1"'	0,089	0,711	0,312	1,100	0,422	1,496	0,283	1,977	0,277	1,321

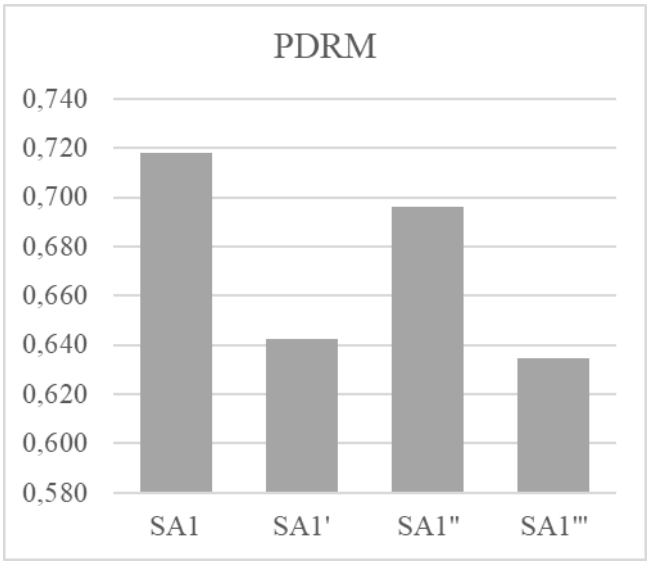
Por último, se compara el desempeño del algoritmo SA1 y las tres nuevas versiones en términos de desviaciones relativas, calculando el porcentaje de desviación relativa (PDR) sobre el mínimo dado por los cuatro algoritmos en cada caso. En la Tabla 7-7 se muestran mas medias de los PDR para cada nivel de trabajo, así como la media total para todos los niveles de la batería grande.

Tabla 7-7. Comparación PDRM de las versiones de SA1.

	n = 40	*n* = 60	*n* = 80	*n* = 100	Total
	PDRM	PDRM	PDRM	PDRM	Media PDRM
SA1	0,820	0,799	0,629	0,624	0,718
SA1'	0,332	0,811	0,818	0,609	0,643
SA1"	0,654	0,787	0,595	0,749	0,696
SA1"'	0,305	0,868	0,779	0,586	0,635

Como se puede observar en la última columna de la Tabla 7-7, así como en la Gráfica 7-7 la versión SA1"' es la que ofrece mejores resultados en general. Así pues, los porcentajes de desviación relativa

total son 0,718 %, 0,643 %, 0,696 % y 0,635 %, respectivamente.



Gráfica 7-7. Comparación PDRM de las versiones de SA1.

8 CONCLUSIONES

En este Trabajo Fin de Máster se estudia un problema de programación de la producción. Concretamente, un problema de entorno de flujo de taller regular con dos máquinas y dos conjuntos de trabajos diferentes, cuyos objetivos son el de minimizar el tiempo total de finalización de un conjunto sin provocar retrasos en los trabajos del otro conjunto.

Para resolver el problema, se plantea un modelo matemático de programación lineal entera y una metaheurística basada en el *Simulated Annealing* ya que, debido a la complejidad computacional del mismo, catalogado como NP-*Hard*, es necesario el desarrollo de métodos aproximados para estudiar el problema en profundidad.

El modelo matemático de programación lineal entera está basado en el modelo SGST de la familia Manne, adaptado a los requerimientos específicos del problema de estudio. Tras la resolución de una batería de instancias de niveles de trabajo pequeños, se puede concluir que el modelo es únicamente válido para resolver problemas que tengan hasta 16 trabajos. A partir de dicho tamaño, los tiempos computacionales son muy elevados, no pudiendo obtener la solución óptima. Sin embargo, para los tamaños más pequeños, el modelo resuelve en un tiempo de cómputo reducido, si bien es cierto que se pueden encontrar casos en los que no hay solución factible, debido a la ajustada restricción que impone el objetivo del segundo conjunto de trabajos.

Para el método aproximado se plantean diferentes versiones del algoritmo *Simulated Annealing*, mezclando diversas técnicas de generación de secuencia inicial, generación de secuencias candidatas y funciones probabilísticas para aceptar soluciones.

Tras el primer estudio, basado en undiseño factorial completo, donde se analizan todas las combinaciones, se concluye que aquellas versiones que tienen implementadas la generación de secuencia candidata basada en un intercambio aleatorio de par de trabajos (*random pairwise interchange*) ofrecen mejores resultados, dominando sobre el resto de las alternativas.

Los experimentos y análisis posteriores sobre la batería pequeña de los algoritmos dominantes indican que los resultados son muy similares, pero hay un par de alternativas que generalmente ofrecen mejores soluciones. Estas son estudiadas en más profundidad con la batería de instancias

grandes, donde se puede dilucidar una versión como la que ofrece mejores resultados. Esta versión incluye una generación inicial de secuencia basada en EDD para el conjunto B y SPT en máquina 1 del conjunto A, una generación de secuencia candidata basada en intercambio aleatorio de par de trabajos y una función probabilística con enfriamiento multiplicativo lineal.

Se puede concluir que el aspecto más importante del algoritmo es la función de probabilidad y su esquema de enfriamiento, el cual debe ser expuesto a un exhaustivo estudio y calibración, ya que, si no, los resultados pueden variar en grandes cantidades. Posteriormente, y relativo a la importancia, se encuentra la generación de secuencia candidata, pues se ha demostrado que algunas de ellas no son tan directas en la evolución hacia la solución óptima. Por último, la generación de la secuencia inicial, cuya influencia en los resultados finales es mucho menor.

Finalmente, y en relación con las modificaciones para mejorar el algoritmo, se concluye que las dos propuestas son buenas técnicas que merecen un estudio en más profundidad. Queda demostrado que tanto la implementación de una técnica de reinicio que reconduzca al algoritmo, como el uso de una función adaptativa que varíe según las soluciones analizadas, implican típicamente la obtención de mejores resultados.

REFERENCIAS

Addou, E. H., Serghini, A. & Mermri, E. B. (2018). Simulated annealing algorithm with restart strategy for optimizing k-minimum spanning tree problems. *Business Intelligence & Big Data, 14ème Edition de la conference EDA,* 321–330. Tanger, Maroc.

