Universidad de Oviedo

Programa de Doctorado en Economía y Empresa

Metaheurísticas basadas en población para la toma de decisiones en la programación de la producción

Tesis Doctoral

Autor: Nicolás Álvarez Gil

Directores:

David de la Fuente García

Rafael Rosillo Camblor

Año 2022

Universidad de Oviedo

RESUMEN DEL CONTENIDO DE TESIS DOCTORAL

1.- Título de la Tesis	
Español/Otro Idioma: **Metaheurísticas basadas en población para la toma de decisiones en la programación de la producción**	Inglés: **Population-based metaheuristics for decision-making in manufacturing task scheduling**

2.- Autor	
Nombre: **NICOLÁS ÁLVAREZ GIL**	DNI/Pasaporte/NIE:
Programa de Doctorado: **Economía y Empresa**	
Órgano responsable: **CIP y Comisión académica del programa de doctorado en Economía y Empresa**	

RESUMEN (en español)

La programación de la producción es una actividad presente en la gran mayoría de las empresas industriales que tiene un gran impacto en la reducción de los costes de producción y en la satisfacción de los tiempos de entrega. Este tipo de problemas suelen consistir en complejos problemas de optimización combinatoria, para los que es conveniente tener métodos de solución inteligentes y eficientes para así competir en un mercado cada vez más exigente y digitalizado. Uno de los métodos más adecuados para la resolución de este tipo de problemas son las metaheurísticas, dada su capacidad de adaptarse a una gran variedad de aplicaciones y de obtener buenas soluciones en tiempo de computación reducido, aspecto crítico en la toma de decisiones operativas de las empresas industriales.

La presente tesis doctoral está enfocada en el diseño y mejora de métodos inteligentes para la toma de decisiones en el proceso de programación de tareas de producción, que sean capaces de su optimización mediante una exploración eficiente del gran espacio de búsqueda combinatorio asociado a este tipo de problemas, enfocando el estudio y la investigación realizada a la realidad más actual de las empresas industriales de producción de bienes, contribuyendo así a aproximar los avances y desarrollos obtenidos en el mundo académico y teórico a los retos actuales de estas empresas.

Para ello se estudian principalmente dos problemas concretos. Uno de ellos es una versión modificada de un conocido problema de *scheduling*, el *Flexible Job Shop Scheduling Problem*, aproximado a la realidad más actual de las empresas mediante un novedoso marco que permite la priorización de los pedidos en función de sus principales características, teniendo en cuenta estas prioridades en el modelo matemático del problema y en el proceso de resolución del mismo. Para resolver este problema se diseñó e implementó una reciente metaheurística, el *Firefly Algorithm*, dada su prometedora capacidad de exploración y su adecuación para problemas multiobjetivo. Este algoritmo, diseñado originalmente para problemas de optimización continua y utilizado con éxito en otras aplicaciones, está inspirado en el comportamiento de las luciérnagas y cómo éstas usan la luminiscencia para atraer a otras. Además, se analizan posibles mejoras para este algoritmo mediante diferentes estrategias de búsqueda local y de inicialización de la población.

El segundo problema estudiado es un problema real de una empresa internacional de producción de acero que consiste en la optimización de la secuenciación de una línea de galvanizado. Se tuvo acceso directo a la realidad del problema y a casos prácticos con datos reales. Los métodos de resolución que estaban en uso, algoritmos de la familia *Ant Colony System*, se encontraban con dificultades pare obtener buenas soluciones en ciertas situaciones, dada la creciente necesidad de reducir los niveles de inventario y la aparición de nuevos grados de acero que trajeron consigo un aumento en las restricciones de secuenciación. A través de dos publicaciones diferentes se explica el problema en detalle y se analizan las razones que generaban una mala actuación de estos algoritmos. En base a dicho

análisis, se diseñó e implementó un algoritmo de búsqueda novedoso que, combinándolo con una metaheurística *Ant System*, es capaz de asegurar buenas soluciones para todos los casos estudiados.

A través de varios experimentos, se muestra cómo las implementaciones propuestas de ambos algoritmos obtienen buenos resultados en los dos problemas estudiados, siendo uno de ellos un reto real de una empresa industrial y el otro una aproximación realista de un conocido problema de programación de tareas.

RESUMEN (en Inglés)

The process of manufacturing task scheduling is present in most industrial companies. It has a huge impact both on the reduction of production costs and on meeting the customers' expectations in terms of delivery timing. This kind of challenges usually imply complex combinatorial optimization problems demanding efficient and intelligent solution methods with a view to facing a growing digital and highly competitive market. One of the most promising solution methods for these problems are the metaheuristics, due to their wide range of potential applications and their ability to find good solution in a reasonable computational time, what is a critical issue in the operational decision-making process of the industrial companies.

The main goal of this doctoral thesis is the design and improvement of intelligent methods for the decision-making in the process of scheduling production tasks, enabling streamlining through an efficient exploration of the large combinatorial search space related to the above-mentioned problems. Both the study and research focus on the current reality of industrial goods production companies, thus contributing to bringing the advances and developments obtained in the academic, content-oriented world closer to the current challenges of those companies.

Two problems are studied for the above purposes. First, a modified version of the well-known, Flexible Job Shop Scheduling Problem (FJSP), closer to the current manufacturing paradigm, introducing a novel framework allowing for prioritization of the customer's orders on the basis of their main characteristics. Those preferences are introduced both in the mathematical model of the problem and in the resolution process. For the resolution of this problem, a metaheuristic algorithm, known as the "Firefly Algorithm" (FA), was designed and implemented since it has a promising search capacity and its suitability for multi-objective problems. The FA, originally designed for continuous optimization problems and successfully applied to other applications, is inspired in the behavior of the fireflies, and how they use their flashing lights to attract others. Additionally, different local search and initialization strategies are analysed in order to improve the performance of the FA.

The other problem studied is a real one from an international steelmaking company that consists in the optimization of the sequencing of a continuous galvanizing line. Direct access to the problem information and real instances of practical cases were provided. The solution methods in use, Ant Colony System algorithms, presented difficulties to obtain good solutions in some situations due to the increasing need to reduce stock levels and the growing number of new steel grades, that brought about more sequencing constraints. The problem is explained in detail in two different publications, and the reasons that generated a bad performance of these algorithms are analysed. On the basis of this analysis, a novel search algorithm was designed and implemented and, together with an Ant System algorithm, it was proved to be successful in finding good solutions in all the studied problem instances.

The different experiments show how the proposed implementations of both algorithms provide good results in the two problems studied, one of them being a real challenge for an industrial company, and the other a realistic approximation of a well-known task scheduling problem.

SR. PRESIDENTE DE LA COMISIÓN ACADÉMICA DEL PROGRAMA DE DOCTORADO EN ECONOMÍA Y EMPRESA

Índice de contenidos

1. Introducción

1.1. Motivación

1.1.1. Importancia de programación de tareas de producción

La programación (*scheduling*) puede definirse de manera genérica como el proceso de asignación de recursos a tareas a lo largo del tiempo (Framinan et al, 2014a), y por tanto está presente en numerosas aplicaciones del mundo real desde la asignación de turnos en un hospital hasta en la gestión de recursos de computación en la nube.

Uno de los campos en el que la programación de tareas juega un papel crucial es el mundo de las empresas manufactureras o de producción, que se dedican a la fabricación de bienes para ser entregados a sus clientes. Esta tesis se centra en la programación de tareas para este perfil de empresas y por tanto nos referiremos al problema como la programación de tareas de producción (*manufacturing scheduling*). El proceso de programación de tareas de producción consiste en definir qué se debe fabricar, dónde y cuándo (Parunak, 1991). Por tanto, un programa de producción (*schedule*) será un subconjunto del producto cartesiano de tres conjuntos: las tareas que se deben fabricar, los periodos de tiempo disponibles y los recursos que pueden ejecutar las tareas.

Tal y como se expone en Framinan et al (2014a), la gestión de la producción es un proceso que incluye numerosas decisiones destinadas a asegurar que los bienes se fabriquen con el menor coste y mayor calidad posible, y con un tiempo de entrega que satisfaga los requisitos de los clientes. Las decisiones que deben tomarse para ello pueden diferir en cuanto a su alcance, impacto y periodicidad con la que suelen realizarse. Estas diferencias permiten clasificar las decisiones en la gestión de la producción en tres categorías, en función de su impacto y rango temporal: estratégicas, tácticas y operativas. Las decisiones estratégicas de una empresa de producción (selección de proveedores, introducir un nuevo producto, apertura de una nueva planta y su localización, etc.) son aquellas que tienen un impacto enorme en la empresa y suelen tomarse con relativa baja frecuencia. Las decisiones tácticas (asignación mensual de producción a cada planta/línea de producción, definición y planificación de campañas de producción, etc.) suelen tener un impacto y rango intermedios en comparación con las decisiones estratégicas y las operativas. Por último, las decisiones operativas son aquellas decisiones de la gestión de la producción de más bajo impacto directo y mayor frecuencia.

La programación de tareas de producción cae en esta última categoría. El hecho de que una sola decisión operativa no tenga un impacto directo y notable sobre los resultados finales de la empresa no significa que sean decisiones menos importantes. Al tomarse con una frecuencia tan alta, una mejora en el proceso de la toma de decisiones operativas hace que en conjunto tengan un gran impacto final a lo largo del tiempo. Por ello, el diseño y uso de técnicas capaces de optimizar la programación de tareas de producción es fundamental para este tipo de empresas, y objetivo principal de esta tesis doctoral (Objetivo Principal 1).

Los problemas de optimización de la programación de la producción llevan estudiándose en la literatura académica durante muchos años (Graham et al, 1979; Gao et al, 2019; Gen & Li, 2013;), estando la gran mayoría de ellos enfocados principalmente a problemas de *scheduling* teóricos. Sim embargo, a día de hoy, en algunas empresas todavía se toman decisiones de programación de manera desestructurada, en base a la experiencia de los programadores (*schedulers*) y con información limitada, dando lugar por tanto a una mala utilización de los recursos, retrasos en las fechas de entrega y mayores costes de producción (Framinan et al, 2014a).

Los problemas de programación de tareas de producción requieren decisiones complejas que deben ser tomadas de manera estructurada y metódica para garantizar una óptima utilización de los recursos y satisfacción de los clientes, y por tanto aumentar los beneficios de la empresa. Para ello, es fundamental desarrollar modelos y algoritmos capaces de hacerlo. Para resolver un problema de *scheduling*, deben definirse de manera clara las variables involucradas (tareas, recursos, tiempos, etc.), las restricciones del problema y sus objetivos (optimizar coste de producción, el tiempo de producción, la utilización de recursos, etc.) y hacer una formulación matemática de todo ello para tener acotado y definido el problema de programación en cuestión, dando lugar al modelo del problema (Framinan et al, 2014b). Una vez definido el modelo, se debe desarrollar un método de solución (Framinan et al, 2014c), es decir, un método (algoritmo) capaz de explorar el espacio de soluciones de una manera eficiente y proporcionar buenas soluciones en un tiempo operativo para la actividad real de la empresa.

El reciente nuevo paradigma de las empresas de fabricación, conocido como Industria 4.0 (Brettel et al., 2014; Wang et al., 2016a), promovido por las tecnologías emergentes y cada vez más exigente de cara a la satisfacción de los clientes y la elevada competitividad en el mercado (Güçdemir & Selim, 2017), hace que el desarrollo de herramientas digitales para la resolución de los problemas de *scheduling*, como los algoritmos de optimización inteligentes, sea un aspecto fundamental para mantenerse en una buena posición de mercado. Tal es la rapidez con la que cambia el paradigma de las empresas industriales, que actualmente ya se está hablando de Industria 5.0, la cual sería una combinación de los pilares tecnológicos que permiten una industria más inteligente e interconectada que dieron pie a la Industria 4.0 (Inteligencia Artificial, Internet de la Cosas, computación en la nube, etc.) con la creatividad humana de los expertos involucrados (ElFar et al., 2020; Maddikunta et al., 2021).

Estos dos últimos puntos, el hecho de que gran cantidad de los estudios existentes sean aplicados a problemas de *scheduling* teóricos y la imperante necesidad de aplicar métodos de optimización inteligentes en las empresas de producción actuales, nos lleva al segundo objetivo principal de esta tesis doctoral, que busca adaptar las técnicas existentes de optimización de la programación de tareas a la realidad actual de las empresas (Objetivo Principal 2).

1.1.2. Complejidad de los problemas de *scheduling*

Los problemas de programación de tareas suelen ser problemas de optimización en los que se busca una asignación de recursos y una ordenación de las tareas que optimicen ciertos criterios: minimizar el coste de producción, minimizar los tiempos de entrega, maximizar la calidad de los productos y la satisfacción de los clientes, etc. En muchos casos, sobre todo en escenarios industriales reales, se busca optimizar varios objetivos simultáneamente, dando lugar a problemas de optimización multiobjetivo (Gen & Li, 2013).

A la hora de buscar un método para la resolución de un problema de optimización es importante analizar primero su complejidad. En El-Ghazali (2009) se puede encontrar información detallada sobre la complejidad de los algoritmos y problemas de optimización, y es la fuente principal de lo que se expone a continuación respecto a la teoría de la complejidad.

La complejidad de un problema equivale a la complejidad del mejor algoritmo capaz de resolver dicho problema. La teoría de la complejidad se basa en problemas de decisión, pero todo problema de optimización puede reducirse a un problema de decisión (un problema de optimización de encontrar una ordenación de tareas que minimice el tiempo de producción puede reducirse al problema de decisión de determinar si existe o no una ordenación con un tiempo de producción menor que cierto límite). Esta teoría de la complejidad categoriza los problemas en tres clases:

- **Complejidad de clase P**: son aquellos problemas de decisión que pueden ser resueltos por una máquina determinista en tiempo polinomial. Un sistema determinista es aquel en el que no hay aleatoriedad involucrada en el desarrollo de estados futuros. Tiempo polinomial es cuando el tiempo de ejecución es menor que cierto valor calculado a partir del número de variables implicadas usando una forma polinómica. En principio, este tipo de problemas son los más fáciles de resolver.
- **Complejidad de clase NP**: son aquellos problemas de decisión que pueden ser resueltos por un algoritmo no determinista en tiempo polinomial. La cuestión $P = NP$ es todavía una de las preguntas abiertas que mayor impacto tendría en la teoría de la complejidad. De momento solo se sabe que $P \subseteq NP$ dado que para todo problema en P existiría un algoritmo no determinista para resolverlo.
- **Complejidad de clase NP-*completo***: un problema de decisión es de esta clase si todos los problemas de la clase NP son polinomialmente reducibles a dicho problema. Por tanto, si existiera un algoritmo polinomial determinista para resolver un problema de clase NP-completo, ello supondría que todos los problemas de clase NP podrían ser resueltos en tiempo polinomial. Este tipo de problemas se consideran los más difíciles de resolver.

Dentro de los problemas de optimización, la programación de tareas de producción suele caer dentro del conjunto de problemas conocidos como optimización combinatoria. Un problema de optimización combinatoria $P = (S, f)$ (Blum & Roli, 2003) es aquel en el que las soluciones están codificadas en variables discretas y la solución óptima por tanto se encuentra en un conjunto finito de soluciones, y puede definirse de manera formal por:

- Un conjunto de variables $X = \{x_1, x_2, \dots, x_n\}$;
- El dominio de cada variable $D_1, D_2, \dots, D_n$;
- Restricciones definidas sobre las variables;
- Una función objetivo f a ser minimizada (todo problema de optimización puede ser formulado como un problema de minimización), donde $f: D_1 \times D_2 \times \dots \times D_n \rightarrow \mathbb{R}^+$

El conjunto de todas las soluciones factibles sería entonces:

$$S = \{s = \{ (x_1, v_1), (x_2, v_2), \dots, (x_n, v_n)\} \mid v_i \in D_i, \quad s \text{ satisfaga todas las restricciones}\}$$

Por tanto, resolver un problema de optimización combinatoria consiste en encontrar una solución óptima $s^* \in S$, siendo S el espacio de soluciones o de búsqueda, que tenga el mínimo valor de la función objetivo $f(s^*) \leq f(s) \; \forall \, s \in S$.

Muchos problemas de *scheduling* estudiados en la literatura, como en *Job Shop Scheduling Problem* (Çaliş & Bulkan, 2015) o el *Flexible Job Shop Scheduling Problem* (Khan & Chaudhry, 2015), son problemas de optimización combinatoria dado que consisten en encontrar una configuración de variables discretas que optimicen cierto criterio. Además, como otros muchos problemas de optimización combinatoria, estos problemas de *scheduling* se ha demostrado que son problemas NP-*hard* (Applegate & Cook, 1991). Los problemas NP-*hard* son aquellos problemas de optimización cuyos problemas de decisión asociados son NP-completos y requieren tiempo exponencial para ser resueltos óptimamente (El-Ghazali, 2009). Esta razón hace que los métodos aproximados sean una alternativa adecuada para este tipo de problemas en lugar de los métodos exactos. Estos problemas de *scheduling* teóricos suelen tener varias asunciones para simplificarlos a una formulación más genérica, por lo que la mayoría de los problemas de programación de tareas encontrados en la realidad de las empresas de producción son también problemas NP-*hard*.

1.1.3. Métodos de solución para problemas de scheduling

Dependiendo de la complejidad del problema, éste puede ser resuelto mediante métodos exactos (*Branch & Bound*, *Constraint Programming*, etc.) o aproximados (reglas heurísticas, metaheurísticas, etc.).

Los métodos exactos son aquellos capaces de obtener y garantizar una solución óptima al problema. Dentro de los algoritmos exactos, se pueden encontrar algoritmos clásicos como (El-Ghazali, 2009): *dynammic programming*, algoritmos de la familia *branch & X* (*Branch & Bound*, *Branch & Cut*, *Branch & Price*), *constraint programming*, etc. Estos métodos enumerativos pueden verse como algoritmos de búsqueda en árbol, en los que la búsqueda se lleva a cabo a lo largo de todo el espacio de búsqueda, subdividiendo el problema en problemas más simples (El-Ghazali, 2009). El punto fuerte de estos métodos de ser capaces de asegurar soluciones óptimas conlleva un elevado coste computacional, que normalmente crece de manera exponencial con el tamaño del problema (Framinan et al., 2014c). Ésta es una de las razones que hace que esta clase de métodos no sean operativos ni la opción más usada en el ámbito industrial, donde el tamaño de los problemas de *scheduling* suele ser elevado, y se requieren tiempos de computación reducidos dado que normalmente se trata de decisiones operativas que se toman con alta frecuencia.

En dichas situaciones donde no se puede aplicar un algoritmo exacto o el tiempo de éstos no es práctico, los algoritmos aproximados son más interesantes. Los métodos aproximados suelen proporcionar soluciones de calidad en un tiempo razonable para el ámbito práctico, lo que los convierte en la principal opción para problemas de tamaño elevado. Sin embargo, no son capaces de garantizar ni demostrar la obtención de un óptimo global del problema. Hay gran cantidad de métodos aproximados para casi todos los problemas de *scheduling* NP-hard, que difieren mucho en cuanto a las ideas y conceptos en los que se basan (Framinan et al., 2014c). Una de las posibles clasificaciones de los algoritmos aproximados es diferenciar entre heurísticas específicas y las metaheurísticas.

Las heurísticas específicas son aquellas diseñadas para un problema en particular, aunque ello no quiera decir que no puedan adaptarse a otros problemas. Dentro de esta clase de algoritmos aproximados es útil diferenciar entre las heurísticas constructivas y las heurísticas de mejora. Los enfoques constructivos generan soluciones basándose solamente en las entradas del problema, sin tener en cuenta ninguna otra solución del problema. Suelen usar reglas de programación para la construcción, muchas veces basadas en la experiencia de los expertos encargados de la toma de decisión en la programación de tareas, y por los resultados de su aplicación. Por otro lado, las heurísticas de mejora tratan de mejorar una solución ya existente (muchas veces generadas por heurísticas constructivas). Estas heurísticas aplican una serie de reglas de modificación a una solución completa hasta que se cumple cierto criterio de finalización (p.ej. límite de tiempo, límite de iteraciones). Merece la pena destacar los procedimientos de búsqueda local o *local search*, muy utilizada por las metaheurísticas, que se basan en explorar el *vecindario* de una solución para intentar encontrar una solución mejor que este próxima en el espacio de búsqueda a la solución actual. Las desventajas de las heurísticas específicas son su alta dependencia del problema a resolver y, dado que gran parte de los problemas de optimización combinatoria como los problemas de *manufacturing scheduling* son no-convexos (Framinan et al., 2014c), estos métodos aproximados tienden a proporcionar soluciones situadas en o próximas a óptimos locales (Groetschel & Lovasz, 1993), siendo difícil para ellas explorar lo suficiente como para acercarse al óptimo global. Estas dos desventajas son una de las principales razones por las que las metaheurísticas son capaces de encontrar mejores soluciones y, por tanto, son una mejor alternativa para los problemas de *manufacturing scheduling*.

1.1.4. Las Metaheurísticas

Las metaheurísticas son metodologías de alto nivel que proporcionan un conjunto de directrices o estrategias a seguir para explorar el espacio de soluciones de un problema, sin estar específicamente definidas para ningún problema en concreto (Glover, 1986). Un algoritmo metaheurístico es una implementación de un algoritmo de optimización para un problema específico siguiendo las reglas generales definidas por una metaheurística.

Las metaheurísticas no están sujetas a la explosión combinatoria que experimentan los métodos de optimización exactos, dado que están concebidas para obtener soluciones los *suficientemente buenas*, en un *tiempo razonable* (Sörensen & Glover, 2013). Por ello, se ha demostrado que son una alternativa viable y muchas veces superior a los métodos exactos tradicionales como *Mixed-Integer Programming*, *Branch & Bound*, *Dynamic Programming*, etc. (Sörensen & Glover, 2013), especialmente para problemas complejos o de gran tamaño, como son los problemas de *scheduling* de empresas industriales de producción. Esta clase de métodos aproximados proporciona un balance entre calidad de las soluciones y tiempo de computación.

Además, las metaheurísticas son más flexibles que los métodos exactos en dos aspectos principales (Sörensen & Glover, 2013). Por un lado, las metaheurísticas están definidas en términos generales, y los algoritmos metaheurísticos pueden ser adaptados a una gran cantidad de problemas reales, satisfaciendo sus necesidades de calidad de soluciones y tiempo de computación. Por otro lado, las metaheurísticas no requieren una formulación matemática rigurosa y completa del problema de optimización, facilitando su uso en problemas complejos y reales.

Existen una gran variedad de metaheurísticas, pero todas comparten una serie de características (Blum & Roli, 2003):

- Son estrategias que guían el proceso de búsqueda, encontrando soluciones cada vez mejores de manera iterativa.
- Su objetivo es explorar el espacio de soluciones de manera eficiente para encontrar soluciones óptimas o próximas al óptimo global.
- Normalmente son métodos no deterministas, con componentes estocásticos, que van desde simples métodos de búsqueda local iterativa a sofisticados mecanismos de aprendizaje.
- Suelen tener estrategias para evitar quedarse atrapadas en óptimos locales, lo que es fundamental en problemas de optimización no convexos.
- Sus conceptos básicos permiten una descripción a nivel abstracto, sin estar asociadas a ningún problema específico.
- Los algoritmos metaheurísticos pueden hacer uso de conocimiento específico de los problemas en forma de heurísticas controladas por una estrategia de nivel superior.
- Las más actuales hacen uso de la experiencia de búsqueda, en forma de algún tipo de memoria o sistema de aprendizaje, para guiar la exploración.
- Permiten obtener una solución en cualquier momento de su ejecución (*anytime behavior*), lo que las hace atractivas para su uso en problemas de optimización de mundo industrial.
- Proporcionan un balance entre la diversificación (exploración e identificación de regiones prometedoras del espacio de búsqueda) y la intensificación (explotación de la experiencia acumulada).

Algunas de las metaheurísticas que se pueden encontrar en la literatura y que han sido ampliamente utilizadas para resolver problemas de optimización son: *Ant Colony Optimization* (ACO) (Dorigo & Stützle, 2002), *Artificial Immune Systems* (AIS) (Farmer et al., 1986), *Bee Colony* (BC) (Yonezawa & Kikuchi, 1996), *Evolutionary Programming* (EP) (Fogel, 1962), *Evolution Strategies* (ES) (Rechenberg, 1965), *Genetic Algorithms* (GA) (Holland, 1962), *Guided Local Search* (GLS)

(Voudouris, 1998), *Genetic Programming* (GP) (Koza, 1992), *Greedy Adaptive Search Procedure* (GRASP) (Feo & Resende, 1995) , *Iterated Local Search* (ILS) (Martin et al., 1991), *Particle Swarm Optimization* (PSO) (Kennedy & Eberhart, 1995), *Simulated Annealing* (SA) (Kirkpatrick et al., 1983), *Scatter Search* (SS) (Marti et al., 2006), *Tabu Search* (TS) (Glover, 1989), *Variable Neighborhood Search* (VNS) (Mladenovic, 1995) o *Firefly Algorithm* (FA) (Yang, 2008).

En la investigación realizada para la presente tesis doctoral, se estudiaron en profundidad dos de las metaheurísticas anteriormente citadas, *Ant Colony Optimization* (ACO) y *Firefly Algorithm* (FA). Las características de estas dos metaheurísticas y su idoneidad para los problemas tratados se exponen en los capítulos correspondientes.

1.2. Objetivos

Los objetivos principales de la presente tesis doctoral son:

- **Objetivo Principal 1**: Diseñar y mejorar métodos inteligentes para la toma de decisiones en el proceso de programación de tareas de producción, que sean capaces de su optimización mediante una exploración eficiente del gran espacio de búsqueda combinatorio asociado a este tipo de problemas.
- **Objetivo Principal 2**: Enfocar el estudio y la investigación realizada a la realidad más actual de las empresas industriales de producción de bienes, contribuyendo a aproximar los avances y desarrollos obtenidos en el mundo académico y teórico a los retos actuales de estas empresas.

A su vez, estos objetivos principales se pueden descomponer es unos objetivos específicos más concretos que guían la estructura del estudio:

- **Objetivo Específico 1**: Acercar uno de los problemas de *scheduling* teóricos más estudiados en la literatura, el *Flexible Job Shop Scheduling Problem* (FJSP), al paradigma actual de la industria (Industria 4.0/5.0), en que algunos de los pilares fundamentales son la flexibilidad de la producción, el enfoque a la satisfacción de los clientes mediante la personalización y priorización, y la necesidad de satisfacer varios objetivos simultáneamente. De esta manera se busca facilitar la utilización y la transferencia del amplio conocimiento acumulado durante los años sobre este problema a un entorno industrial más real y actual.
- **Objetivo Específico 2**: Diseñar e implementar una de las más recientes metaheurísticas, el *Firefly Algorithm* (FA), para abordar el enfoque adaptado del FJSP, y el análisis de su adecuación como método de optimización dado su potencial de exploración para problemas multiobjetivo.
- **Objetivo Específico 3**: Diseñar, implementar y comparar diferentes estrategias de búsqueda local e inicialización que puedan ser incorporadas al FA para la resolución de problemas FJSP obteniendo una versión híbrida del algoritmo capaz de mejorar las soluciones obtenidas y dar lugar a un enfoque de optimización más robusto y completo.
- **Objetivo Específico 4**: Introducir un problema de programación de la producción de una empresa industrial real, los retos que supone su optimización y las principales diferencias con los problemas de *scheduling* estudiados en la literatura.
- **Objetivo Específico 5**: Diseñar e implementar un método de optimización basado en computación evolutiva para el problema real de la empresa estudiada, capaz de superar los retos y dificultades detectados para su resolución de una manera eficiente y fiable, aportando una solución robusta para la empresa.

1.3. Estructura de la tesis

La presente tesis doctoral se presenta como un compendio de publicaciones científicas. La tesis se estructura en 6 capítulos. El capítulo actual (Capítulo 1), a modo de introducción, se centra en la motivación y en la descripción de los objetivos. A largo de este capítulo se introduce la importancia de la optimización programación de tareas de producción, su complejidad y los métodos de optimización existentes.

El Capítulo 2 aborda simultáneamente los objetivos específicos 1 y 2, y corresponde con la primera publicación científica. En dicho artículo se propone un marco que permite acercar y adaptar el problema del FJSP al paradigma de la Industria 4.0, y se explica en profundidad el método de optimización elegido para su resolución, el *Firefly Algorithm* (FA), que es una metaheurística basada en población introducida por primera vez en 2008 con unas características concretas que la hacen especialmente interesante para este tipo de problemas de optimización combinatoria multiobjetivo.

Tras la investigación realizada para el desarrollo de la primera publicación, se detectó un campo de mejora en la actuación del FA para la resolución del FSJP, introduciendo estrategias de búsqueda local e inicialización que permitan mejorar la exploración del espacio de búsqueda y evitar el estancamiento en regiones subóptimas del mismo. En el Capítulo 3 se presenta una publicación que expone dichas estrategias y muestra cómo su aplicación consigue mejorar notablemente los resultados obtenidos en diferentes instancias del FJSP, abordando así el objetivo específico 3.

Durante el desarrollo de la tesis, se tuvo la oportunidad de acceder y conocer en detalle un problema de programación de tareas real de una empresa internacional de producción de acero, de gran interés por su complejidad e impacto en la empresa. El problema consiste en la programación de la producción de una línea de galvanizado, en el que hay que optimizar el orden en el que ciertas bobinas de acero deben ser procesadas en la línea acabadora. Este problema dio lugar a dos publicaciones, y por tanto a dos capítulos de las tesis.

En el Capítulo 4 se introduce el problema real de secuenciación de la línea de galvanizado. Consiste en una publicación centrada en abrir al mundo académico y científico nuevos problemas a analizar, aportando datos reales que permitan una investigación práctica y el desarrollo de nuevos métodos de resolución que contribuyan al avance en su rama de conocimiento. Para ello, además de introducir de manera práctica el problema, se presentan 30 instancias reales del mismo, de modo accesible para toda la comunidad científica interesada, explicando cómo se deben utilizar y el objetivo del problema.

En el Capítulo 5, también relacionado con el problema real de *scheduling* de la línea de galvanizado de la empresa de producción de acero, se profundiza en el problema y en los retos que supone su resolución. Para ello, a través de otra publicación científica, se define el problema de una manera más teórica y formal, analizando las similitudes y diferencias con otros problemas de optimización combinatoria estudiados en la literatura. Desde esta manera, junto con el Capítulo 4, se aborda el objetivo específico 4. Además, como objetivo principal del estudio y en relación con el objetivo específico 5, se presentan las dificultades a las que se enfrentaban los métodos de optimización en uso, algoritmos de la familia *Ant Colony Optimization* (ACO), y se introduce una estrategia novedosa de exploración constructiva que, combinándola con ACO, da lugar a una metaheurística híbrida capaz de garantizar soluciones factibles para el problema, demostrándolo con varios experimentos con las 30 instancias explicadas en la publicación del Capítulo 4.

2. Capítulo 1: A discrete firefly algorithm for solving the flexible job-shop scheduling problem in a make-to-order manufacturing system

Este capítulo corresponde con la primera publicación científica que se centra simultáneamente en dos de los objetivos específicos de la tesis doctoral (objetivos específicos 1 y 2). Por un lado, se propone una manera de acercar uno de los problemas de *scheduling* más estudiados en la literatura al paradigma actual de las empresas industriales de producción. Durante los últimos años, la industria manufacturera está experimentando un cambio debido al incremento de la competitividad global, al aumento en la demanda de una producción cada vez más flexible y personalizada, y al gran impacto disruptivo de las nuevas tecnologías (redes de sensórica avanzada, gran incremento en las capacidades de almacenaje y computación de grades cantidades de datos, *Internet of Things*, etc.). Este nuevo paradigma industrial, conocido como la cuarta revolución industrial o Industria 4.0 (Brettel et al., 2014; Wang et al., 2016a), tiene como unos de sus ejes centrales el enfoque en la satisfacción de los clientes mediante sistemas productivos más flexibles que permitan la personalización de los productos (Güçdemir & Selim, 2017). El sistema productivo que más se adapta a esta necesidad es el sistema de producción flexible basados en producción bajo pedido (Morinaga et al., 2014; Wang et al., 2016b). Estos sistemas productivos llevan siendo estudiados durante varios años mediante su abstracción teórica conocida como el *Flexible Job Shop Scheduling Problem* (FJSP).

En el presente artículo se presenta una adaptación del FJSP al paradigma industrial de la Industria 4.0. En el marco propuesto, se permite personalizar los pedidos, introduciendo una metodología para simular dicho proceso de personalización a partir de instancias clásicas del FJSP. Por otro lado, como en la gran mayoría de los problemas de *scheduling* reales, se aborda un enfoque multiobjetivo que permita satisfacer varios objetivos simultáneamente. Uno de los principales objetivos a satisfacer es la satisfacción de los clientes, para lo que se expone una manera de analizar el peso de cada pedido a partir de varias variables (su importancia o peso como cliente, los plazos de entrega, etc.). Esta información clave se introduce en el modelo del FJSP mediante unos coeficientes de prioridad que dan diferente peso a los pedidos de cada cliente, en contraposición al enfoque clásico del FJSP en el que se da el mismo peso a todos los pedidos.