Agnetis, A., Mirchandani, P. B., Pacciarelli, D., & Pacifici, A. (2004). Scheduling problems with two competing agents. *Operations Research*, *52*(2), 229–242. https://doi.org/10.1287/opre.1030.0092

Ahmadi-Darani, M., Moslehi, G., & Reisi-Nafchi, M. (2018). A two-agent scheduling problem in a two-machine flowshop. *International Journal of Industrial Engineering Computations 9*, 289–306. https://doi.org/10.5267/j.ijiec.2017.8.005

Aliaga, A. (2015). *Análisis de un Problema de Secuenciación en Flow-shop de Permutación con Dos Conjuntos de Trabajos y Objetivo Makespan* (Proyecto Fin de Carrera Ingeniero Aeronáutico). Tutor: Paz Pérez González. Universidad de Sevilla. Escuela Técnica Superior de Ingeniería. Recuperado de http://bibing.us.es/proyectos/.

Baker, K. R., & Trietsch, D. (2018). *Principles of Sequencing and Scheduling, Second Edition*. John Wiley&Sons. https://doi.org/10.1002/9781119262602

Duan, J., Yang, Y., Gao, K., Li, J., & Pan, Q. (2013). A Speed-up Method for calculating total flowtime in permutation flow shop scheduling problem. *25th Chinese Control and Decision Conference (CCDC),* 2755–2758. Guiyang, China. https://doi.org/10.1109/CCDC.2013.6561411

Fan, B. Q., & Cheng, T. C. E. (2016). Two-agent scheduling in a flowshop. *European Journal of Operational Research*, *252*(2), 376–384. https://doi.org/10.1016/j.ejor.2016.01.009

Fattahi, P., Saidi, M. & Jolai, F. (2007). Mathematical modeling and heuristic approaches to flexible job. *J Intell Manuf 18:*331–342. .https://doi.org/10.1007/s10845-007-0026-8

Framinan, J. M., Leisten, R., & Ruiz García, R. (2014). Manufacturing Scheduling Systems. In *Manufacturing Scheduling Systems*. https://doi.org/10.1007/978-1-4471-6272-8

Gallo, C., & Capozzi, V. (2019). A Simulated Annealing Algorithm for Scheduling Problems. *Open Journal of Applied Mathematics and Physics*. https://doi.org/10.4236/jamp.2019.simann

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. H. G. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. In *Annals of Discrete Mathematics* (Vol. 5, Issue C, pp. 287–326). https://doi.org/10.1016/S0167-5060(08)70356-X

Hoon, Y., & Pinedo, M. (1997). Theory and Methodology. Scheduling jobs on parallel machines with sequence-dependent setup time*s*. *European Journal of Operations Research 100*, 464-474.

Jeong, B. J., Kim, Y. D., & Shim, S. O. (2020). Algorithms for a two-machine flowshop problem with jobs of two classes. *International Transactions in Operational Research*, *27*(6), 3123–3143. https://doi.org/10.1111/itor.12530

Johns, S., & Brucker, P. (1996). Scheduling Algorithms. In *The Journal of the Operational Research Society* (Vol. 47, Issue 8). https://doi.org/10.2307/3010416

Johnson, S. M. (1954). Optimal Two- and Three-stage production schedules with setup times included. *Naval Research Logistic Quaterly,* John Wiley & Sons, vol.1(1), 61-68.

Kirkpatrick, S., Gelatt, C. D. & Vecchi, P. M. (1983). Optimization by Simulated Annealing. Science, Vol. 220, No. 4598, pp. 671-680.

Lee, W. C., Chen, S. K., Chen, C. W., & Wu, C. C. (2011). A two-machine flowshop problem with two agents. *Computers and Operations Research*, *38*(1), 98–104. https://doi.org/10.1016/j.cor.2010.04.002

Lee, W., Chen, S., & Wu, C. (2010). Expert Systems with Applications Branch-and-bound and simulated annealing algorithms for a two-agent scheduling problem. *Expert Systems With Applications*, *37*(9), 6594–6601. https://doi.org/10.1016/j.eswa.2010.02.125

Lei, D. (2015). Computers & Industrial Engineering Variable neighborhood search for two-agent flow shop scheduling. *COMPUTERS & INDUSTRIAL ENGINEERING*, *80*, 125–131. https://doi.org/10.1016/j.cie.2014.11.024

Li, X., Wang, Q., & Wu, C. (2009). Efficient composite heuristics for total flowtime minimization in

permutation flow shops. *Omega 37*, 155–164. https://doi.org/10.1016/j.omega.2006.11.003

Locatelli, M. (2000). Convergence of a Simulated Annealing Algorithm for Continuous Global Optimization. *Journal of Global Optimization 18*:219–234.

Luke, S. (2011). *Essentials of metaheuristics.* Lulu, second edition.

Luo, W., Chen, L., & Zhang, G. (2012). Approximation schemes for two-machine flow shop scheduling with two agents. *J Comb Optim 24:*229–239. https://doi.org/10.1007/s10878-011-9378-2

Normasari, N. M. E., Yu, V. F., Bachtiyar, C. y Sukoyo (2019). A Simulated Annealing Heuristic for the Capacitated Green Vehicle Routing Problem. *Hindawi, Mathematical Problems in Engineering*.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., & Teller, A. H. (1953). Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics 21, 1087*.

Mor, B., & Mosheiov, G. (2014). Polynomial time solutions for scheduling problems on a proportionate flowshop with two competing agents. *Journal of the Operational Research Society*, *65*(1), 151–157. https://doi.org/10.1057/jors.2013.9

Ogbu, F. A. & Smith, D. K. (1990). The application of the simulated annealing algorithm to the solution of the n / m / C_{max} flowshop problem. *Computers Opns. Res. 17*(3), 243–253.

Karasakal, E. K., & Köksalan, M. K. (2000). A Simulated Annealing Approach to Bicriteria Scheduling Problems on a Single Machine. *Journal of Heuristics, 6,* 311–327.

Perez-Gonzalez, P., & Framinan, J. M. (2010). Setting a common due date in a constrained flowshop: A variable neighbourhood search approach. *Computers and Operations Research*, *37*(10), 1740–1748. https://doi.org/10.1016/j.cor.2010.01.002

Perez-Gonzalez, P., & Framinan, J. M. (2014). A common framework and taxonomy for multicriteria scheduling problems with interfering and competing jobs: Multi-agent scheduling problems. *European Journal of Operational Research*, *235*(1), 1–16. https://doi.org/10.1016/j.ejor.2013.09.017

Pinedo, M. L. (2008). Scheduling: Theory, algorithms, and systems. In *Scheduling: Theory, Algorithms, and Systems*. https://doi.org/10.1007/978-0-387-78935-4

Ponce-Ortega, J. M., & Hernández-Pérez, L. G. (2019). Optimization of Process Flowsheets through Metaheuristic Techniques. In *Optimization of Process Flowsheets through Metaheuristic Techniques*. https://doi.org/10.1007/978-3-319-91722-1

Potts, C. N. & Osman, I. H. (1989). Simulated Annealing for Permutation Flow-Shop Scheduling. *Omega 17*(6), 551–557.