La otra parte principal del trabajo consiste en la aplicación del *Firefly Algorithm* (FA) como método de solución para el problema planteado. Se identificó esta metaheurística, introducida por primera vez en 2008, como una de las más prometedoras al tratarse de una metaheurística evolutiva basada en población con gran capacidad de exploración, y al haber superado los resultados de otras conocidas metaheurísticas como GA y PSO en otras aplicaciones (Yang, 2008; Lohrer, 2013). El FA está inspirado en el comportamiento de las luciérnagas, y como éstas emplean su luminiscencia para atraer a otras. En el FA, una luciérnaga corresponde a una solución del problema a resolver, y su luminiscencia está directamente relacionada con el valor de la función objetivo de la solución. De esta manera, en cada iteración, las luciérnagas se mueven por el espacio de búsqueda en función de la atracción que generen el resto de luciérnagas y de una componente de movimiento aleatoria. Originalmente el FA fue introducido para problemas de optimización continua por lo que en el artículo se exponen, además de todas sus características generales, las modificaciones necesarias para aplicar esta metaheurísticas a un problema de optimización discreta como el FJSP.

Finalmente, la implementación propuesta del FA multiobjetivo se compara con otros métodos en varias instancias clásicas del FJSP, y posteriormente se emplea para resolver distintos casos del problema introducido del FJSP adaptado con los coeficientes de prioridad, demostrando como consigue buenas soluciones dando mayor peso a acabar antes los pedidos de mayor prioridad.

Central European Journal of Operations Research
https://doi.org/10.1007/s10100-020-00701-w

A discrete firefly algorithm for solving the flexible job-shop scheduling problem in a make-to-order manufacturing system

Nicolás Álvarez-Gil[1] · Rafael Rosillo[1] · David de la Fuente[1] · Raúl Pino[1]

Abstract
This work presents a multi-objective discrete firefly algorithm (MO-DFFA) for solving the flexible job-shop scheduling problem (FJSP) in a make-to-order production. Three different objectives are minimised simultaneously, being these objectives the weighted sum of the completion times of the orders, the workload of the critical machine and the total workload of all machines. Customer orders are ranked by priority according to the variables that the company considers the most relevant for its classification. Then, this priority is included in the FJSP model giving more preference in the scheduling phase to client requests with higher priority while, at the same, the volume of work of the resources is balanced to avoid machines saturation or under-utilization. With this approach both customer and company requirements can be satisfied and balanced. Furthermore, in the proposed framework customers can customize their orders choosing between some eligible product characteristics, which are considered as the different manufacturing operations which constitute each job. For solving the two sub-problems (i.e. operations assignment and sequencing) required for the scheduling of a flexible manufacturing system, we implemented a discrete version of the firefly algorithm metaheuristic. The computational results of several problem instances show that the presented MO-DFFA is a promising and efficient alternative to solve the FJSP in a customer-centric production system.

✉ Nicolás Álvarez-Gil
uo226901@uniovi.es

Rafael Rosillo
rosillo@uniovi.es

David de la Fuente
david@uniovi.es

Raúl Pino
pino@uniovi.es

[1] Business Management Department, Polytechnic School of Engineering of Gijón, University of Oviedo, C\ Wilfredo Ricart s/n, 33203 Gijón, Asturias, Spain

Published online: 10 August 2020

Keywords Flexible job shop scheduling · Firefly algorithm · Make-to-order

1 Introduction

The increase of the global competitiveness and performance boosted by industrial globalization, the growing demand of flexible and customized products over standard ones and the new emerging technologies (i.e. Big Data, wireless sensor networks, embedded systems, Internet of Things, etc.) are driving manufacturing industry to the Fourth Industrial Revolution, named as Industry 4.0 (Brettel et al. 2014; Wang et al. 2016a). This new paradigm marks a shift of focus from mass production to mass customization. Therefore, customer-centric production is becoming an important strategy for manufacturing companies to be competitive (Güçdemir and Selim 2017). Many of them select make-to-order manufacturing based on flexible job-shop systems to satisfy the specific market demand (Morinaga et al. 2014; Wang et al. 2016b).

In the literature, production scheduling in flexible job-shop systems is known as the flexible job-shop scheduling problem (FJSP). It is an extension of the job-shop scheduling problem (JSP), a well-known NP-hard problem and one of the most difficult problems in the field of managing and planning of manufacturing process (Garey et al. 1976; Pezzella et al. 2008). In the JSP, the jobs to be produced represent a sequence of operations that must be scheduled in a way that certain criteria are optimized (e.g. maximal completion time, total tardiness, etc.). The operations have initially a fixed machine assigned where they should be processed without interruption.

Nevertheless, new technologies and flexible manufacturing systems make that the assumption of the JSP that one operation can only be handled by a single machine does not meet the requirements of modern manufacturing systems (Lunardi and Voos 2018). In consequence, it seems to be more appropriate to classify scheduling in Industry 4.0 factories as flexible job or flow shop (Ivanov et al. 2016). Since in the FJSP each operation can be processed by more than one compatible machines, a higher flexibility is considered while, at the same time, the complexity of the problem increases because an additional decision is required: which machine should be selected for each operation. Hence, the FJSP can be seen as two problems: (1) the assignment of start/end times and resources for each operation, and (2) the sequencing of the assigned operations.

Due to the complexity of FJSP, methods that are able to obtain, in a reasonable time, near-optimal solutions have come up in recent years (Nasser and Ghasemisha-bankareh 2013). In particular, metaheuristic algorithms such as tabu search (TS), genetic algorithms (GA), particle swarm optimization (PSO) or simulated annealing (SA) have been proved to lead to better results than other methods such as simple heuristics algorithms like the classical dispatching rules (Najid et al. 2002; Du et al. 2008; Pezzella et al. 2008). Recently, an algorithm proposed in Yang (2008), the firefly algorithm (FA), has been shown to outperform other algorithms (i.e. PSO and GA) in multiple applications (see Yang 2010 and Lohrer 2013). In this work, we propose a discrete FA version to solve the FJSP in a customer-centric make-to-order production system. We would like to highlight the following contributions. First, a model in which the customer orders can be customized and ranked by priority is proposed,

aiming to bring the classical FJSP closer to the Industry 4.0 paradigm. Secondly, a discrete version of the FA is implemented to solve the proposed model considering the specific characteristics of each order and its priority.

The content of this work is organised in the following sections: a literature review about the research topic of production scheduling and the different methods that have been used for its resolution is conducted in Sect. 2. Section 3 explains the model and the problem formulation. The discrete version of the FA proposed to solve the FJSP is described in detail in Sect. 4. Finally, Sect. 5 shows the analysis of the obtained results and Sect. 6 provides the conclusions and future research directions.

2 Literature review

Since the early 1950's, extensive research has been done on the topic of scheduling (Lawler et al. 1993). In the production branch of scheduling, problems have been categorized into numerous schemes following different dimensions (Graves 1981): the requirement generation (i.e. by customer order or by inventory replenishment decisions), the processing complexity (i.e. number of processing steps associated to each manufacturing task) and the scheduling criteria.

Regarding the processing complexity, a common breakdown can be studies about a single machine, parallel machines, flow shop and job shop scheduling problems (Brucker 2001). Single machine problems (Biskup 1998; Kuo and Yang 2006) and parallel machines problems (Lee 1991; De Paula et al. 2007) are the simplest problem forms since all tasks require one single processing step which may be done in one machine or in one of multiple identical parallel machines, respectively. In flow-shop scheduling problem (Osman and Potts 1989; Chen et al. 1995; Ruiz and Vázquez-Rodríguez 2010) the tasks should be processed on a set of machines following the same production flow (i.e. identical precedence ordering of the processing steps). The job-shop scheduling problem (Brucker et al. 1994; Gonçalves et al. 2005; Sha and Hsu 2006) is more general since alternative routings for a task are allowed. In combinatorial optimization, the JSP is a very well-known and studied problem due to its complexity (Lawler et al. 1993).

Nevertheless, all these scheduling approaches lack flexibility and may not suit some of the requirements of modern manufacturing systems, which should be flexible enough to offer customized products as the market demands. In the flexible job-shop scheduling problem, a higher level of flexibility in the production is considered since both alternative routing and machines are allowed for the tasks. As other scheduling problems, the FJSP has been studied for many years and different heuristics techniques such as dispatching rules, local search algorithms and metaheuristics have been proposed for solving it. These approximate methods are usually preferred because exact algorithms may need too much computational time to be used for industrial applications (Pezzella et al. 2008).

A common classification of these procedures is to differentiate between hierarchical and integrated approaches. The former (Brandimarte 1993; Paulli 1995; Kacem et al. 2002a) consider the assignment and the sequencing problems separately, generally using first a heuristic or dispatching rule for the assignment problem, and then solving

the resulting sequencing problem (JSP) with a different technique (i.e. tabu search). Solving the two FJSP subproblems separately reduces the complexity of the problem, but it usually leads to worse results compared to the integrated approaches in which both subproblems are considered simultaneously (Vaessens et al. 1994). Two relevant studies in which the integrated approach is adopted are Dauzère-Pérès and Paulli (1997) and Mastrolilli and Gambardella (2000), both using tabu search algorithms (TS).

Concerning the approximate methods, metaheuristic algorithms have become a widely used alternative because they can provide better solutions than simple heuristics, and usually require less computational time than exact algorithms (Blum and Roli 2003). Kacem et al. (2002a, b) used a genetic algorithm (GA) controlled by an assignment model based on the approach by localization, for both mono-objective and multi-objective FJSP. This approach by localization gave the basis for the initial population strategy presented in the GA of Pezzella et al. (2008), which also includes novel methods for the selection and reproduction of the individuals. In De Giovanni and Pezzella (2010), a genetic algorithm improved with local search heuristics is proposed to solve the flexible and distributed job-shop scheduling problem. Bagheri et al. (2010) presented an integrated approach for minimizing the makespan in a FJSP using the artificial immune algorithm. In Zhang et al. (2009) they use a hybrid metaheuristic approach to solve the multi-objective FJSP, combining particle swarm optimization (PSO) and a local search procedure based on TS. Moslehi and Mahnam (2011) also present another integrated multi-objective approach that hybridizes PSO and a local search algorithm. Furthermore, constructive metaheuristics such as ant colony optimization (ACO) can be used to solve the FJSP. Rossi and Dini (2007) introduced an ACO-based software system considering both transportation and set-up times, and Xing et al. (2010) proposed an approach that integrates an ACO model and a knowledge-based model.

In this study, we present an integrated approach that uses the Firefly Algorithm to solve the multi-objective FJSP. Since the first time it was formulated (Yang 2008), the FA has been successfully applied to diverse applications. To name a few, there are studies on the application of the FA to digital image compression (Horng 2012), antenna design optimization (Chatterjee et al. 2012), feature selection (Banati and Bajaj 2011) or clustering and classification (Senthilnath et al. 2011). However, compared to other metaheuristics, scarce research has been done on the application of the FA on combinatorial optimization, especially on the FJSP.

3 Problem definition

In the proposed framework, orders enter to the system on a daily basis. Customers can fulfil the functional and non-functional requirements of their orders by different combinations of the product characteristics. Each product attribute is associated to a different manufacturing operation that can be handled by any machine from set of compatible machines, requiring each machine a different processing time (Table 1).

In this case, simplified for illustration, customers can define the functional aspects of their orders (i.e. materials, components, structure, etc.) by choosing one eligible

Table 1 Eligible operations, compatible machines and processing times

Options	Machines							
	1	2	3	4	5	6	7	8
Type 1								
1	5	–	7	–	3	–	–	9
2	–	–	–	4	3	–	6	–
3	–	–	–	5	–	2	4	–
4	4	–	3	–	8	–	9	–
5	–	8	–	2	6	–	–	–
6	–	–	–	5	–	4	1	7
7	10	–	9	–	4	7	–	–
Type 2								
1	–	–	–	–	–	5	2	4
2	–	5	6	4	–	9	–	–
3	1	–	–	6	–	10	–	7
4	–	1	6	–	–	–	8	–
5	–	11	–	8	9	5	6	–
6	–	–	2	–	3	–	5	7
7	3	–	7	–	9	–	–	–
Type 3								
1	–	–	7	4	–	–	6	–
2	–	9	–	–	4	2	–	–
3	–	–	–	6	7	–	3	6
4	–	–	1	–	3	–	–	–
5	11	–	9	–	–	–	6	4
6	–	5	9	10	–	–	–	–
7	5	–	2	–	–	–	3	–
Type 4 (optional)								
1	–	9	–	–	11	9	–	5
2	–	–	–	3	–	6	–	7
3	2	8	–	9	–	–	–	–
Type 5 (optional)								
1	–	4	7	–	9	–	10	–
2	–	9	–	8	5	–	–	1
3	9	–	3	–	1	5	–	–

option from each of the three main functional types of operations (Type 1, Type 2 and Type 3). In addition, clients have the possibility to optionally include some extra non-functional attributes to their ordered products (i.e. special coating or painting, special package, etc.) by selecting one option from Type 4 and/or one option from Type 5 (Fig. 1).

As an illustrative example, let´s assume that there are three customers and each one orders one product J_i ($n = 3$, $i = 1 \ldots n$). The set of jobs to be produced daily is

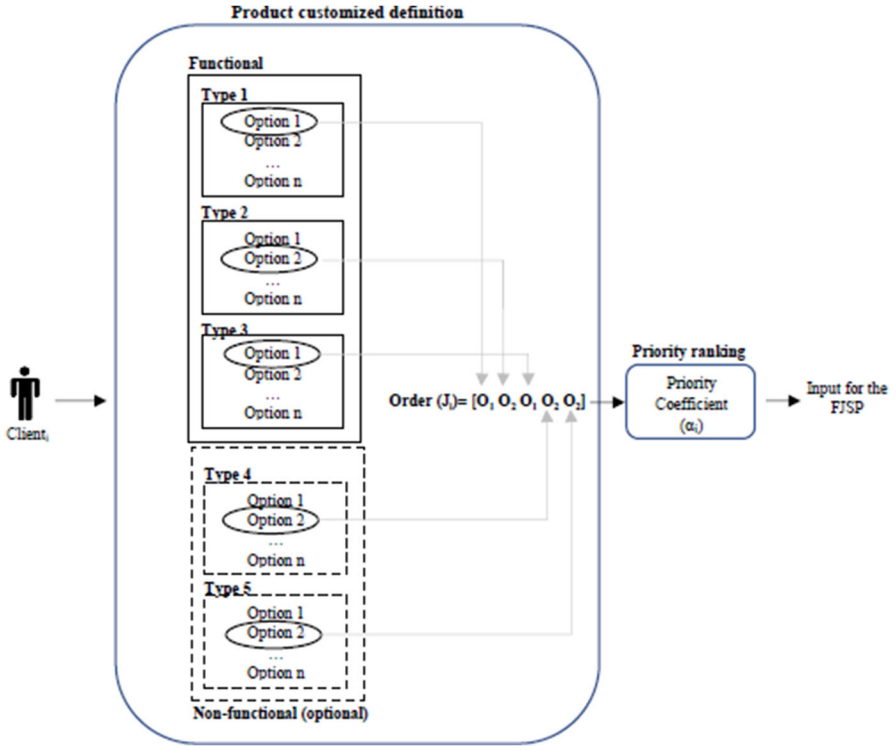

Fig. 1 Order customization process

represented as C ($|C| = n$). The clients will customize their orders by choosing one option of each functional type and, optionally, selecting one or more options from the non-functional types.

Each type option is encapsulated as a manufacturing operation which can be handled by one or more different machines in a certain amount of time (Table 1). The orders J_i will consist of n_i operations obtained from the client customized selection phase. Hereafter, the following notation will be used: if Client 2 selects the 7th, 4th, 5th and 3rd options of Types 1, 2, 3 and 4 respectively, then the operations that make up the order are $J_2 = [7\ 4\ 5\ 3\ 0]$. Similarly, the operations of orders $J_1 = [1\ 3\ 1\ 0\ 0]$ and $J_3 = [4\ 2\ 7\ 0\ 0]$ are shown in Table 2. The data reflected in Table 2 would be the input of the FJSP. The processing times shown in Tables 1 and 2 are based on a FJSP instance from Rajkumar et al. (2011).

Once received and defined, the orders should be evaluated in order to know their priority. This process allows us to increase the customer satisfaction by considering different preferences in the scheduling. Many aspects can be used to rank the orders (Güçdemir and Selim 2017) and, in this study, the same variables (x_k) and weights (w_k) used in Yao et al. (2017) are applied: the importance (x_1, $w_1 = 3$), due date (x_2, $w_2 = 5$) and revenue (x_3, $w_3 = 2$) of the orders. The priority of each order is obtained as the weighted sum average of these three metrics (Eq. 1).

Table 2 Example 3 orders and 8 machines

Job	Operación	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
(J_1)	O_{11}	5	–	7	–	3	–	–	9
	O_{12}	1	–	–	6	–	10	–	7
	O_{13}	–	–	7	4	–	–	6	–
(J_2)	O_{21}	10	–	9	–	4	7	–	–
	O_{22}	–	1	6	–	–	–	8	–
	O_{23}	11	–	9	–	–	–	6	4
	O_{24}	2	8	–	9	–	–	–	–
(J_3)	O_{31}	4	–	3	–	8	–	9	–
	O_{32}	–	5	6	4	–	9	–	–
	O_{33}	5	–	2	–	–	–	3	–

Table 3 Order ranking

Order	Importance (x_1)	Due date ($1/x_2$)	Revenue (x_3)	Pr_i	α_i
J_1	1	1	0.5	0.9	0.45
J_2	1	0.5	0.8	0.71	0.36
J_3	0.6	0.25	0.4	0.385	0.19

$$Pr_i = \frac{\sum_{k=1}^{3} w_k f\left(x_k^i\right)}{\sum_{k=1}^{3} w_k}, \quad \forall i \in C \tag{1}$$

$$\alpha_i = \frac{Pr_i}{\sum_{l \in C} Pr_l}, \quad \forall i \in C \tag{2}$$

In Eq. (1), Pr_i and x_k^i represent the priority and the value of an input variable k of an order i, respectively. Similar to Yao et al. (2017), we chose as membership function for the importance and revenue the linear function $f\ (x_k) = x_k$, and $f\ (x_k) = 1/x_k$ for the due date. Table 3 shows the values of the input variables (x_k), the order priorities (Pr_i) and the coefficients of priority (α_i) for the three orders of the 3×8 illustrative example (Table 2).

The next step to order composition and priority ranking would be production scheduling in the flexible manufacturing system. In the FJSP, there are m machines and n jobs, consisting each job J_i ($1 \leq i \leq n$) of a sequence of n_i operations. An operation O_{ij} ($i = 1,2, \ldots, n; j = 1, 2, \ldots, n_i$) of job J_i has to be processed on a machine m_{ij} out of a set of compatible machines M_{ij}. P_{ijk} denotes the processing time that a machine kM_{ij} requires to perform an operation O_{ij}, and C_{ij} its completion time.

Although some existing FJSP studies are focused on the satisfaction of one single objective, in this work multiple objectives are considered simultaneously since in rare situations the scheduling decisions of a company are based on a single performance indicator, and it usually exists a trade-off between production costs and service levels. In this case, we consider the following criteria: the completion time of each order (C_i) according to its priority, the maximal machine workload (W_m) and the total workload of the machines (W_t). In Eq. 3, it can be noticed that the first objective considers the

completion time of each order (C_i) together with its priority (α_i), differently to other approaches which aims to minimize the makespan C_{max} (i.e. Rajkumar et al. 2011; Karthikeyan et al. 2014).

In the FSP model, the following assumptions are made: an operation cannot be interrupted during its performance (h_1); each machine can perform at most one operation at any time (h_2); jobs are independent from each other (h_3); machines are independent from each other (h_4); setup and transfer time are included in P_{ijk} (h_5); and the precedence constraints are defined for any pair of operations in a job (h_6). All these assumptions and constraints, together with the three objective functions are reflected in the following mathematical model:

$$\min f_1 = \sum_{i=1}^{n} \alpha_i C_i \tag{3}$$

$$\min f_2 = max_{1 \leq k \leq m}(W_k) \tag{4}$$

$$\min f_3 = \sum_{k=1}^{m} W_k \tag{5}$$

subject to:

$$C_{ij} - C_{i(j-1)} \geq P_{ijk} X_{ijk}, \quad j = 2, \ldots, n_i; \forall i, k \tag{6}$$

$$[(C_{hg} - C_{ij} - P_{hgk}) X_{hgk} X_{ijk} \geq 0] \mathrm{V} [(C_{ij} - C_{hg} - P_{ijk}) X_{hgk} X_{ijk} \geq 0], \quad \forall i, j, h, g, k \tag{7}$$

$$\sum_{k \in M_{ij}} X_{ijk} = 1, \quad \forall i, j \tag{8}$$

$$X_{ijk} \in \{0, 1\}, \quad \forall i, j, k \tag{9}$$

Equations (3), (4) and (5) ensure the minimization of the objectives. Inequalities (6) and (7) ensure constraints h_6 and h_2, respectively. Equation (8) indicates that only one machine could be assigned to each operation. Equation (9) is the binary decision variable ("1" when machine k processes operation O_{ij} and "0" otherwise).

The overall objective function uses one of the three types of multi-objective optimization methods presented in Hsu et al. (2002): the weighted sum of the objective values (Eq. 10). Since in this case the range and the unit of measurement of the objectives functions do not differ significantly (the three objectives are time variables ranging within similar value intervals), it is possible to directly sum the objective functions without occurring in a non-standardized error which would lead to unrealistic scheduling solutions. If the values of one or more objective functions differ in their range and/or unit of measurement, a normalization procedure (Akbaripour et al. 2018) or the analytical hierarchy process (Cao et al. 2016) should be conducted.

$$\min F(x) = w_1 f_1 + w_2 f_2 + w_3 f_3 \tag{10}$$

subject to:

$$w_1 + w_2 + w_3 = 1, w_i \in [0, 1]. \tag{11}$$

Equation (11) represents the weight coefficients for each objective value. In this case, we give the largest weight to minimizing the completion times of the orders in order to satisfy the customer needs, but we also consider the workload criteria (W_m, W_t) because an overemphasis on due dates might excessively burden the plant resources (Morinaga et al. 2014).

4 Multi-objective discrete firefly algorithm for the FJSP

The firefly algorithm (FA) is a swarm intelligence algorithm proposed by Xin-She Yan in 2008. It was motivated by the social behaviour of the fireflies and how they attract others using their flashing lights.

This novel algorithm was originally proposed to solve continuous optimization problems and it was inspired by the simulation of a special characteristic of the fireflies: the luminescence. The main idea is to associate the light intensity of the fireflies with the encoded objective function. Initially, fireflies are spread over the search space and their positions represent solutions of the optimization problem. At each iteration fireflies will move towards the brighter ones according to their attractiveness. It is known (Yang, 2009) that light intensity (I), and hence the attraction, decreases with the distance ($I \alpha 1/r^2$). It makes that fireflies are only visible to others at a certain distance, and thus fireflies will move towards the brighter ones within the region of the search space in which they are positioned. This movement of all fireflies towards the most attractive ones and the use of a random movement component for avoiding local optimum stagnation and premature convergence make the FA an algorithm with a promising capacity of exploration. Algorithm 1 shows the basic structure of the original FA.

Algorithm 1 Original FA pseudocode

```
1   Initialize the population P at different locations x_i of the search space
2   Calculate the light intensity of each firefly I_i according to the objective function f(x_i)
3   while TerminationCriteria not met:
4       for each firefly i ∈ P:
5           for each firefly j ∈ P:
6               if (I_i < I_j):
7                   Move firefly i towards firefly j according to the attractiveness β(r_ij)
8               end
9               Update light intensity at new locations
10          end
11      end
12      Evaluate fireflies at final locations and get the global best
13  end
```

The FA is a bio-inspired algorithm and it was required to simplify and idealize the behaviour of the fireflies before using it for solving optimization problems (Yang 2008): (1) the attraction between them will not depend on their sex, (2) the attraction will be proportional to the light intensity. Hence, if there is a difference in their brightness, the less bright firefly will move towards the brighter one, being the movement random otherwise; (3) the light intensity is given by the objective function.

Since the FA was very successful is finding (near-)optimal solutions in optimization problems, and experimental results have shown that it can get better results than other metaheuristics such as the PSO and the GA (Yang 2008; Lohrer 2013), it is worth to study the application of the FA to solve the FJSP. But for this purpose, a discrete version should be developed. In the rest of this section, the main issues of the original version of the FA and the biggest challenges of its discretization are explained. The original aspects, definitions and formulas of the algorithm were taken from the studies of Yang (2008, 2009, 2010), and the discrete approach is inspired in Karthikeyan et al. (2014) and Lunardi and Voos (2018).

4.1 Attractiveness

In the FA, the objective function defines the brightness (I) of a firefly at a certain position. A position x represents a possible solution of the optimization problem and, for minimization problems, the objective function value of the solution is inversely associated to the light intensity of the firefly $I(x) \propto 1/f(x)$.

As the second idealized rule of the FA states, the attractiveness (β) is proportional to the brightness. Nevertheless, the brightness is relative: two fireflies at different distances (r) from the light source (I_s) will experiment a different attraction. This is because the light intensity varies with the inverse square law of the distance $I(r)=I_s/r^2$, decreasing as the distance from the source increases. To avoid the singularity at $r = 0$ of the inverse square law, the original FA uses the Gaussian form represented in Eq. 12.

$$I(r) = I_0 e^{-\gamma r^2} \tag{12}$$

In addition, the variation of the light intensity (and attractiveness) also depends on the media in which the light is propagated. Different media have different degrees of light absorption, and that is the reason why the light absorption coefficient γ is introduced in the variation of the light intensity (Eq. 12) and in the formulation of the attractiveness (Eq. 13).

$$\beta = \beta_0 e^{-\gamma r^2} \tag{13}$$

In Eq. 13, β_0 stands for the attractiveness at $r = 0$. Approximating the exponential form of Eq. 13 to $1/(1 + r^2)$ allows to speed up the calculations (Yang, 2008), and it gives the following final formulation for the attractiveness (Eq. 14), which is the one used in the MO-DFFA and has an important role in the movement:

AV	1	1	4	1	2	3	1	3	2	3
	O_{11}	O_{12}	O_{13}	O_{21}	O_{22}	O_{23}	O_{24}	O_{31}	O_{32}	O_{33}

SV	1	1	3	2	3	3	1	2	2	2
	O_{11}	O_{12}	O_{31}	O_{21}	O_{32}	O_{33}	O_{13}	O_{22}	O_{23}	O_{24}

Solution:

(O_{11},M_1) (O_{12},M_1) (O_{31},M_3) (O_{21},M_1) (O_{32},M_2) (O_{33},M_3) (O_{13},M_4) (O_{22},M_2) (O_{23},M_3) (O_{24},M_1)

Fig. 2 Representation of the decoded solution

$$\beta = \frac{\beta_0}{1 + \gamma r^2} \tag{14}$$

4.2 Solution codification and initialization

One of the main modifications required to adapt the original version of the FA to the FJSP was the solution representation and its decoding into a feasible schedule. In this case, each firefly position will represent one possible solution of the problem, and it will be obtained from two different solution components.

The first component will be the assignment component and it will refer to the machines m_{ij} selected to perform each operation O_{ij}. The second component will be the sequencing component and it indicates the order in which the operations will be produced in the selected machines. With this two-component solution representation the two sub-problems of the FJSP are optimized simultaneously, unlike other hierarchical approaches in which they are conducted separately, one after the other (Fattahi et al. 2007).

Therefore, the MO-DFFA solution will contain two vectors, one for each subproblem with length equal to the total number of operations to be produced, $\sum_{i=1}^{n} n_i$. The assignment vector (AV) denotes the machine selected for each operation, where an item AV[i] stores a machine index. The sequencing vector (SV) represents the sequence of the operations. In the SV each job number J_i appears n_i times. Figure 2 shows how a firefly solution can be decoded from its AV and SV.

The decoding procedure of the solution components into a feasible schedule is the following. First, the sequence is obtained looking at the SV. The jth time that a job number J_i appears in the SV will refer to the jth operation of J_i (O_{ij}). From the SV shown in Fig. 2 the following sequence is obtained $\{O_{11}, O_{12}, O_{31}, O_{21}, O_{32}, O_{33}, O_{13}, O_{22}, O_{23}, O_{24}\}$. Then, the AV indicates the machines m_{ij} selected for each operation O_{ij}: $\{(O_{11}, M_1), (O_{12}, M_1), (O_{13}, M_4), (O_{21}, M_1), (O_{22}, M_2), (O_{23}, M_3), (O_{24}, M_1), (O_{31}, M_3), (O_{32}, M_2), (O_{33}, M_3)\}$. Combining the information of both vectors the final solution of a firefly can be decoded: $\{(O_{11}, M_1), (O_{12}, M_1), (O_{31}, M_3), (O_{21}, M_1), (O_{32}, M_2), (O_{33}, M_3), (O_{13}, M_4), (O_{22}, M_2), (O_{23}, M_3), (O_{24}, M_1)\}$. Once the sequence and assignment of the operations are obtained, they are allocated over time defining the initial and end time of each operation on each machine satisfying the constraints of the model described in Sect. 3. Finally, the completion times and the workload are calculated, and the objective value of each solution is obtained (Eq. 10), which will determine the brightness of each firefly (and attractiveness).

To ensure the quality of the initial population and enhance the performance of the MO-DFFA, both the AVs and the SVs of the initial solutions are generated follow-

Table 4 Movement steps for AV and SV

	Assignment vector	Sequencing vector
P	1 1 4 3 2 1 1 3 2 3	2 3 1 2 1 3 1 2 2 3
P_{best}	1 1 4 1 2 3 1 3 2 3	1 1 3 2 3 3 1 2 2 2
d_{av} and d_{sv}	{(4,1), (6,3)}	{(1,3), (3,10), (2,5)}
$\|d_{av}\|$ and $\|d_{sv}\|$	2	3
$\beta(r)$	0.71	0.53
rand $\in$ [0,1]	{0.13, 0.88}	{0.25, 0.64, 0.48}
Mov. β-step	{(4,1)}	{(1,3), (2,5)}
Position after β-step	1 1 4 1 2 1 1 3 2 3	1 1 2 2 3 3 1 2 2 3
Position after α -step	1 1 4 1 2 1 1 3 3 3	1 1 2 2 3 3 2 1 2 3

ing some specific rules that initially locate the fireflies at promising regions of the search space. These rules are: for AV components, initialization is conducted using the AssignmentRule2 (R_{av1}) presented in Pezzella et al. (2008), which in turn is based on the *approach by localization* introduced by Kacem et al. (2002b). This approach considers at the same time the utilization of the machines and the processing times. For the SV components, some are initiated randomly (R_{sv1}) and the rest according to the priorities obtained from the ranking process explained in Sect. 3 (R_{sv2}).

4.3 Distance between fireflies

The original version of the FA uses the Cartesian distance to calculate the distance (r_{ij}) between the position x_i of a firefly i and the position x_j of a firefly j (Eq. 15).

$$r_{ij} = \|x_i - x_j\| = \sqrt{\sum_{k=1}^{d} (x_{i,k} - x_{j,k})^2} \tag{15}$$

In Eq. 15, $x_{i,k}$ refers to the kth component of the spatial coordinate x_i of the ith firefly, and d represents the number of dimensions. Nevertheless, in the proposed MO-DFFA, the distance measurement cannot be carried out using the Cartesian distance since the solutions are not locations of the search space with defined spatial coordinates, but are solutions made up of two vectors with specific characteristics.

In the MO-DFFA, measuring the distance between two fireflies is worked out with two different methods. For AV components the distance is carried out with the Hamming distance ($|d_{av}|$), which is the number of non-corresponding elements in the sequence. Table 4 shows the Hamming distance between the AVs of a firefly (P) and a brighter one (P_{best}). The non-corresponding elements are $AV_P[4]$ and $AV_P[6]$, and thus the distance between the two fireflies is $|d_{av}| = 2$.

On the other hand, the distance between the SVs of two fireflies can be measured by the number of required swaps of the first solution to get the SV of the brighter one ($|d_{sv}|$). From Table 4 it can be seen how 3 different permutations in SV_P are required to obtain the SV of P_{best} ($|d_{sv}| = 3$): position 1 with position 3; position 3 with position

10; and position 2 with position 5. Once the distance between fireflies is calculated, the next step is the movement (Algorithm 1).

4.4 Movement

In the original FA, the equation that represents the movement of a firefly i towards a brighter and more attractive firefly j is:

$$x_i = x_i + \beta_0 e^{-\gamma r_{ij}^2}\left(x_j - x_i\right) + \alpha\left(rand - \frac{1}{2}\right) \tag{16}$$

The second term of Eq. 16 represents the movement of a firefly induced by the attraction. The last term is the random component of the movement where *rand* is a random number obtained from a uniform distribution between [0,1] and α is the randomization parameter.