Shi, Z., Fernando, W. A. C., & Kondoz, A. (2012). Simulated Annealing for Fast Motion Estimation Algorithm in H.264/AVC. *Simulated Annealing - Single and Multiple Objective Problems*. https://doi.org/10.5772/50974

Shiau, Y., Tsai, M., Lee, W., & Cheng, T. C. E. (2015). Computers & Industrial Engineering Two-agent two-machine flowshop scheduling with learning effects to minimize the total completion time. *Computers & Industrial Engineering*, *87*, 580–589. https://doi.org/10.1016/j.cie.2015.05.032

Smith, J. C., & Kenneth R. Baker, J. C. S. (2003). A Multiple-Criterion Model for Machine Scheduling. *Journal of Scheduling*, *6*, 7–16.

Stafford, E. F., Tseng, F. T., & Gupta, J. N. D. (2005). Comparative evaluation of MILP flowshop models. *Journal of the Operational Research Society*, *56*(1), 88–101. https://doi.org/10.1057/palgrave.jors.2601805

Lundy, M., & Mess, A. (1986). *Convergence of an anneling algorithm.* Math. Prog., 34, 111–124.

Ben-Arieh, D., Maimon, O., (1997). Annealing method for PCB assembly scheduling on two sequential machines. *International Journal of Computer Integrated Manufacturing, 5:6,* 361-367. https://doi.org/10.1080/09511929208944543

Dowsland, K. A. (1995). Simulated Annealing. In *Modern Heuristic Techniques for Combinatorial Problems* (ed. Reeves, C.R.), McGraw-Hill.

Yazgan, H. R. & Akkaya, G. (2015). Minimization of Total Flow Time with using Simulated Annealing Algorithm in Flow Line Manufacturing. *2nd International Symposium on Intelligent Manufacturing Systems.* Sakarya, Turkey: Sakarya University.

Yin, Y., Cheng, T. C. E., Wang, D., & Wu, C. (2017). Two-agent flowshop scheduling to maximize the weighted number of just-in-time jobs. *Journal of Scheduling*, *20*(4), 313–335.

https://doi.org/10.1007/s10951-017-0511-7

https://www.gurobi.com/documentation/8.0/refman/matlab_the_model_argument.html (última fecha de visita: 31/07/2020)

Liang, F. (2020). Optimization Techniques – Simulated Annealing. Recuperado de https://towardsdatascience.com/optimization-techniques-simulated-annealing-d6a4785a1de7

ANEXO A. MODELO MATEMÁTICO

En el Anexo A se recoge la programación del modelo matemático y los resultados correspondientes a las soluciones exactas.

En primer lugar, se encuentran los modelos implementados en Excel y GAMS usados para la validación del modelo en MATLAB, cuyo código se detalla seguidamente, así como el código para la generación de datos mediante la batería de instancias en MATLAB.

En segundo lugar, se plasman todos los resultados obtenidos, sin procesar, de la batería de problemas pequeña, provenientes de la resolución exacta de los problemas mediante el modelo matemático de programación lineal.

Anexo A.1. Modelo matemático de programación lineal

Anexo A.1.1 Modelo matemático en Excel

$$PF2|\sum_{j\in B} T_j^B = 0|\sum_{j\in A} C_{2j}^A$$

*Datos ejemplo sencillo

n_A = n_B = 2 --> n = 4
14 variables
34 restricciones

Datos														
p_1j	7	8	5	1										
p_2j	2	9	2	8										
d_j	17	25	10	15										
M	4,20E+01													

Objetivo (minimizar)	47													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Variables	C_11	C_12	C_13	C_14	C_21	C_22	C_23	C_24	delta_12	delta_13	delta_14	delta_23	delta_24	delta_34
Valores	13	21	5	6	17	30	7	15	1	0	0	0	0	1

Restricciones																LHS		RHS	
1	(R1)															7	<=	10	
2																15	<=	15	
3	(R2)	1														13	>=	7	
4			1													21	>=	8	
5				1												5	>=	5	
6					1											6	>=	1	
7	(R3)	-1				1										4	>=	2	
8			-1				1									9	>=	9	
9				-1				1								2	>=	2	
10					-1				1							9	>=	8	
11	(R4)	1	-1							42						34	>=	7	
12		1		-1							42					8	>=	7	
13		1			-1							42				7	>=	7	
14			1	-1									42			16	>=	8	
15			1		-1									42		15	>=	8	
16				1	-1										42	41	>=	5	
17						1	-1		42							29	>=	2	
18						1		-1		42						10	>=	2	
19						1			-1		42					2	>=	2	
20							1	-1				42				23	>=	9	
21							1		-1				42			15	>=	9	
22							1		-1						42	34	>=	2	
23	(R5)	1	-1							42						34	<=	34	
24		1		-1							42					8	<=	37	
25		1			-1							42				7	<=	41	
26			1	-1									42			16	<=	37	
27			1		-1									42		15	<=	41	
28				1	-1										42	41	<=	41	
29						1	-1		42							29	<=	33	
30						1		-1		42						10	<=	40	
31						1			-1		42					2	<=	34	
32							1	-1				42				23	<=	40	
33							1		-1				42			15	<=	34	
34								1	-1						42	34	<=	34	