As it was required for the solution representation and the measurement of the distance, a different method for the movement of the fireflies should be defined for the discrete approach. In the MO-DFFA the movement of the original FA is broken up into two not interchangeable steps: first the β-step (Eq. 17), and then the α-step (Eq. 18). Table 4 illustrates the movement of a firefly towards the global best with $\beta_0 = 1$, $\gamma = 0.1$ and $\alpha = 1$.

$$x_i = \beta(r)\left(x_j - x_i\right) \tag{17}$$

$$x_i = x_i + \alpha\left(rand - \frac{1}{2}\right) \tag{18}$$

The β-step is used to bring the AV and SV of a firefly closer to the global best. For the AV, it is an insertion mechanism. For the SV, it consists in a pair-wise exchange. The β-step is made up of the following sub-steps: all necessary insertions in AV and all pair-wise exchanges in SV are found and stored in d_{av} and d_{sv}, respectively; the distances $|d_{av}|$ and $|d_{sv}|$ are stored in r; the probability $\beta(r) = \beta_o/(1 + \gamma r^2)$ is computed; a random number $rand \in [0,1]$ is generated for each element of d_{av} and d_{sv}; and then the corresponding insertion/pair-wise exchange is performed if $rand \leq \beta$ (Table 4).

The α-step allows us to shift the permutation into one of the neighboring permutations. To do it, we approximate the term $\alpha(rand - ½)$ of Eq. 18 to $\alpha(rand_{int})$, where $rand_{int} \in [0, \sum_{i=1}^{n} n_i]$. For the SV, this α-step consists in randomly choosing two non-equal element positions and swapping them. For the AVs, the α-step consists in selecting a random position of the vector and assigning a different eligible machine to the corresponding operation. The whole procedure followed in the MO-DFFA is summarized in Algorithm 2.

Algorithm 2 MO-DFFA pseudocode

```
Initialize the two solutions components AV_i, SV_i of each firefly i ∈ P
Calculate the light intensities I_i according to the overall objective function F(AV_i, SV_i)
while TerminationCriteria not met:
    for each firefly i ∈ P:
        for each firefly j ∈ P:
            if I_i < I_j:
                Find the required insertions/swaps (d_av, d_sv) and distances (|d_av|, |d_sv|)
                for each solution component c in [AV, SV]:
                    Calculate the attractiveness β(|d_c|)
                    for each element in d_c:        # β-step of the movement #
                        if rand[0,1] ≤ β(|d_c|):
                            Perform the corresponding insertion (if c=AV) or swap (if c=SV)
                        end
                    end
```

Table 5 Computational results

Comparison results						Proposed problem				
Instances[a]		f_1	f_2	f_3	$F(c)$	Instances[b]	f_1	f_2	f_3	$F(c)$
8×5	GA	27	27	109	43.4	3×4	15.72	18	44	19.01
	GRASP	24	24	101	39.4	6×6	16.44	16	70	21.71
	MO-DFFA	24	24	101	39.4	6×8	12.66	10	60	16.86
12×5	GA	33	33	145	55.5	8×6	19.92	20	99	27.85
	GRASP	33	33	138	54	8×8	13.31	15	80	20.32
	MO-DFFA	32	32	141	53.8					

[a] $w_1 = 0.5$, $w_2 = 0.3$, $w_3 = 0.2$, [b] $w_1 = 0.7$, $w_2 = 0.2$, $w_3 = 0.1$

5 Computational results and analysis

In order to test the performance of the proposed MO-DFFA, we conducted two different computational experiments. First, we evaluated the algorithm in two classical FJSP instances taken from Du et al. (2008). For these two instances (problems 8×5 and 12×5), the objectives to be minimised were the maximal completion time or makespan, the maximal machine workload and the total workload (f_1, f_2 and f_3, respectively). The aim of this first experiment was to compare the performance of the proposed algorithm with other state-of-the-art algorithms: the GA proposed in Du et al. (2008) and the greedy randomised adaptive search procedure (GRASP) proposed in Rajkumar et al. (2011). From the results shown in Table 5, it can be noticed how the proposed MO-DFFA obtained better results than the GA in the 8×5 instance, and outperformed both the GA and GRASP in the 12×5 instance.

Once the effectiveness of the MO-DFFA was validated by comparison, we conducted a second experiment in which it was applied to solve five instances of the problem proposed in this paper. Table 5 shows the best results obtained from 30 runs of the algorithm, together with the weights used for the objectives functions in both experiments. The MO-DFFA was implemented in Matlab 2013 on an Intel Core 7 2.1 GHz PC with 8 GB RAM memory.

Following the notation introduced in Sect. 3, the operations that make up each order and its priority coefficients are: for problem 3×4, $J_1 = [1\ 3\ 1\ 0\ 0]$, $J_2 = [7\ 4\ 5\ 3\ 0]$, $J_3 = [4\ 2\ 7\ 0\ 0]$, $\alpha_i = [0.45\ 0.36\ 0.19]$; for problems 6×6 and 6×8, $J_1 = [1\ 2\ 5\ 0\ 1]$, $J_2 = [7\ 3\ 7\ 0\ 0]$, $J_3 = [4\ 2\ 4\ 0\ 0]$, $J_4 = [5\ 7\ 1\ 3\ 0]$, $J_5 = [7\ 4\ 5\ 3\ 0]$, $J_6 = [3\ 5\ 3\ 0\ 0]$, $\alpha_i = [0.35\ 0.25\ 0.14\ 0.11\ 0.1\ 0.05]$; and for problems 8×6 and 8×8, $J_1 = [1\ 1\ 1\ 0\ 1]$, $J_2 = [2\ 2\ 2\ 0\ 0]$, $J_3 = [3\ 3\ 3\ 0\ 0]$, $J_4 = [4\ 4\ 4\ 3\ 0]$, $J_5 = [5\ 5\ 5\ 3\ 0]$, $J_6 = [6\ 6\ 6\ 0\ 0]$, $J_7 = [7\ 7\ 7\ 0\ 2]$, $J_8 = [4\ 2\ 6\ 0\ 0]$, $\alpha_i = [0.33\ 0.24\ 0.13\ 0.1\ 0.1\ 0.04\ 0.01\ 0.01]$. The P_{ijk} of each instance are taken from Table 1, as it was explained for the illustrative example of Sect. 3. The parameters of the MO-DFFA are the following: population $P = 200$, $R_{sv1} = 80\%$, $R_{sv2} = 20\%$, $max_{gen} = 50$, $\beta_o = 1$, $\alpha = 1$ and $\gamma = 0.1$.

Finally, it is worth to highlight how in this paper the FJSP model is adapted to better suit the requirements of a manufacturing system in which flexibility and customer centricity are of critical importance. Many multi-objective FJSP approaches aim to

minimize the makespan (C_{max}), together with other manufacturing indicators such as the maximal and/or the total machine workload (i.e. Rajkumar et al. 2011; Karthikeyan et al. 2014). Nevertheless, since the makespan tries to minimize the maximal completion time of all the jobs, this kind of approaches consider the completion time of each order as important as the completion time of the others, and this *likewise*-treatment might not be the best way to guide the scheduling process in a customer-oriented manufacturing system. This is the reason why, in the proposed model, we introduced the coefficients of priority α_i (see Eq. 3). The introduction of these coefficients in the overall objective function allows us to give a different preference to the completion time of each order, and thus it helps to find schedules that can increase the client satisfaction.

To illustrate the differences between the classical approaches that tries to minimize the makespan and the one proposed in this paper, we will use as reference the 8×5 instance with 20 operations from Du et al. (2008). Table 6 shows the processing times and the job operations of this instance.

Figure 3 depicts the Gantt chart of the best solution obtained for the 8×5 instance, being the three objectives C_{max}, W_{tot} and W_{max}, and the weights $w_1 = 0.5$, $w_2 = 0.3$ and $w_3 = 0.1$ respectively. Since the makespan (C_{max}) is one of the objectives that guides the search, all the orders are equally *push-to-the-left* to find a schedule in which the maximal completion time is as lower as possible, minimizing then the overall objective function. But, as mentioned before, if specific due dates or different client importance exists, this equal treatment of the orders might get to worse schedules than the ones obtained with a customer-centric approach as the one presented in this study.

Following the framework and the model introduced in Sect. 3, let's assume that orders are already defined (for comparison, with the same operations as the jobs of instance 8×5), and the customer orders are ranked by priority according to their importance (x_1, $w_1 = 3$), due date (x_2, $w_2 = 5$) and revenue (x_3, $w_3 = 2$). These three metrics are the same used in Yao et al. (2017), but a deeper analysis of the clients can be done looking at other aspects as, for example, the customer segments, the customer expectations or the customer penalties (see Güçdemir and Selim 2017).

If after the ranking some orders obtain a much greater Pr value than the others (i.e. because they come from clients that are more important for the company and/or have tighter dues dates), it would be necessary to integrate this information in the scheduling model, as it is done here with the coefficients of priority.

Table 7 shows the coefficients of priority for an instance 8×5a, in which orders J_1 and J_2 are considered significantly more important than the others, and an instance 8×5b, in which the same happens but for orders J_6 and J_7. The order priorities Pr_i and coefficients of priority α_i are calculated using the Eqs. 1 and 2, respectively.

If the makespan is one of the objectives, the priority information is not taken into account and all the orders are scheduled in such a way that the overall completion time is minimized. This might affect the orders with higher priority and, hence, the customer satisfaction. From Fig. 3, the completion times when considering C_{max} are $C_1 = 16$, $C_2 = 9$, $C_6 = 24$ and $C_7 = 13$.

But, if the priority information is considered using the coefficients of priority, the scheduling can be guided to find schedules in which the completion times of the more important orders have more weight than the rest (proportionally to their importance). In Fig. 4, it can be seen how, when using the priority information via the coefficients

Table 6 Instance 8×5 with 20 operations from Du et al. (2008)

	Machines				
	1	2	3	4	5
J_1					
O_{11}	5	3	–	–	–
O_{12}	–	7	–	–	–
J_2					
O_{21}	–	–	6	–	–
O_{22}	–	–	–	3	4
J_3					
O_{31}	–	4	6	–	–
O_{32}	7	–	–	–	–
O_{33}	–	–	–	7	–
J_4					
O_{41}	–	–	–	–	10
J_5					
O_{51}	–	–	–	5	–
O_{52}	4	5	8	–	–
O_{53}	–	–	–	6	5
O_{54}	–	3	–	–	4
J_6					
O_{61}	–	2	6	–	–
O_{62}	–	–	8	–	–
J_7					
O_{71}	–	–	3	8	–
O_{72}	–	–	–	7	4
J_8					
O_{81}	3	–	5	–	–
O_{82}	–	–	–	9	6
O_{83}	–	–	7	–	–
O_{84}	–	3	–	–	–

α_i, the completion time of order J_1 is reduced from $C_1 = 16$ (8×5 with C_{max}) to $C_1 = 10$ (8×5a using priority information). Logically, giving more priority to some orders may harm other orders (for example, C_8 goes from 16 to 30 and C_3 from 23 to 30), but the overall schedule is better since J_1 was ranked as much more important than J_8 and J_3 (see Table 7). This *trade-off* can be controlled in the ranking phase adjusting the values x_1, x_2 and x_3 at its weights w_1, w_2 and w_3.

Similarly, looking at the Gantt chart of the 8×5b instance (Fig. 5), using the proposed model the completion times of J_6 and J_7 are reduced from $C_6 = 24$ to $C_6 = 11$ and from $C_7 = 13$ to $C_7 = 7$, since those two orders are the ones with higher priority (see Table 7). It can be also noticed that in both instances 8×5a and 8×5b, as in the original 8×5 instance, the machines workload is balanced by considering W_{tot}

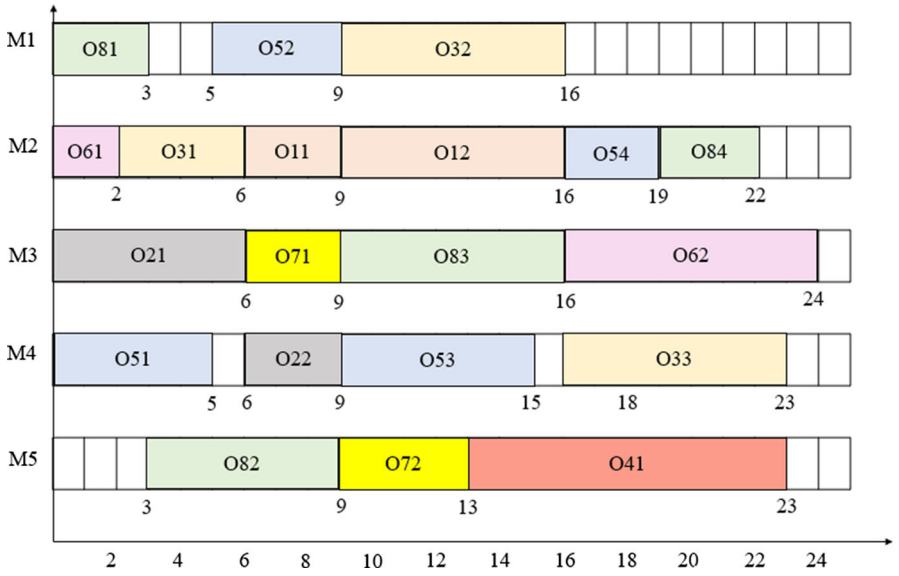

Fig. 3 Gantt chart 8×5 with 20 operations optimizing the makespan

and W_{max} in the overall objective function, since both system and customer interests should be granted.

Some existing approaches do something similar using the weighted tardiness or lateness as objectives (Morinaga et al. 2014), aiming to satisfy the clients due dates, but they do not consider other customer attributes such as their importance or potential revenue, as it is done in the proposed model which also allows to introduce as many variables as necessary in the ranking phase.

6 Conclusion

In this work, a discrete version of the Firefly Algorithm has been proposed for solving the Flexible Job-Shop Scheduling Problem, an important and well-studied NP-hard optimization problem. We introduced a framework that allows product customization and customer-centric manufacturing, aiming to bring the classical FJSP closer to the Industry 4.0 paradigm. Clients can define the characteristics of their orders according to their requirements by choosing between different types of product attributes. These specific characteristics are encapsulated as manufacturing operations that can be handled by some compatible machines in a certain amount of time. These would be the operations to be scheduled and thus the input of the FJSP. In addition, in order to enhance customer satisfaction, orders are ranked so that the ones classified as more important can be scheduled with higher priority. Since it was considered important to guarantee customer satisfaction while taking care about company resources utilization, three objectives where optimised simultaneously: the completion time of the orders, the maximal machine workload and the total workload of the machines.

Table 7 Order ranking for 8 × 5a and 8 × 5b instances

Order	Importance (x_1)	Due date ($1/x_2$)	Revenue (x_3)	Pr	α_i
8 × 5a					
J_1	1	1	0.9	0.98	0.35
J_2	0.9	0.9	0.8	0.88	0.31
J_3	0.3	0.1	0.3	0.2	0.07
J_4	0.3	0.1	0.4	0.22	0.08
J_5	0.3	0.1	0.1	0.16	0.06
J_6	0.1	0.2	0.1	0.15	0.05
J_7	0.1	0.1	0.2	0.12	0.04
J_8	0.1	0.1	0.2	0.12	0.04
8 × 5b					
J_1	0.2	0.2	0.1	0.18	0.06
J_2	0.3	0.1	0.2	0.18	0.06
J_3	0.3	0.1	0.3	0.2	0.07
J_4	0.4	0.1	0.4	0.25	0.09
J_5	0.3	0.1	0.1	0.16	0.05
J_6	0.8	0.9	0.9	0.87	0.30
J_7	0.9	1	0.8	0.93	0.32
J_8	0.1	0.2	0.1	0.15	0.05

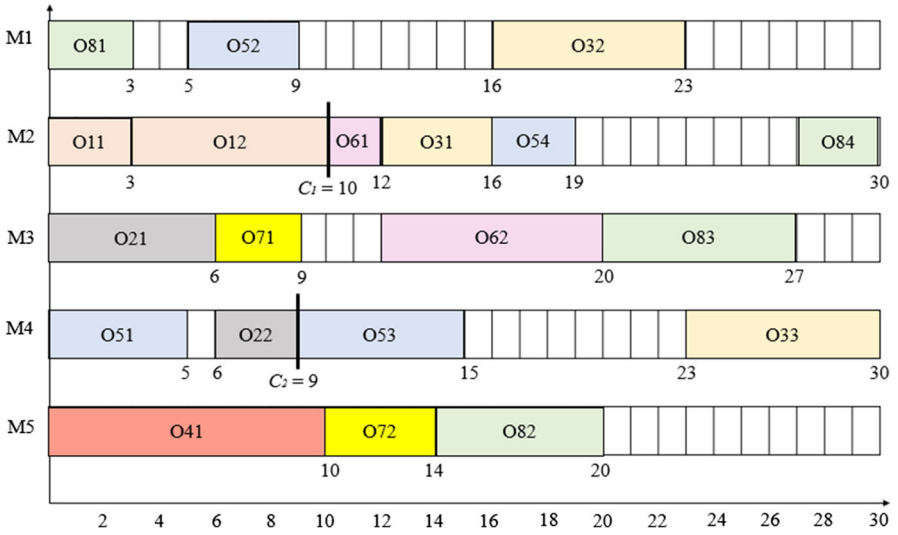

Fig. 4 Gantt chart 8 × 5a instance with 20 operations

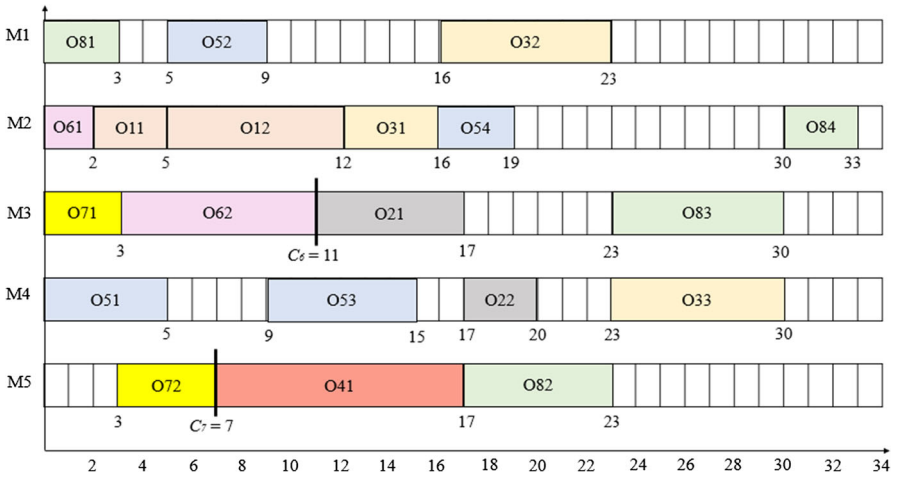

Fig. 5 Gantt chart 8 × 5b instance with 20 operations

The method selected for solving the problem was the Firefly Algorithm, a swarm intelligence algorithm motivated by some behavioural patterns of the fireflies. The main aspects of the original version of the algorithm were introduced while, at the same time, the challenges of its discretization, required to be applied to the FJSP, have been explained. Once the discrete approach was introduced, its performance and efficiency were evaluated by comparison with other existing metaheuristic algorithms in solving some FJSP instances, and then it was applied for solving several instances of the model presented in this study.

Although in this work and in other studies the FA was demonstrated to be a very efficient and successful algorithm which has promising capacity of exploration, some more research should be carried out to further analyse the potential and possible limitations of this algorithm. Next steps would be focused on studying the scalability of the MO-DFFA and its possible combinations with other local search procedures that can lead to a hybrid version with a better success rate and efficiency.

References

Akbaripour H, Houshmand M, van Woensel T, Mutlu N (2018) Cloud manufacturing service selection optimization and scheduling with transportation consideration: mixed-integer programming models. Int J Adv Manuf Technol 95:43–70

Bagheri A, Zandieh M, Mahdavi I, Yazdani M (2010) An artificial immune algorithm for the flexible job-shop scheduling problem. Future Gener Comput Syst 26(4):533–541

Banati H, Bajaj M (2011) Fire fly based feature selection approach. Int J Comput Sci Issues 8(4):473–479

Biskup D (1998) Single-machine scheduling with learning considerations. Eur J Oper Res 115:173–178

Blum C, Roli A (2003) Meta-heuristics in combinatorial optimisation: overview and conceptual comparison. ACM Comput Surv 35(3):268–308

Brandimarte P (1993) Routing and scheduling in a flexible job shop by tabu search. Ann Oper Res 41:157–183

Brettel M, Friederichsen N, Keller M, Rosenberg M (2014) How virtualization, decentralization and network building change the manufacturing landscape: an Industry 4.0 perspective. World Acad Sci Eng Technol Int J Mech Aerosp Ind Mech Manuf Eng 8:37–44

Brucker P, Jurisch B, Sievers B (1994) A branch and bound algorithm for the job-shop scheduling problem. Discrete Appl Math 49:107–127

Brucker P (2001) Scheduling algorithms. Springer, Berlin

Cao Y, Wang S, Kang L, Gao Y (2016) A TQCS-based service selection and scheduling strategy in cloud manufacturing. Int J Adv Manuf Technol 82:235–251

Chatterjee A, Mahanti GK, Chatterjee A (2012) Design of a fully digital controlled reconfigurable switched beam concentric ring array antenna using firefly and particle swarm optimization algorithm. Prog Electromagn Res B 36:113–131

Chen CL, Vempati VS, Aljaber N (1995) An application of genetic algorithms for flow shop problems. Eur J Oper Res 80(2):389–396

Dauzère-Pérès S, Paulli J (1997) An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. Ann Oper Res 70:281–306

De Giovanni L, Pezzella F (2010) An improved genetic algorithm for the distributed and flexible job-shop scheduling problem. Eur J Oper Res 200(2):395–408

De Paula MR, Ravetti MG, Mateus GR, Pardalos PM (2007) Solving parallel machines scheduling problems with sequence-dependent setup times using variable neighbourhood search. IMA J Manag Math 18(2):101–115

Du X, Li Z, Xiong W (2008) Flexible Job shop scheduling problem solving based on genetic algorithm with model constraints. In: IEEE international conference on industrial engineering and engineering management (IEEM 2008), pp 1239–1243

Fattahi P, Saidi-Mehrabad M, Jolai F (2007) Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. J Intell Manuf 18(3):331–342

Garey MR, Johnson DS, Sethi R (1976) The complexity of flowshop and jobshop scheduling. Math Oper Res 1(2):97–196

Gonçalves JF, Mendes JJM, Resende MGC (2005) A hybrid genetic algorithm for the job shop scheduling problem. Eur J Oper Res 167(1):77–95

Graves SC (1981) A review of production scheduling. Oper Res 29(4):646–675

Güçdemir H, Selim H (2017) Customer centric production planning and control in jobs shops: a simulation optimization approach. J Manuf Syst 43:100–116

Horng MH (2012) Vector quantization using the firefly algorithm for image compression. Expert Syst Appl 39(1):1078–1091

Hsu T, Dupas R, Jolly D, Goncalves G (2002) Evaluation of mutation heuristics for solving a multiobjective flexible job shop by an evolutionary algorithm. In: Proceeding of the IEEE international conference on systems, man and cybernetics, pp 655–660

Ivanov D, Dolgui A, Sokolov B, Werner F, Ivanova M (2016) A dynamic model and an algorithm for short-term supply chain scheduling in the smart factory industry 4.0. Int J Prod Res 54(2):386–402

Kacem I, Hammadi S, Borne P (2002a) Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic. Math Comput Simul 60(3):245–276

Kacem I, Hammadi S, Borne P (2002b) Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. IEEE Trans Syst Man Cybern Part C (Appl Rev) 32(1):1–13

Karthikeyan S, Asokan P, Nickolas S (2014) A hybrid discrete firefly algorithm for multi-objective flexible job shop scheduling problem with limited resource constraints. Int J Adv Manuf Technol 72:1567–1579

Kuo WH, Yang DL (2006) Minimizing the total completion time in a single-machine scheduling problem with a time-dependent learning effect. Eur J Oper Res 174(2):1184–1190

Lawler EL, Lenstra K, Rinooy AHK, Shmoys DB (1993) Sequencing and scheduling: Algorithms and complexity. In: Graves SS, Rinnooykan AHG, Zipkin P (eds) Logistics of production and inventory. Handbooks in operations research and management science, vol 4. North-Holland Publishing Company, Amsterdam, pp 445–522

Lee CY (1991) Parallel machines scheduling with nonsimultaneous machine available time. Discrete Appl Math 30:53–61

Lohrer M (2013) A comparison between the firefly algorithm and particle swarm optimization. PhD thesis

Lunardi W, Voos H (2018) Comparative study of genetic and discrete firefly algorithm for combinatorial optimization. In: 33rd ACM/SIGAPP symposium on applied computing, At Pau, France. https://doi.org/10.1145/3167132.3167160
Mastrolilli M, Gambardella LM (2000) Effective neighbourhood functions for the flexible job shop problem. J Sched 3(1):3–20
Morinaga Y, Nagao M, Sano M (2014) Optimization of flexible job-shop scheduling with weighted tardiness and setup-worker load balance in make-to-order manufacturing. In: Joint 7th international conference on soft computing and intelligent systems (SCIS) and 15th international symposium on advanced intelligent systems (ISIS). https://doi.org/10.1109/scis-isis.2014.7044681
Moslehi G, Mahnam M (2011) A Pareto approach to multi-objective flexible job-shop scheduling problem using particle swarm optimization and local search. Int J Prod Econ 129(1):14–22
Najid NM, Dauzere-Peres S, Zaidat A (2002) A modified simulated annealing method for flexible job shop scheduling problem. In: IEEE international conference of systems, man and cybernetics, vol 5
Nasser S, Ghasemishabankareh (2013) A novel hybrid meta-heuristic algorithm for solving multi objective flexible job shop scheduling. J Manuf Syst 32(4):771–780
Osman IH, Potts CN (1989) Simulated annealing for permutation flow-shop scheduling. Omega 17(6):551–557
Paulli J (1995) A hierarchical approach for the FMS scheduling problem. Eur J Oper Res 86(1):32–42
Pezzella F, Morganti G, Ciaschetti G (2008) A genetic algorithm for the Flexible Job-shop scheduling problem. J Comput Oper Res 35(10):3202–3212
Rajkumar M, Asokan P, Anilkumar N, Page T (2011) A GRASP algorithm for flexible job-shop scheduling problem with limited resource constraints. Int J Prod Res 49:2409–2423
Rossi A, Dini G (2007) Flexible job-shop scheduling with routing flexibility and separable set up times using ant colony optimisation method. Robot Comput Integr Manu 23(5):503–516
Ruiz R, Vázquez-Rodríguez JA (2010) The hybrid flow shop scheduling problem. Eur J Oper Res 205(1):1–18
Senthilnath J, Omkar SN, Mani V (2011) Clustering using firefly algorithm: performance study. Swarm Evol Comput 1(3):164–171
Sha DY, Hsu CY (2006) A hybrid particle swarm optimization for job shop scheduling problem. Comput Ind Eng 51(4):791–808
Vaessens RJM, Aarts EHL, Lenstra JK (1994) Job shop scheduling by local search. COSOR Memorandum 94-05, Eindhoven University
Wang S, Wan J, Li D, Zhang C (2016a) Implementing smart factory of Industrie 4.0: an outlook. Int J Distrib Sens Netw 12(1):3159805
Wang S, Zhang C, Li D (2016b) A Big Data centric integrated framework and typical system configurations for smart factory. In: Wan J, Humar I, Zhang D (eds) Industrial IoT technologies and applications. Industrial IoT 2016. Lecture notes of the institute for computer sciences, social informatics and telecommunications engineering, vol 173. Springer, Cham, pp 12–23
Xing LN, Chen YW, Wang P, Zhao QS, Xiong J (2010) A knowledge-based ant colony optimization for flexible job shop scheduling problems. Appl Soft Comput 10:888–896
Yang XS (2008) Nature-inspired metaheuristic algorithm. Luniver Press, Bristol
Yang XS (2009) Firefly algorithm for multimodal optimization. Stoch Algorithms Found Appl 5792:169–178
Yang XS (2010) Firefly algorithm, stochastic test functions and design optimization. Int J Bio Inspir Comput 2(2):78–84
Yao X, Zhang J, Li Y, Zhang C (2017) Towards flexible RFID event-driven integrated manufacturing for make-to-order production. Int J Comput Integr Manuf 31(10):1–15
Zhang G, Shao X, Li P, Gao L (2009) An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem. Comput Ind Eng 56:1309–1318

3. Capítulo 2: Local Search and Initialization in the Firefly Algorithm: Performance Analysis in Solving the Flexible Job-Shop Scheduling Problem

Tras la investigación realizada para el artículo expuesto en el Capítulo 2, se detectaron posibles mejoras a ser añadidas al FA que permitan obtener soluciones mejores y más robustas. Las metaheurísticas como el FA son la mejor alternativa para los problemas de optimización complejos NP-*hard* como es el FSJP estudiado, dada su capacidad de exploración y el balance entre calidad de las soluciones y el tiempo de computación. Todas las metaheurísticas tienen unas estrategias determinadas para guiar la exploración de una manera eficiente pero, sin embargo, ninguna estrategia es ideal y esto genera que en ciertas situaciones puedan verse atrapadas en óptimos locales. Para hacer frente a este problema, una de las tendencias más habituales es el empleo de lo que se conoce como metaheurísticas híbridas, que combinan estrategias de búsqueda de varias metaheurísticas o búsquedas locales para dar lugar a una exploración más robusta que sea capaz de escapar de dichos óptimos locales.

En este capítulo, correspondiendo con la segunda publicación, se presenta un FA híbrido para el FJSP, capaz de obtener mejores resultados que la versión del FA presentada en el Capítulo 1. Esta versión mejorada hace uso de dos nuevas técnicas: uso de una inicialización de la población más avanzada y la incorporación de varias estrategias de búsqueda local.

Una buena inicialización de la población inicial de una metaheurística puede tener un impacto notable en su actuación. Principalmente, las estrategias de inicialización buscan situar las soluciones de la población inicial en zonas prometedoras y diversas del espacio de búsqueda. Para ello, es importante tener en cuenta las características del problema específico a resolver. En la versión híbrida del FA propuesta se emplean tres estrategias de inicialización diferentes, cada una de ellas aplicada para generar un porcentaje determinado de la población inicial de luciérnagas. Dos de las estrategias son específicas del problema, que buscan buenas soluciones iniciales que tengan en cuenta simultáneamente la minimización de los tiempos de producción y la carga de trabajo de las máquinas involucradas, al tratarse de un FSJP multiobjetivo. La última estrategia es una inicialización aleatoria. El uso de estas tres estrategias diferentes permite obtener una población inicial diversa y bien situada en el espacio de búsqueda.

La versión mejorada del FA propuesta en este capítulo también hace uso de varias estrategias de búsqueda local, que son aplicadas tras cada generación de soluciones completas obtenidas mediante en esquema general de FA, realizando pequeñas modificaciones que puedan mejorarlas. En concreto, se implementan cinco estrategias diferentes inspiradas en otra conocida metaheurística conocida, el VNS, en concreto la versión introducida en Yazdani et al. (2009). Incorporando estas estrategias de mejora al FA se obtiene una versión híbrida mejorada, capaz evitando problemas de estancamiento en la búsqueda.

En el estudio se comparan tres versiones diferentes de FA para analizar claramente la aportación de la inicialización y de las estrategias de búsqueda local. Se compara una versión clásica del FA, otra a la que solo se le añade la inicialización mejorada, y otra con la inicialización y las búsquedas locales. Mediante una serie de experimentos con diferentes instancias del FSJP, se muestra como la versión híbrida que hace uso tanto de la inicialización como de las estrategias de mejora encuentra consistentemente mejores soluciones en todas las instancias usadas para la evaluación.

Local Search and Initialization in the Firefly Algorithm: Performance Analysis in Solving the Flexible Job-Shop Scheduling Problem

N. Alvarez-Gil, R. Rosillo, D. De la Fuente, and R. Pino

Abstract Hybrid metaheuristics are becoming a widely used alternative to solve some combinatorial optimization problems such as the Flexible Job-shop Scheduling Problem (FJSP). The inherent complexity of this type of problem requires methods that can find near optimal solutions in a reasonable computational time, since exact methods may be impractical in the real industry because of their exhaustive nature. Here is where metaheuristics, which have been proved to be very time-efficient in providing *quality* solutions, play a key role. Nevertheless, they also present some shortcomings like premature convergence and local optima stagnation. Hybrid versions are commonly used to avoid these issues and increase its search capability. In this paper, we conduct a comparative study of the performance of the Firefly Algorithm and two variants, one improved with an initialization phase and another that integrates both this initialization and multiple local search structures, in solving state-of-the-art FJSP instances. The study demonstrates how local search and initialization can notably enhance the performance of the algorithm.