Anexo A.1.2 Modelo matemático en GAMS

```
*Ejemplo modelo matemático SGST(Manne)
*-------------------------------------
option limrow = 12

Sets
         i máquinas  / 1, 2 /
         j trabajos  / 1, 2, 3, 4, 5, 6, 7, 8 /
         j_A(j) subset trabajos conjunto A    / 1, 2, 3, 4 /
         j_B(j) subset trabajos conjunto B    / 5, 6, 7, 8 /
;

alias(j,k)

Parameters

       d(j)  Fecha de entrega del trabajo j
         / 1 327
           2 258
           3 250
           4 153
           5 323
           6 268
           7 203
           8 242 /


      table  p(i,j)    Tiempo de proceso del trabajo j en la
máquina i

                    1    2   3   4   5   6   7   8
               1    11   97  1   78  82  87  9   40
               2    26   81  44  92  19  27  15  14
;

Variables
```

```
Z  Función a minimizar (objetivo conjunto A)
C(i,j)  Tiempo de finalización del trabajo j en la máquina i
delta(j,k)  =1 si el trabajo j es proceso antes que k
;

Binary Variable delta ;

Positive Variable C  ;

Equations

Objective

Constraint1(j_B)
Constraint2(j)
Constraint3(j)
Constraint4(i,j,k)
Constraint5(i,j,k)
;

Objective..           Z =e= sum((j_A), C('2',j_A));

Constraint1(j_B)..   C('2',j_B) =l= d(j_B);
Constraint2(j)..     C('1',j) =g= p('1',j);
Constraint3(j)..     C('2',j)-C('1',j) =g= p('2',j) ;
Constraint4(i,j,k)$(ord(j)   <   ord(k))..      C(i,j)-C(i,k)   +
723*delta(j,k) =g= p(i,j);
Constraint5(i,j,k)$(ord(j)   <   ord(k))..      C(i,j)-C(i,k)   +
723*delta(j,k) =l= 723 - p(i,k);

Model Manne_SGST / all /;

Solve Manne_SGST mip minimizing Z;

Display Z.l, C.l, delta.l ;
```

Anexo A.1.3. Batería de datos en MATLAB

```
%% Batería de DATOS: tiempos de proceso (p_ij) y fechas de
entrega (d_j)
% clear all
% for n_A = 2:2:16
n_A = 50; %
%n_B = n_A;
n = n_A*2;
    for z = 1:30 % 30 instancias aleatorias para cada tamaño
    %z = 10; %
    p = randi(100, 1, 2*n); % [p_1jA, p_1jB, p_2jA, p_2jB]
    p_sum = sum(p(n+1:2*n))+min(p(1:n));
    T = 0.25;
    R = 0.75;
    d = randi([floor(p_sum*(1-T-R/2)) floor(p_sum*(1-T+R/2))],
1, n); % [d_jA, d_jB]
    datos = [p d];
    % Los datos se almacenan en archivos de texto diferentes
    archivo = sprintf('Instancia_%d_%d', n_A, z);
    fid = fopen(archivo, 'w');
    fprintf(fid, '%d\n', datos);
    fclose(fid);
    end
% end
```