Keywords Firefly algorithm · Local search · Initialization

N. Alvarez-Gil (✉) · R. Rosillo · D. De la Fuente · R. Pino
Departamento de Administración de Empresas, Escuela Politécnica de Ingeniería de Gijón, Universidad de Oviedo, Oviedo, Spain
e-mail: uo226901@uniovi.es

R. Rosillo
e-mail: rosillo@uniovi.es

D. De la Fuente
e-mail: david@uniovi.es

R. Pino
e-mail: pino@uniovi.es

D. De la Fuente et al. (eds.), *Organizational Engineering in Industry 4.0*,
Lecture Notes in Management and Industrial Engineering,
https://doi.org/10.1007/978-3-030-67708-4_9

1 Introduction

Combinatorial optimization problems have been the focus of many scientific research because of their complexity and practical importance in a wide variety of fields, and different algorithms have been developed for their resolution. These algorithms can be classified as complete and approximate algorithms [1], the former being algorithms that ensure to find the optimal solutions.

Nevertheless, when the problem is NP-hard [2], complete algorithms require an impractical amount of time, and thus approximate algorithms (i.e. metaheuristics) are the alternative commonly used in the real world. Approximate algorithms cannot guarantee to find the optimal solution but they notably reduce the amount of time required to provide good solutions [1].

The Job-shop Scheduling Problem (JSP) is "not only NP-hard, but it also has the well-earned reputation of being one of the most computationally stubborn combinatorial problems (…)" [3], and hence it is a problem that elicits interest in areas like planning and managing of manufacturing processes or operations research. In the JSP, there is a set of jobs that has to be performed and each job consists in a set of operations that must be processed in exactly one machine and the operations are subjected to precedence constraints. The goal of the JSP is to find a sequence of the operations that optimizes certain criteria, e.g. minimize the completion time, minimize the total machines' workload, etc.

The Flexible Job-shop Scheduling Problem (FJSP) is an extension of the JSP in which, in addition to the sequencing, a further decision level is required, the assignment: the operations that make up each job can be processed by any machine from a given set of compatible machines, and thus it is required to assign each operation to one machine. This fact makes the FJSP even more complex than the JSP.

Many approaches have been proposed for the resolution of the FJSP, especially (meta)heuristics algorithms. To name a few within the most relevant ones, Brandimarte [4] described a hierarchical approach for the FJSP, solving separately the sequencing and the assignment problems, both tackled by a tabu search (TS) algorithm. TS was also used by Mastrolilli and Gambardella [5] where, additionally, two neighborhood functions are introduced. Kacem et al. [6] presented a hybrid approach combining evolutionary algorithms and fuzzy logic for solving the FJSP with multiple objectives. In Pezzella et al. [7], a genetic algorithm (GA) with different strategies for initializing the population and for the selection and reproduction of the individuals is presented. Regarding constructive metaheuristics, particle swarm optimization and ant colony optimization has also been applied for this problem (see [8, 9], respectively).

In this paper, we present a comparative study of the performance of different versions of the firefly algorithm (FA) in optimizing the total completion time (makespan) in the FJSP. With this purpose, we have developed a standard discrete version of the FA, another version in which some strategies to generate the initial

population are used, and one hybrid version in which, together with this initialization, multiple local search procedures are integrated to increase the performance of the algorithm. These three versions have been tested with multiple state-of-the-art FJSP instances, providing a detailed analysis of the results.

The rest of the paper is organized as follows. In Sect. 2, a general description of the FJSP is provided. In Sect. 3, we describe the discrete version of the FA, the initialization procedure, and the local search strategies. The computational results of the tests and the comparative analysis are summarized in Sect. 4. Finally, Sect. 5 closes the paper with the conclusions obtained in the study.

2 Problem Definition

In the FJSP, there are n jobs, consisting each job J_i ($1 \leq i \leq n$) in a sequence of n_i operations and m machines. An operation O_{ij} ($i = 1,2, \ldots, n; j = 1,2, \ldots, n_i$) needs to be performed on one machine m_{ij} from the set of available machines M_{ij}. Each machine $k \in M_{ij}$ requires a certain processing time (P_{ijk}) to perform an operation O_{ij}.

Some assumptions are made: an operation cannot be interrupted (a_1); there are precedence constraints defined for any pair of operations within a job (a_2); machines are independent of each other (a_3); there are no precedence relations between jobs (a_4); transport and set-up times are already considered in P_{ijk} (a_5); and each machine can process at most one operation at any time (a_6).

The mathematical model could be given as follows:

$$\min f = \max_{1 \leq k \leq m} (C_k) \tag{1}$$

subject to

$$C_{ij} - C_{i(j-1)} \geq P_{ijk} X_{ijk},\ j = 2, \ldots, n_i; \forall i, j \tag{2}$$

$$[(C_{hg} - C_{ij} - P_{hgk}) X_{ghk} X_{ijk} \geq 0] \vee [(C_{ij} - C_{hg} - P_{ijk}) X_{hgk} X_{ijk} \geq 0], \forall i, j, h, g, k \tag{3}$$

$$\sum_{k \in M_{ij}} X_{ijk} = 1, \forall i, j \tag{4}$$

$$X_{ijk} \in \{0, 1\}, \forall i, j, k \tag{5}$$

Equation (1) ensures the minimization of the objective, the *makespan* (i.e. the maximal completion time of all the jobs). Constraint a_2 is ensured by Inequality (2) and constraint a_6 by Inequality (3). Equation (4) indicates that, for each operation, only one machine can be selected. Equation (5) is the binary decision variable ("1" if operation O_{ij} is assigned to machine k, "0" otherwise).

3 Firefly Algorithm

The firefly algorithm (FA) is a swarm intelligence algorithm inspired by the social behavior of the fireflies. The fireflies use their flashing lights to attract others with predation or mating purposes. It was originally developed for solving continuous optimization problems by Yang [10].

The main aspect of the FA is the association of the fireflies' light intensity with the objective function of the optimization problem. It is assumed that the firefly brightness (I) determines its attractiveness (β), both being in turn linked with the encoded objective function [11], the makespan in this study. If the aim is to minimize an objective, the objective function value of a firefly at a position x can be inversely associated with its brightness $I(x) \propto 1/f(x)$.

Fireflies are initially spread over the search space and each position stands for a solution to the optimization problem. At each iteration of the algorithm, the fireflies will move toward the brighter ones (i.e. for minimization problems, solutions with lower objective function value) within a certain region of the search space, depending on their distance and visibility. The fireflies' movement driven by its brightness, plus a random movement component, allows efficiently exploring the search space. A more detailed explanation of the original FA can be found in [10, 12].

This section is focused on the discrete version of the FA, which is required for the FJSP. The main aspects of the FA that need to be adapted are the representation of the solutions, the calculation of the distance, and how the movement is performed. The discrete approach presented in this paper is based on [11].

Solution Representation and Decoding. Each solution of the problem is obtained from two different vectors, one for each FJSP subproblem. The Sequencing Vector (SV) indicates the sequence of the operations and each job number J_i appears n_i times. The Assignment Vector (AV) denotes the machine assigned to each operation, an item AV[i] being a machine index. Both vectors have a length equal to the total number of operations, $\sum_{i=1}^{n} n_i$. Figure 1 shows the decoding of a solution from its AV and SV.

First, SV gives the sequence of the operations. The jth operation of J_i (O_{ij}) corresponds to the jth time a job index J_i appears. Then the machines m_{ij} assigned for each operation O_{ij} are obtained from the AV. The combination of these two vectors provides the final solution of a firefly: $\{(O_{11}, M_1), (O_{12}, M_1), (O_{31}, M_3), (O_{21}, M_1), (O_{32}, M_2), (O_{33}, M_3), (O_{13}, M_4), (O_{22}, M_2), (O_{23}, M_3), (O_{24}, M_1)\}$.

Measurement of the Distance Between Two Fireflies. The distance between the SVs of two fireflies can be carried out as the minimum number of swaps required

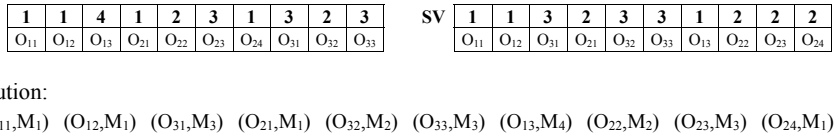

Fig. 1 Solution representation and decoding

Table 1 Distances and movement

P	AV = [2 4 3 1 3 1 4 4 1 2]	SV = [1 2 2 3 1 2 1 3 3 2]
P_{best}	AV = [2 1 3 4 1 1 4 4 1 2]	SV = [1 3 2 2 1 2 1 2 3 3]
d_{av} and d_{sv}	{(2,1), (4,4), (5,1)}	{(2,4), (8,10)
$\|d_{av}\|$ and $\|d_{sv}\|$	3	2
$\beta(r)$	0.53	0.71
rand $\in$ [0,1]	{0.34, 0.17, 0.76}	{0.09, 0.82}
Mov. β-step	{(2,1), (4,4)}	{(2,4)}
Position after β-step	2 1 3 4 3 1 4 4 1 2	1 3 2 2 1 2 1 3 3 2
Position after α-step	2 1 3 4 3 1 4 4 2 1	1 3 2 2 1 1 2 3 3 2

to bring one SV closer to the most attractive one ($|d_{sv}|$). For AV components, the distance is the number of non-corresponding items in the sequence, known as the Hamming distance ($|d_{av}|$). Table 1 shows how both distances between the AVs and the SVs of two fireflies (P and P_{best}) are calculated.

Fireflies' Movement. The movement of the original FA is divided into two not interchangeable steps: First, the β-step and then the α-step. The β-step is an insertion and pair-wise exchange mechanism used to bring the AV and SV of a firefly closer to the global best firefly. The β-step consist in the following sub-steps: all necessary pair-wise exchanges in SV and insertions in AV are found and stored in d_{sv} and d_{av}, respectively; the distances $|d_{sv}|$ and $|d_{av}|$ are stored in R; the probability $\beta(r) = \beta_o/(1 + \gamma r^2)$ is computed; a random number rand $\in$ [0,1] is generated for each element of d_{av} and d_{sv}; and then the corresponding insertion/pair-wise exchange is performed if $Rand \leq \beta$. The α-step is a swapping mechanism in which two non-equal element positions are chosen at random and swapped. Table 1 shows how the different steps of a firefly movement are performed, with $\beta_o = 1$, $\gamma = 0.1$, and $\alpha = 1$.

3.1 Initialization Module

In order to study how the initial population can impact the performance of the FA, we have implemented an initialization module. This module aims to initially locate the fireflies in promising areas of the search space instead of doing it randomly. The initialization rules explained below are the same used in the GA of Pezzella [8] who, in turn, follow the *approach by location* of Kacem et al. [13].

For AVs, two different rules are used: Ar_1 and Ar_2, both considering the processing times and the machines' workload. Ar_1 works as follows: the minimum processing time P_{ijk} is selected, and the machine k is assigned to the operation O_{ij}. Then all the columns corresponding to machine k are updated with P_{ijk}, and the process is repeated

until all the operations have been assigned to a machine. Ar_2 works similarly but, before starting, all rows (i.e. operations) and columns (i.e. machines) are randomly shuffled. Then, instead of selecting the minimum of the table, the minimum P_{ijk} of the new first row is selected, then the minimum of the second row, and so on, updating the columns as it was explained for Ar_1. The advantage of Ar_2 is that different assignments can be obtained in different runs of the algorithm.

For SVs, the well-known sequencing dispatching rules, the Most Work Remaining (MWR), and the Most number of Operations Remaining (MOR) are applied. In addition, to enhance diversity, some of the initial SVs are randomly generated.

3.2 Local Search Module

To enhance the search performance of the FA and to avoid common problems of the metaheuristics such as premature convergence or local optima stagnation, we have introduced several local search procedures into the algorithm. These local search (LS) strategies are based on the neighborhood structures used in the variable neighborhood search (VNS) algorithm presented in [14].

LS Strategy 1. Two element positions of the SV are selected at random and swapped, ensuring that no precedence constraint is violated. This is repeated multiple times depending on the total time of operations.

LS Strategy 2. A random operation is selected from the AV, and a different machine of the set of available machines for that operation is assigned, randomly as well. This step is repeated depending on the total time of operations. It can be noticed that LS strategies 1 and 2 are very similar to the α-step of the discrete FA.

LS Strategy 3. Two jobs are selected at random, and the positions of all its operations are swapped maintaining the precedence relations within each job, while the machine assignments remain the same.

LS Strategy 4. One operation assigned to the machine with the maximal workload (e.g. the sum of the processing times of the operations assigned to that machine) is randomly selected, and then it is assigned to the machine with the least workload, if possible. If not, it is assigned to any random available machine.

LS Strategy 5. One random operation assigned to the machine spending the maximum time to complete its assigned operations (equal to makespan) is selected, and then it is assigned to the machine spending the minimum time, if possible. If not, it is assigned to any random available machine.

4 Results and Comparative Study

This section describes the computational study conducted to compare how differently the FA performs with the initialization and the local search. We implemented three different versions of the FA: a standard discrete FA (V_S), another version integrated with the initialization module (V_I), and one more version with both the initialization and local search modules (V_{LS}). The algorithms were implemented in Python 3 and the tests were run on an Intel Core 7 2.1 GHz PC with 8 GB RAM memory.

Table 2 shows the average results (*Cmax*-Av.), the best results (*Cmax*-Best), and the average and best relative time of 30 runs of the three FA versions (V_S, V_I, and V_{LS}) for six different FJSP instances. Nxmxn_i stands for the number of jobs (n), the numbers of machines (m), and the total number of operations (n_i). Large instances were selected for the study because it is where more differences in the performance of algorithm's versions exist. Behnke and Geiger [15] provide a detailed explanation of the instances, from where we took the best-known results (BKR). *Av.Rel* and *Best.Rel* were calculated as $(tVx-tVs)/tVs$, tVx being the time spent by version x ($x = V_I$, V_{LS}) in reaching the maximal numbers of generations allowed (Max_{Gen}), which is the termination criteria. The parameters of the FA are number of fireflies $nf = 200$,

Table 2 Comparative study

				Cmax		Time	
Instance	*n* x *m* x *ni*	BKR	Ver	Av	Best	Av.Rel	Best.Rel
Mk1	10 x 6 x 55	40	V_S	50.4	48	–	–
			V_I	47.7	46	0.007	0.118
			V_{LS}	43	42	−0.008	−0.050
Mk3	15 x 8 x 150	204	V_S	246.5	231	–	–
			V_I	243.6	237	−0.127	−0.227
			V_{LS}	219.7	**204**	−0.255	−0.264
MFJS2	5 x 7 x 15	446	V_S	474.43	456	–	–
			V_I	460.96	**446**	−0.002	0.014
			V_{LS}	453.33	**446**	−0.026	−0.017
MFJS3	6 x 7 x 18	466	V_S	549.4	507	–	–
			V_I	530.4	497	−0.003	−0.005
			V_{LS}	497	**466**	−0.030	−0.035
Kacem 2	10 x 7 x 29	11	V_S	24.9	***21***	–	–
			V_I	14	13	−0.002	−0.004
			V_{LS}	11.96	**11**	−0.012	−0.008
Kacem 3	10 x 10 x 30	7	V_S	20.6	18	–	–
			V_I	8.03	**7**	−0.006	−0.004
			V_{LS}	8	**7**	−0.009	−0.008

$\text{Max}_{\text{Gen}} = 50$, $\beta_o = 1$, $\gamma = 0.1$, and $\alpha = 1$. For the initialization module: $Ar_1 = 20\%$, $Ar_2 = 80\%$, MWR $= 40\%$, MOR $= 40\%$, and random $= 20\%$.

It can be noticed how V_{LS} notably and consistently outperforms V_S and V_I (Table 2), both in average and best values, reaching the BKR for almost all the tested instances. It was expected since V_{LS} is the most complete version, using both the initialization and local search modules. Nevertheless, what catches our attention is that the time spent by V_{LS} is almost the same or lower than for V_S and V_I. As a preliminary hypothesis, we believe a possible reason is that the more different the fireflies are, the more computational time for the distance calculation and movement is required. Hence, when applying local search, many solutions are neighborhood solutions of others, and the distance between them is very low, requiring less time to compute it and perform the movement. Another explanation may be how we apply the local search to the firefly's population. To each solution from the first $nf/2$ set (better fireflies), we randomly apply one of the LS strategies. But, at each iteration, the second $nf/2$ (worse fireflies) is renewed with solutions obtained after applying one of the LS strategies to the best solution obtained so far, and thus half of the population are neighborhood solutions of the best one (more focused in the exploitation), while the other half is used for the exploration of promising areas.