Anexo A.1.4. Modelo matemático en MATLAB

```
% clear all
% clear model
% clear params
% clc

%% MODELO: Manne SGST (adaptado: min total completion time de A
sin trabajos retrasados en B)
% VARIABLES: vector de Completion times C_ij, vector binario
delta_jk
% C_1j: vector de n componentes
% C_2j: vector de n componentes
% delta_jk: vector de n*(n-1)/2 componentes que determinan el
orden de la secuencia
% delta_jk = 1 si "j" va antes que "k", delta_jk = 0 si "k" va
antes que "j"
% x = [C_1j, C_2j, delta_jk]: vector de 2*n+(n-1)*n/2
componentes
%   compenentes desde 1 hasta n_A --> C_1j del Conjunto A
%   componentes desde n_A+1 hasta n --> C_1j del Conjunto B
%   compenentes desde n+1 hasta n+n_A --> C_2j del Conjunto A
%   componentes desde n+n_A+1 hasta 2*n --> C_2j del Conjunto B
```

```
fid = fopen('resultados.txt', 'w');
fprintf(fid, '%s  %s  %s  %s\n\n', 'Instancia', 'SUM_C_2j(A)',
'T_opt', 'Estado');

for n_A = 4:2:16 % bucle para resolver todos los tamaños
%n_A = 4; % activar para instancia concreta
n_B = n_A;
n = n_A+n_B;
for z = 1:30 % diferentes instancias para cada tamaño
%z = 10; % activar para instancia concreta
% Lee los datos de un archivo
clear datos

archivo = sprintf('Instancia_%d_%d', n_A, z);
fid2 = fopen(archivo, 'r');
datos = fscanf(fid2, '%f');
datos = transpose(datos);
fclose(fid2);

p_ij = datos(1:2*n); % [p_1jA, p_1jB, p_2jA, p_2jB]
% d_j = [d_jA, d_jB]
d_Aj = datos(2*n+1:2*n+n_A);
d_Bj = datos(2*n+n_A+1:3*n);

M = sum(p_ij);

clear A; clear model.A
clear sense; clear model.sense
clear rhs; clear model.rhs
clear vtype; clear model.vtype
clear obj; clear model.obj
clear result
clear model

%% COEFICIENTES de las restricciones, Matriz [A], vectores [rhs]
y [sense]
% A(x,y): filas x -> restricciones: n_B+2*n+2*n*(n-1) / columnas
y -> variables: 2*n+n*(n-1)/2
A = zeros(n_B+2*n+2*n*(n-1), 2*n+n*(n-1)/2); % [A] almacena
coeficientes
% [sense]: n_B+2*n+2*n*(n-1) restricciones
sense = blanks(n_B+2*n+2*n*(n-1)); % [sense] almacena sentido de
restricciones
% [rhs]: n_B+2*n+2*n*(n-1) restricciones
rhs = zeros(1, n_B+2*n+2*n*(n-1)); % [rhs] almacena el valor a
la derecha de cada restricción

%% Restricción 1: Condición Tardanza total de B = 0 (n_B
restricciones)
% Para (n_A+1)<j<n: C_2j >= d_Bj
```

```
ini = 1;
fin = n_B;

cont = n+n_A+1; % contador que lleva hasta la variable
C_(2,n_A+1)
for x = ini:fin
A(x,cont) = 1;
sense(x) = '<';
cont = cont+1;
end
cont = 1;
for x = ini:fin
rhs(x) = d_Bj(cont);
cont = cont+1;
end

%% Restricción 2: Trabajos en la primera máquina (n
restricciones)
% Para 1<j<n: C_1j >= p_1j
ini = n_B+1;
fin = n_B+n;

cont = 1;
for x = ini:fin
A(x, cont) = 1;
sense(x) = '>';
rhs(x) = p_ij(cont);
cont = cont+1;
end

%% Restricción 3: Un trabajo que pasa de una máquina a la
siguiente (n restricciones)
% Para 1<j<n: C_2j-C_1j >= p_2j
ini = n_B+n+1;
fin = n_B+2*n;

cont = 1;
for x = ini:fin
A(x, cont+n) = 1; % máquina 2
A(x, cont) = -1; % máquina 1
sense(x) = '>';
rhs(x)= p_ij(cont+n);
cont = cont+1;
end

%% Restricción 4: Secuencia de trabajos en una máquina (k antes
que j)
% Para 1<j<k<n; C_ij-C_ik+M*delta_jk >= p_ij; n*(n-1)
restricciones

% Bucle para las variables C_ij/C_ik y el vector rhs
ini = n_B+2*n+1;
```

```
fin = n_B+2*n+n*(n-1);

cont_j = 1; cont_k = 1; r = 1;
for x = ini:fin
    if cont_k < r*n+1
        if cont_k ~= cont_j
        A(x, cont_j) = 1;
        A(x, cont_k) = -1;
        rhs(x) = p_ij(cont_j);
        cont_k = cont_k+1;
        else
        cont_k = cont_k+1;
        A(x, cont_j) = 1;
        A(x, cont_k) = -1;
        rhs(x) = p_ij(cont_j);
        cont_k = cont_k+1;
        end
    else
    cont_j = cont_j+1;
        if cont_j < r*n % primera máquina: C_1j-C_1k
        cont_k = cont_j+1;
        A(x, cont_j) = 1;
        A(x, cont_k) = -1;
        rhs(x) = p_ij(cont_j);
        cont_k = cont_k+1;
        else
        r = 2; % se pasa a la siguiente máquina: C_2j-C_2k
        cont_j = cont_j+1;
        cont_k = cont_j+1;
        A(x, cont_j) = 1;
        A(x, cont_k) = -1;
        rhs(x) = p_ij(cont_j);
        cont_k = cont_k+1;
        end
    end
end

% Bucle para las variables delta
cont_delta = 2*n+1;
for x = ini:fin
    if cont_delta <= 2*n+(n-1)*n/2
    A(x, cont_delta) = M;
    cont_delta = cont_delta+1;
    else
    cont_delta = 2*n+1;
    A(x, cont_delta) = M;
    cont_delta = cont_delta+1;
    end
end

% Bucle para vector sense
```

```
for x = ini:fin
sense(x) = '>';
end

%% Restricción 5: Secuencia de trabajos en una máquina  (j antes
que k)
%Para 1<j<k<n; C_ij-C_ik)+M*delta_jk <= M-p_rk; n*(n-1)
restricciones

% Bucle para las variables C_ij(C_ik) y el vector rhs
ini = n_B+2*n+n*(n-1)+1;
fin = n_B+2*n+2*n*(n-1);

cont_j = 1; cont_k = 1; r = 1;
for x = ini:fin
    if cont_k < r*n+1
        if cont_k ~= cont_j
        A(x, cont_j) = 1;
        A(x, cont_k) = -1;
        rhs(x) = M-p_ij(cont_k);
        cont_k = cont_k+1;
        else
        cont_k = cont_k+1;
        A(x, cont_j) = 1;
        A(x, cont_k) = -1;
        rhs(x) = M-p_ij(cont_k);
        cont_k = cont_k+1;
        end
    else
    cont_j = cont_j+1;
        if cont_j < r*n % primera máquina: C_1j-C_1k
        cont_k = cont_j+1;
        A(x, cont_j) = 1;
        A(x, cont_k) = -1;
        rhs(x)= M-p_ij(cont_k);
        cont_k = cont_k+1;
        else
        r = 2; % se pasa a la siguiente máquina: C_2j-C_2k
        cont_j =cont_j+1;
        cont_k = cont_j+1;
        A(x, cont_j) = 1;
        A(x, cont_k) = -1;
        rhs(x) = M-p_ij(cont_k);
        cont_k = cont_k+1;
        end
    end
end

% Bucle para las variables delta
cont_delta = 2*n+1;
for x = ini:fin
    if cont_delta <= 2*n+(n-1)*n/2
```

```
    A(x, cont_delta) = M;
    cont_delta = cont_delta+1;
    else
    cont_delta = 2*n+1;
    A(x, cont_delta) = M;
    cont_delta = cont_delta+1;
    end
end

% Bucle para vector sense
for x = ini:fin
sense(x) = '<';
end

%% Función Objetivo
obj = zeros(1, 2*n+(n-1)*n/2); % coeficientes para cada variable
en la función objetivo

ini = 1;
fin = n_A;

cont = n+1;
for x = ini:fin
obj(cont) = 1;
cont = cont+1;
end

%% Optimización en GUROBI
vtype = blanks(2*n+(n-1)*n/2);
for cont = 1:2*n
vtype(cont) = 'I';
end
for cont = 2*n+1:2*n+(n-1)*n/2
vtype(cont) = 'B';
end

model.vtype = vtype; % tipo de variables
model.A = sparse(A); % matriz de coeficientes de las
restricciones
model.rhs = rhs; % valor a la derecha de cada restricción
model.obj = obj; % función objetivo
model.sense = sense; % sentido de cada restricción
params.timeLimit = 900; % límite para el tiempo (s) empleado en
la optimización
result = gurobi(model,params);

% Mostrar resultados
T_opt = result.runtime; % tiempo empleado en hallar el óptimo
estado = result.status;

% delta = result.x(2*n+1:end);
```

```
% C = result.x(1:2*n);
% variables = result.x;
% lhs = A*result.x;

if isequal(estado, 'OPTIMAL')|| isequal(estado, 'TIME_LIMIT')
%SUM_C_2j_A = sum(result.x(n+1:n+n_A))
SUM_C_2j_A = result.objval; % total completion time del conjunto
A
fprintf(fid, '%d_%d  %d  %6.3f  %s\n', n_A, z, SUM_C_2j_A,
T_opt, estado);
else
fprintf(fid, '%d_%d  %s  %s  %s\n', n_A, z, 'N/A', 'N/A',
estado);
end

end
end
fclose(fid);
```

Anexo A.2. Resultados modelo matemático

Instancia	SUM_C_2j(A)	T_opt	Estado	Instancia	SUM_C_2j(A)	T_opt	Estado
2_1	539	0,047	OPTIMAL	4_1	1002	0,156	OPTIMAL
2_2	449	0,132	OPTIMAL	4_2	1050	0,126	OPTIMAL
2_3	N/A	N/A	INFEASIBLE	4_3	713	0,151	OPTIMAL
2_4	N/A	N/A	INFEASIBLE	4_4	1010	0,141	OPTIMAL
2_5	N/A	N/A	INFEASIBLE	4_5	971	0,194	OPTIMAL
2_6	435	0,11	OPTIMAL	4_6	816	0,126	OPTIMAL
2_7	N/A	N/A	INFEASIBLE	4_7	703	0,094	OPTIMAL
2_8	N/A	N/A	INFEASIBLE	4_8	954	0,171	OPTIMAL
2_9	302	0,062	OPTIMAL	4_9	1193	0,151	OPTIMAL
2_10	459	0,047	OPTIMAL	4_10	600	0,23	OPTIMAL
2_11	539	0,09	OPTIMAL	4_11	816	0,15	OPTIMAL
2_12	449	0,109	OPTIMAL	4_12	703	0,067	OPTIMAL
2_13	N/A	N/A	INFEASIBLE	4_13	954	0,125	OPTIMAL
2_14	N/A	N/A	INFEASIBLE	4_14	1193	0,172	OPTIMAL
2_15	N/A	N/A	INFEASIBLE	4_15	N/A	N/A	INFEASIBLE
2_16	435	0,094	OPTIMAL	4_16	1234	0,109	OPTIMAL
2_17	N/A	N/A	INFEASIBLE	4_17	N/A	N/A	INFEASIBLE
2_18	N/A	N/A	INFEASIBLE	4_18	1334	0,157	OPTIMAL
2_19	302	0,078	OPTIMAL	4_19	1258	0,172	OPTIMAL
2_20	459	0,031	OPTIMAL	4_20	831	0,156	OPTIMAL
2_21	367	0,1	OPTIMAL	4_21	1123	0,187	OPTIMAL
2_22	327	0,078	OPTIMAL	4_22	953	0,156	OPTIMAL
2_23	209	0,055	OPTIMAL	4_23	769	0,156	OPTIMAL
2_24	308	0,096	OPTIMAL	4_24	870	0,125	OPTIMAL
2_25	215	0,062	OPTIMAL	4_25	1616	0,109	OPTIMAL
2_26	230	0,078	OPTIMAL	4_26	1949	0,204	OPTIMAL
2_27	405	0,109	OPTIMAL	4_27	N/A	N/A	INFEASIBLE
2_28	488	0,078	OPTIMAL	4_28	1008	0,109	OPTIMAL
2_29	282	0,078	OPTIMAL	4_29	1024	0,156	OPTIMAL
2_30	313	0,047	OPTIMAL	4_30	972	0,078	OPTIMAL

Instancia	SUM_C_2j(A)	T_opt	Estado	Instancia	SUM_C_2j(A)	T_opt	Estado
6_1	1946	0,327	OPTIMAL	8_1	3150	103,792	OPTIMAL
6_2	1210	0,471	OPTIMAL	8_2	1758	1,289	OPTIMAL
6_3	2077	1,728	OPTIMAL	8_3	3108	160,768	OPTIMAL
6_4	1587	0,814	OPTIMAL	8_4	2211	17,728	OPTIMAL
6_5	2552	2,202	OPTIMAL	8_5	2767	316,508	OPTIMAL
6_6	1062	0,203	OPTIMAL	8_6	2715	55,336	OPTIMAL
6_7	2310	1,378	OPTIMAL	8_7	2331	5,15	OPTIMAL
6_8	2684	1,069	OPTIMAL	8_8	2760	32,982	OPTIMAL
6_9	828	0,234	OPTIMAL	8_9	3424	307,711	OPTIMAL
6_10	1893	1,078	OPTIMAL	8_10	3371	146,758	OPTIMAL
6_11	1826	0,426	OPTIMAL	8_11	1854	18,674	OPTIMAL
6_12	1825	0,603	OPTIMAL	8_12	3044	690,21	OPTIMAL
6_13	2054	0,812	OPTIMAL	8_13	3396	391,693	OPTIMAL
6_14	1416	0,5	OPTIMAL	8_14	2965	15,419	OPTIMAL
6_15	N/A	N/A	INFEASIBLE	8_15	2578	73,443	OPTIMAL
6_16	2204	0,312	OPTIMAL	8_16	3130	14,239	OPTIMAL
6_17	2415	1,241	OPTIMAL	8_17	2846	54,322	OPTIMAL
6_18	2269	1,834	OPTIMAL	8_18	3151	114,844	OPTIMAL
6_19	N/A	N/A	INFEASIBLE	8_19	1694	3,858	OPTIMAL
6_20	2525	1,071	OPTIMAL	8_20	1775	1,311	OPTIMAL
6_21	3225	1,321	OPTIMAL	8_21	2114	12,309	OPTIMAL
6_22	1164	0,234	OPTIMAL	8_22	2943	44,367	OPTIMAL
6_23	2394	1,578	OPTIMAL	8_23	2723	900,062	TIME_LIMIT
6_24	2825	0,964	OPTIMAL	8_24	3315	64,06	OPTIMAL
6_25	1417	0,77	OPTIMAL	8_25	4111	27,215	OPTIMAL
6_26	1402	0,293	OPTIMAL	8_26	2103	7,195	OPTIMAL
6_27	2060	8,824	OPTIMAL	8_27	3094	18,053	OPTIMAL
6_28	2411	2,704	OPTIMAL	8_28	3433	45,608	OPTIMAL
6_29	1196	1,197	OPTIMAL	8_29	2132	5,626	OPTIMAL
6_30	2317	0,531	OPTIMAL	8_30	2024	24,95	OPTIMAL

Instancia	SUM_C_2j(A)	T_opt	Estado	Instancia	SUM_C_2j(A)	T_opt	Estado
10_1	4481	900,044	TIME_LIMIT	12_1	11243	900,526	TIME_LIMIT
10_2	3260	652,961	OPTIMAL	12_2	6485	900,078	TIME_LIMIT
10_3	4791	900,075	TIME_LIMIT	12_3	6475	900,321	TIME_LIMIT
10_4	4636	900,073	TIME_LIMIT	12_4	4572	900,432	TIME_LIMIT
10_5	3553	900,067	TIME_LIMIT	12_5	11077	900,472	TIME_LIMIT
10_6	2524	321,922	OPTIMAL	12_6	4378	900,074	TIME_LIMIT
10_7	3125	900,091	TIME_LIMIT	12_7	6771	900,071	TIME_LIMIT
10_8	3471	915,157	TIME_LIMIT	12_8	5065	900,078	TIME_LIMIT
10_9	3893	900,089	TIME_LIMIT	12_9	6272	900,198	TIME_LIMIT
10_10	4888	900,054	TIME_LIMIT	12_10	5623	900,168	TIME_LIMIT
10_11	4580	900,069	TIME_LIMIT	12_11	5686	900,124	TIME_LIMIT
10_12	4278	900,075	TIME_LIMIT	12_12	5907	900,112	TIME_LIMIT
10_13	4955	900,053	TIME_LIMIT	12_13	6130	900,055	TIME_LIMIT
10_14	3665	900,047	TIME_LIMIT	12_14	6997	900,126	TIME_LIMIT
10_15	3727	759,395	OPTIMAL	12_15	4655	900,047	TIME_LIMIT
10_16	3467	900,048	TIME_LIMIT	12_16	6916	900,09	TIME_LIMIT
10_17	3816	900,059	TIME_LIMIT	12_17	5503	900,088	TIME_LIMIT
10_18	6504	900,073	TIME_LIMIT	12_18	5019	900,05	TIME_LIMIT
10_19	4298	900,051	TIME_LIMIT	12_19	4515	900,073	TIME_LIMIT
10_20	2801	900,3	TIME_LIMIT	12_20	7158	900,053	TIME_LIMIT
10_21	3157	900,286	TIME_LIMIT	12_21	6702	900,063	TIME_LIMIT
10_22	3310	900,336	TIME_LIMIT	12_22	3312	900,052	TIME_LIMIT
10_23	4543	900,282	TIME_LIMIT	12_23	5405	900,054	TIME_LIMIT
10_24	4693	900,228	TIME_LIMIT	12_24	4485	900,068	TIME_LIMIT
10_25	2892	900,317	TIME_LIMIT	12_25	3969	900,076	TIME_LIMIT
10_26	5026	900,424	TIME_LIMIT	12_26	5799	900,069	TIME_LIMIT
10_27	3741	900,394	TIME_LIMIT	12_27	5650	900,052	TIME_LIMIT
10_28	3420	900,084	TIME_LIMIT	12_28	5506	900,052	TIME_LIMIT
10_29	4493	900,083	TIME_LIMIT	12_29	4907	900,065	TIME_LIMIT
10_30	4882	900,053	TIME_LIMIT	12_30	8129	900,099	TIME_LIMIT

Instancia	SUM_C_2j(A)	T_opt	Estado	Instancia	SUM_C_2j(A)	T_opt	Estado
14_1	6950	900,086	TIME_LIMIT	16_1	11123	900,517	TIME_LIMIT
14_2	9477	900,089	TIME_LIMIT	16_2	7091	900,462	TIME_LIMIT
14_3	10393	900,091	TIME_LIMIT	16_3	8127	900,434	TIME_LIMIT
14_4	6645	900,06	TIME_LIMIT	16_4	10273	900,618	TIME_LIMIT
14_5	5427	900,476	TIME_LIMIT	16_5	7735	900,442	TIME_LIMIT
14_6	6433	900,13	TIME_LIMIT	16_6	7367	900,471	TIME_LIMIT
14_7	5112	900,106	TIME_LIMIT	16_7	10663	900,498	TIME_LIMIT
14_8	5532	900,118	TIME_LIMIT	16_8	9848	900,451	TIME_LIMIT
14_9	9227	900,616	TIME_LIMIT	16_9	8155	900,603	TIME_LIMIT
14_10	9417	900,424	TIME_LIMIT	16_10	8743	900,435	TIME_LIMIT
14_11	6981	900,089	TIME_LIMIT	16_11	9088	900,063	TIME_LIMIT
14_12	6385	900,123	TIME_LIMIT	16_12	10014	900,036	TIME_LIMIT
14_13	7007	900,092	TIME_LIMIT	16_13	6272	900,08	TIME_LIMIT
14_14	7214	900,107	TIME_LIMIT	16_14	11678	900,037	TIME_LIMIT
14_15	8169	900,249	TIME_LIMIT	16_15	8595	900,097	TIME_LIMIT
14_16	7423	900,211	TIME_LIMIT	16_16	8723	900,053	TIME_LIMIT
14_17	8537	900,101	TIME_LIMIT	16_17	7149	900,101	TIME_LIMIT
14_18	5257	900,239	TIME_LIMIT	16_18	9932	900,118	TIME_LIMIT
14_19	7415	900,117	TIME_LIMIT	16_19	9070	900,053	TIME_LIMIT
14_20	6242	900,071	TIME_LIMIT	16_20	9862	900,118	TIME_LIMIT
14_21	5619	900,075	TIME_LIMIT	16_21	8928	900,128	TIME_LIMIT
14_22	9494	900,103	TIME_LIMIT	16_22	6906	900,194	TIME_LIMIT
14_23	5756	900,081	TIME_LIMIT	16_23	8267	900,083	TIME_LIMIT
14_24	5387	900,04	TIME_LIMIT	16_24	4015	900,06	TIME_LIMIT
14_25	4645	900,059	TIME_LIMIT	16_25	9960	900,056	TIME_LIMIT
14_26	6260	900,136	TIME_LIMIT	16_26	8083	900,133	TIME_LIMIT
14_27	4802	900,061	TIME_LIMIT	16_27	7243	900,054	TIME_LIMIT
14_28	4476	900,107	TIME_LIMIT	16_28	9808	900,103	TIME_LIMIT
14_29	7356	900,097	TIME_LIMIT	16_29	9575	900,074	TIME_LIMIT
14_30	6148	900,436	TIME_LIMIT	16_30	10564	900,081	TIME_LIMIT

ANEXO B. CÓDIGO SIMULATED ANNEALING

En este anexo se detallan los códigos usados para la implementación de la metaheurística *Simulated Annealing*, todos en lenguaje MATLAB. No obstante, no se incluyen los resultados obtenidos, como se hace en el Anexo A, debido a la gran cantidad de volumen de datos generados.

Por un lado, se expone la función para el cálculo de los tiempos de finalización y tardanzas y 3 funciones correspondientes a las 3 técnicas de generación de secuencia candidata distinta. Por otro lado, se expone la función principal del algoritmo, desde donde se llama a las funciones anteriores y que incorpora las 3 técnicas de generación de secuencia inicial y las 2 funciones de probabilidad, comentando cada técnica en el código cuando no sea necesaria su ejecución.

Las modificaciones propuestas en el apartado 6.5 también se encuentran incluidas en la función principal del algoritmo SA.

- Código para el cálculo de tiempos de finalización y tardanzas

```
%% Cálculo de tiempos de finalización y tardanzas
function [SUM_C_2j_A, tardy_B] = calculo_tiempos(n_A, d_j_S,
p_1j_S, p_2j_S, S)

Tiempo_acum_1 = cumsum(p_1j_S);
Tiempo_acum_2 = zeros(length(p_2j_S),1);
Tiempo_acum_2(1) = Tiempo_acum_1(1)+p_2j_S(1);
for j = 2:length(p_2j_S)
    Tiempo_acum_2(j) = max(Tiempo_acum_1(j), Tiempo_acum_2(j-
1))+p_2j_S(j);
end

idx_A = S<=n_A; % posiciones en S de los trabajos del conjunto
A;
SUM_C_2j_A = sum(Tiempo_acum_2(idx_A));
% idx_B = ~idx_A;
% tardy_B = find(Tiempo_acum_2(idx_B)>d_j_S(idx_B),1);

idx_B = find(S>n_A); % posiciones en S de los trabajos del
conjunto B;
tardy_B = 0;
for z = 1:length(idx_B)
    if Tiempo_acum_2(idx_B(z))>d_j_S(idx_B(z))
        tardy_B = 1;
```

```
        break
    end
end
```

- Código para GSC1

```
%% GSC 1: Random Pairwise interchange (RPI)
function s = generacion_secuencia1(s)
ind = randperm(numel(s), 2);
s(ind) = s(flip(ind));
```

- Código para GSC2

```
%% GSC 2: Consecutive Pairwise interchange (CPI)
function s = generacion_secuencia2(s)
ind1 = randperm(numel(s), 1);
if ind1 == numel(s)
    ind2 = numel(s)-1;
else
    ind2 = ind1+1;
end
ind = [ind1 ind2];
s(ind) = s(flip(ind));
```

- Código para GSC3

```
%% GSC 3: Insert Move (IM)
function s = generacion_secuencia3(s)
ind = sort(randperm(numel(s), 2), 'ascend');
if ind(2) == numel(s)
   s = [s(1:ind(1)-1); s(ind(2)); s(ind(1):ind(2)-1)];
else
   s = [s(1:ind(1)-1); s(ind(2)); s(ind(1):ind(2)-1);
s(ind(2)+1:end)];
end
```

- Código de la función principal del algoritmo SA

```
%% ALGORITMO: SIMULATED ANNEALING

fid = fopen('resultados_SA.txt', 'w');
fprintf(fid, '%s  %s  %s\n\n', 'Instancia', 'SUM_C_2j(A)',
'T_opt');
load resultados2.txt % resultados óptimos obtenidos con el
modelo exacto
%ins = 0; % para número total de instancias en resultados2.txt
(cuando haya
```

```
%resultado exacto)

for n_A = 20:10:50 % bucle para resolver varios tamaños
% n_A = 6; % n_B = n_A
n = n_A*2;
% contador = zeros(30,1); % soluciones aceptadas (programar
abajo)
for z = 1:30 % diferentes instancias para cada tamaño
% z = 1;

% Lee los datos de un archivo
clear datos

archivo = sprintf('Instancia_%d_%d', n_A, z);
fid2 = fopen(archivo, 'r');
datos = fscanf(fid2, '%f');
fclose(fid2);

%p_ij = datos(1:2*n); % [p_1jA; p_1jB; p_2jA; p_2jB]
p_1j = datos(1:n); % máquina 1
p_2j = datos(n+1:2*n); % máquina 2
d_j = datos(2*n+1:3*n); %[d_jA; d_jB]
d_Aj = datos(2*n+1:2*n+n_A);
d_Bj = datos(2*n+n_A+1:3*n);

%% GENERACION SECUENCIA INICIAL (GSI):
tic;
[~, S1_ini] = sort(d_Bj, 'ascend');
S1_ini = S1_ini+n_A;
[~, S2_ini1] = sort(p_1j(1:n_A), 'ascend');    % GS1
% [~, S2_ini2] = sort(p_2j(1:n_A), 'ascend'); % GS2
% p_j = p_1j(1:n_A)+p_2j(1:n_A);              % GS3
% [~, S2_ini3] = sort(p_j, 'ascend');

S = [S1_ini;S2_ini1]; %%% 1/2/3

p_1j_S_ini = p_1j(S);
p_2j_S_ini = p_2j(S);
d_j_S_ini = d_j(S);

[SUM_C_2j_A, ~] = calculo_tiempos(n_A, d_j_S_ini, p_1j_S_ini,
p_2j_S_ini, S);

S_best = S;
SUM_C_2j_A_best = SUM_C_2j_A;

%% BUCLE
%ins = ins+1; % Para número total de instancias en
resultados2.txt (cuando
%haya resultado exacto)
it = 0;
cont = 0; % Contador para reinicio
```

```
NI = 600*n; % Número máximo de iteraciones
To = 625*n;
beta = 1.28/n; % Activar si FP1
% alpha = 1/(6250*n); % Activar si FP2

%SUM_C_2j_A_best>resultados2(ins) &&
while it<NI
it = it+1;
%% Generación de secuencia
tardy_B_prima = 1;
while tardy_B_prima ~= 0
S_prima = generacion_secuencia1(S); % 1/2/3

p_1j_S_prima = p_1j(S_prima);
p_2j_S_prima = p_2j(S_prima);
d_j_S_prima = d_j(S_prima);

[SUM_C_2j_A_prima, tardy_B_prima] = calculo_tiempos(n_A,
d_j_S_prima, p_1j_S_prima, p_2j_S_prima, S_prima);

if tardy_B_prima == 0
% if isempty(tardy_B_prima)
    break
end
end

if SUM_C_2j_A_prima<SUM_C_2j_A
    S = S_prima;
    SUM_C_2j_A = SUM_C_2j_A_prima;
    if SUM_C_2j_A<SUM_C_2j_A_best
        S_best = S;
        SUM_C_2j_A_best = SUM_C_2j_A;
    else
        cont = cont+1;
    end
else
    cont = cont+1;
    % Adaptative cooling
    T = To/(1+beta*it);
    m = 1+((SUM_C_2j_A_prima-SUM_C_2j_A_best)/SUM_C_2j_A_prima);
% elseif (exp(-(SUM_C_2j_A_prima-
SUM_C_2j_A)/(To/(1+beta*it))))>rand(1)   % FP1
% elseif (exp(-(SUM_C_2j_A_prima-
SUM_C_2j_A)/(To/(1+alpha*it^2))))>rand(1)   % FP2
    if (exp(-(SUM_C_2j_A_prima-SUM_C_2j_A)/(T*m)))>rand(1) % FP1
    S = S_prima;
    SUM_C_2j_A = SUM_C_2j_A_prima;
    end
end

if cont>150*n
```

```
    cont = 0;
    S = S_best;
    SUM_C_2j_A = SUM_C_2j_A_best;
end

end % end while
tiempo = toc;

fprintf(fid, '%d_%d  %d  %6.3f\n', n_A, z, SUM_C_2j_A_best,
tiempo);

end
end
fclose(fid);
```