5 Conclusions

It is common to look for improvements in the efficiency of metaheuristics algorithms by adding some kind of problem-specific strategies or knowledge. This allows to better explore the search space obtaining quality solutions to the optimization problem in a shorter time. In this work, the original Firefly Algorithm was enhanced with an initialization phase and some different local search procedures for solving the FJSP, aiming to analyze how these two upgrades affect its performance. With this purpose, we have explained and implemented three different versions of the FA—the original discrete version, another version with an initialization phase, and one more version with both the initialization and the local search modules—and compared them in the resolution of some middle- and large-sized state-of-the-art FJSP instances.

Computational results confirmed that the most complete version, the one that starts the search from solutions obtained from the initialization phase and that uses the different local search strategies during the search, consistently outperforms the other two versions, reaching the best known results in most of the tested cases. Future work will be focused on expanding this study to more FJSP instances and on studying further techniques to speed up the algorithm.

References

1. Blum C, Roli A (2003) Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Comput Surv 35(3):268–308
2. Garey MR, Johnson DS (1979) Computers and intractability. Freeman, A Guide to the Theory of NPCompleteness. W.H
3. Applegate D, Cook W (1991) A computational study of the Job-shop scheduling problem. J Comput 3(2):149–156
4. Brandimarte P (1993) Routing and scheduling in a flexible job shop by taboo search. Ann Oper Res 41:157–183
5. Mastrolilli M, Gambardella LM (1996) Effective neighborhood functions for the flexible job shop problem. J Sched 3:3–20
6. Kacem I, Hammadi S, Borne P (2002) Pareto-optimality approach for flexible job-shop scheduling problems: Hybridization of evolutionary algorithms and fuzzy logic. Math Comput Simul 60:245–276
7. Pezzela F, Morganti G, Ciaschetti G (2008) A genetic algorithm for the Flexible Job-shop scheduling problem. J Comput Oper Res 35:3202–3212
8. Zhang G, Shao X, Li P, Gao L (2009) An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem. Comput Ind Eng 56:1309–1318
9. Wang L, Cai J, Li M, Liu Z (2017) Flexible job shop scheduling problem using an improved ant colony optimization. Sci Program. https://doi.org/10.1155/2017/9016303
10. Yang X (2008) Nature-inspired metaheuristic algorithm. Luniver Press, Bristol
11. Karthikeyan S, Asokan P, Nickolas S (2014) A hybrid discrete firefly algorithm for multi-objective flexible job shop scheduling problem with limited resource constraint. Int J Adv Manuf Technol 72:1567–1579
12. Yang XS (2009) Firefly algorithm for multimodal optimization. Stochastic Algorithms: Found Appl 5792:169–178
13. Kacem I, Hammadi S, Borne P (2002) Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. IEEE Trans Syst Man Cybernet Part C 32(1):1–13
14. Yazdani M, Amiri M, Li P, Zandieh M (2009) Flexible job-shop scheduling with parallel variable neighborhood search algorithm. Exp Syst Appl 37:678–687
15. Behnke D, Geiger MJ (2012) Test instances for the flexible job shop scheduling problem with work centers. Research report, Helmut-Schmidt-University. ISSN 2192–0826

4. Capítulo 3: Problem instances dataset of a real-world sequencing problem with transition constraints and asymmetric costs

Este capítulo corresponde con una publicación que busca introducir un problema real de programación de la producción encontrado en una de las instalaciones de una empresa internacional de producción de acero. Para satisfacer el objetivo principal de la presente investigación de abordar retos actuales y reales de las empresas industriales, se pudo conocer en detalle el problema real de secuenciación de bobinas de acero en una línea de galvanizado.

El artículo asociado a este capítulo introduce brevemente este problema, que luego será objeto de la investigación en profundidad desarrollada en el Capítulo 5. La principal intención de esta publicación es aportar a la comunidad científica un nuevo problema que pueda ser de gran interés para el diseño, desarrollo y comparación de métodos de optimización combinatoria.

El problema presentado consiste en obtener una secuencia óptima de producción de un conjunto de bobinas de acero (formato final de los productos de acero planos entregado a los clientes). Estas bobinas son desbobinadas y soldadas una a otra para entrar en la línea de galvanizado, creando una banda continua de acero. Producir una bobina a continuación de otra tiene un coste determinado. Este coste se obtiene mediante un modelo matemático que considera diferentes aspectos de las bobinas (ancho, espesor, recubrimiento de zinc, calidad superficial, grado de acero, etc.). El modelo calcula un coste para cada par de bobinas a ser programadas en función de sus propiedades. Por ejemplo, transiciones bruscas de ciertas propiedades entre bobinas pueden generar que ciertos metros de la banda no cumplan los requisitos de calidad del cliente y tengan que ser achatarrados, con la correspondiente pérdida económica. Además, añadiendo gran complejidad al problema, ciertas bobinas no pueden ser secuenciadas juntas (p.ej. sus grados de acero no pueden ser soldados), dando lugar a restricciones bobina-bobina. La prioridad principal es encontrar una secuencia en la que ninguna de las transiciones viole las restricciones (solución factible) y, una vez encontrada una solución factible, minimizar el coste. Si se produce una secuencia con restricciones, se corre el riesgo de que la banda continua se rompa y la instalación tenga que pararse varios días, suponiendo un coste muy superior al de cualquier transición bobina-bobina.

Este problema presenta similitudes con algunos de los problemas de *scheduling* teóricos ampliamente estudiados en la literatura (*Asymmetric Traveling Salesman Problem*, *Hamiltonian Cycle Problem*, *Sequencing Ordering Problem*, etc), todos ellos demostrados ser NP-*Hard*. Sin embargo, la naturaleza de las restricciones encontradas en este problema real y su combinación con una distribución de costes asimétrica hace que el problema introducido merezca su propio interés y análisis. Este problema de secuenciación de bobinas de acero en una línea de galvanizado podría verse como una combinación del *Asymmetric Traveling Salesman Problem* y del *Hamiltonian Cycle Problem* dado que se podría definir como el problema de encontrar un camino hamiltoniano de mínimo coste, siendo los costes entre bobinas asimétricos. Solamente el problema de encontrar una solución factible, es decir, una secuenciación sin restricciones violadas (equivalente a encontrar un camino hamiltoniano en un grafo dirigido) es un reto altamente complejo, requiriendo técnicas de resolución con mecanismos de gestión de restricciones.

Además de explicar el problema, su origen e interés para la comunidad científica, se aportan 30 instancias reales del problema obtenidas de secuencias reales producidas en la línea acabadora. Estas instancias están disponibles y se explica en detalle cómo deben ser interpretadas, su formato y los objetivos del problema a resolver.

Data in Brief 41 (2022) 107844

Contents lists available at ScienceDirect

Data in Brief

journal homepage: www.elsevier.com/locate/dib

Data Article

Problem instances dataset of a real-world sequencing problem with transition constraints and asymmetric costs

Nicolás Álvarez-Gil[a,*], Segundo Álvarez García[b], Rafael Rosillo[a], David de la Fuente[a]

[a] Department of Business Administration, University of Oviedo, C\ Luis Ortiz Berrocal s/n Campus de Gijón, Gijón, Asturias 33203, Spain
[b] ArcelorMittal Global R&D Asturias, P.O. Box 90, Avilés, Asturias 33400, Spain

ARTICLE INFO

Article history:
Received 24 December 2021
Revised 11 January 2022
Accepted 14 January 2022
Available online 19 January 2022

Keywords:
Steel industry
Combinatorial optimization
Manufacturing scheduling
Metaheuristics

ABSTRACT

This data article describes 30 instances of the real-world problem of sequencing steel coils in a continuous galvanizing line. Each instance is represented by a cost matrix that gives information of the cost of sequencing each pair of coils or items together (e.g. a transition). Some transitions are forbidden due to technical limitations of the line and/or because of the properties of the coils, what makes the problem more challenging. These costs were previously obtained by a cost model that estimates the final cost of each transition for a set of coils to be sequenced in the line. Although the instances come from this real context, the problem can be theoretically seen as finding a minimum cost Hamiltonian path (e.g. a minimum cost feasible production sequence with all the coils appearing just once). It is a well-known NP-Hard combinatorial optimization problem. Since these instances represent real challenges found in the industry, they can be very useful for algorithm development and testing. Due to the cost distributions obtained for the given coils, just finding a feasible sequence can be a challenging task, especially for some types of approximate algorithms (Alvarez-Gil et al., 2022).

DOI of original article: 10.1016/j.cie.2021.107908
* Corresponding author.
E-mail address: alvareznicolas@uniovi.es (N. Álvarez-Gil).

https://doi.org/10.1016/j.dib.2022.107844

Specifications Table

Subject	Business, Management and decision sciences (Management Science and Operations Research)
Specific subject area	Operations Research applications (planning & scheduling, logistics, etc.) and Combinatorial Optimization (algorithm development and benchmarking)
Type of data	• Problem instances in separated text files (.txt) • Figures • Tables
How the data were acquired	The data were acquired using a customized software that helps the scheduling crew with the sequencing tasks. The software receives the information of the coils to be sequenced. The software has an embedded cost model that calculates, for each pair of coils, the cost of sequencing them together based on its properties, generating a cost matrix for the set of coils. Then the software allows to export the cost matrices in the provided format (.txt).
Data format	Raw
Description of data collection	The data were acquired ensuring that: • Each instance corresponds to a daily real production schedule • Each set of coils is sequenceable (e.g. there exist a Hamiltonian path)
Data source location	Spanish steel factory
Data accessibility	Repository name: Problem instances of the sequencing problem of a steel continuous galvanizing line (Mendeley Data) Data identification number: DOI:10.17632/v357z2ncbh.2 Direct URL to data: https://data.mendeley.com/datasets/v357z2ncbh/2
Related research article	N. Alvarez-Gil, S. Alvarez, R. Rosillo, D. de la Fuente, Sequencing Jobs with Asymmetric Costs and Transition Constraints in a Finishing Line: A real case study, Computers & Industrial Engineering 165 (2022). 10.1016/j.cie.2021.107908

Value of the Data

- The dataset provides 30 different instances of a real-world sequencing problem with transition constraints and asymmetric costs that can be used to evaluate and compare algorithms performance. The reference results presented in [1] can be used as benchmark solutions.
- The instances can help in the development, design and testing of combinatorial optimization algorithms. Since the instances are highly constrained, it can help in the development of new approaches able to efficiently explore the solution space and in the design of constraint-handling strategies. Due to the cost distributions obtained from the real sets of items to be sequenced in the line, some of the instances are very complex and can be very challenging for the approximate algorithms, especially for those algorithms that use the cost information to guide the exploration, and this focus on the costs can misguide the feasibility search (see [1]). The use of the presented dataset can be very useful to develop robust solutions able to handle constraints while minimizing the cost.
- Problem instances are directly provided as a cost matrix what makes them easy and ready to use. They represent real challenges of a scheduling problem found in the industry, but they can also be abstracted to the more theoretical problem of finding a minimum-cost Hamiltonian path [2]. Hence, they can be used both to gain insights for the different combinatorial optimization applications (scheduling, routing, etc.) and for theoretical purposes (graph analysis, constraint-handling strategies, approximate algorithms, etc.).

1. Data Description

The dataset consists in 30 different problem instances with sizes ranging from 17 to 114 items (e.g. coils, nodes, etc.). Each instance is named as "*cgl_X.txt*", being *X* the size of the instance. Table 1 shows the name and the size of the problem instances provided in the dataset.

Each instance is represented by a n x n square matrix, being n the size of the instance (e.g. number of items to be sequenced). The cost matrix provides information about the cost of scheduling two pair of coils together in the sequence. The element [i,j] of the matrix represents the cost of processing item i right before item j (C_{ij}). An element C_{ij} can take two possible values:

- $C_{ij} > 0$, $C_{ij} \in \mathbb{R}$: This means that the transition between item i and j is allowed (e.g. item i can be produced immediately before item j) with a cost C_{ij}.
- $C_{ij} = -1$: This means that the transition between item i and j is not allowed (e.g. item i cannot be produced immediately before item j) and it represents a hard constraint.

The problem consists in finding a minimum-cost feasible sequence containing all the nodes just once, what can be seen as finding a minimum-cost Hamiltonian path [2]. We refer to a feasible sequence as a sequence for which all the transitions are allowed: a sequence without constraints violations. Just finding a feasible sequence is a challenge itself, but the cost should also be minimized. The lower number of constraints the better. For a feasible sequence, the lower the cost the better. The initial and final items of the sequence are not fixed and thus that decisions are part of the optimization problem, since they can directly impact in the final cost of the sequence.

The problem shares some similarities with other well-known combinatorial optimization problems such as the Asymmetric Traveling Salesman Problem [3,4], the Sequencing Ordering Problem [5,6] and the Hamiltonian Cycle Problem [7,8]. More information about the similarities and differences can be found in [1].

All the cost matrices are provided in the same format, shown in Fig. 1.

Table 1
List of problem instances names, sizes and reference solutions.

Instance name	Size	Instance name	Size
cgl_17.txt	*17 items*	*cgl_51b.txt*	*51 items*
cgl_26.txt	*26 items*	*cgl_57.txt*	*57 items*
cgl_28.txt	*28 items*	*cgl_58.txt*	*58 items*
cgl_32.txt	*32 items*	*cgl_60.txt*	*60 items*
cgl_33.txt	*33 items*	*cgl_66.txt*	*66 items*
cgl_37.txt	*37 items*	*cgl_70.txt*	*70 items*
cgl_38.txt	*38 items*	*cgl_70b.txt*	*70 items*
cgl_43.txt	*43 items*	*cgl_72.txt*	*72 items*
cgl_44.txt	*44 items*	*cgl_73.txt*	*73 items*
cgl_45.txt	*45 items*	*cgl_76.txt*	*76 items*
cgl_47.txt	*47 items*	*cgl_78.txt*	*78 items*
cgl_48.txt	*48 items*	*cgl_81.txt*	*81 items*
cgl_48b.txt	*48 items*	*cgl_88.txt*	*88 items*
cgl_50.txt	*50 items*	*cgl_107.txt*	*107 items*
cgl_51.txt	*51 items*	*cgl_114.txt*	*114 items*

```
-1;300;-1;654;926;-1;-1;-1;-1;-1;-1;-1;-1;-1;-1;-1;-1
305;-1;963;117;237;-1;-1;-1;-1;-1;-1;-1;-1;-1;-1;-1;-1
952;493;-1;704;686;-1;-1;-1;-1;-1;-1;-1;-1;-1;-1;-1;-1
678;123;-1;-1;41;-1;-1;-1;-1;-1;-1;-1;-1;-1;-1;-1;-1
946;237;1155;34;-1;804;-1;-1;-1;-1;-1;-1;-1;-1;-1;-1;-1
-1;-1;-1;-1;-1;-1;71;1721;-1;-1;-1;-1;-1;-1;-1;-1;-1
-1;-1;-1;-1;-1;73;-1;1777;-1;-1;-1;-1;-1;-1;-1;-1;-1
-1;-1;-1;-1;-1;2034;2020;-1;634;634;634;634;634;634;634;634;67
-1;-1;-1;-1;-1;-1;-1;685;-1;0;0;0;0;0;0;0;308
-1;-1;-1;-1;-1;-1;-1;685;0;-1;0;0;0;0;0;0;308
-1;-1;-1;-1;-1;-1;-1;685;0;0;-1;0;0;0;0;0;308
-1;-1;-1;-1;-1;-1;-1;685;0;0;0;-1;0;0;0;0;308
-1;-1;-1;-1;-1;-1;-1;685;0;0;0;0;-1;0;0;0;308
-1;-1;-1;-1;-1;-1;-1;685;0;0;0;0;0;-1;0;0;308
-1;-1;-1;-1;-1;-1;-1;685;0;0;0;0;0;0;-1;0;308
-1;-1;-1;-1;-1;-1;-1;685;0;0;0;0;0;0;0;-1;308
-1;-1;-1;-1;-1;-1;-1;77;293;293;293;293;293;293;293;293;-1
```

Fig. 1. Content of file cgl_17.txt.

Each text file (.txt) has n rows, and each row has n columns delimited by the character ";". Taking instance *cgl_17.txt* for illustration (Fig. 1), the cost matrix should be interpreted in the following way:

From row 1:

- Matrix element [0, 0] ($C_{00}= -1$): not possible, each item must appear just once in the sequence.
- Matrix element [0, 1] ($C_{01}= 300$): cost of producing item 0 right before item 1.
- Matrix element [0, 2] ($C_{02}= -1$): not possible, production constraint.
- Matrix element [0, 3] ($C_{03}= 654$): cost of producing item 0 right before item 3.
- Matrix element [0, 4] ($C_{04}= 926$): cost of producing item 0 right before item 4.
- (...)

From row 2:

- Matrix element [1, 0] ($C_{10}= 305$): cost of producing item 1 right before item 0.
- Matrix element [1, 1] ($C_{11}= -1$): not possible, each item must appear just once in the sequence.
- Matrix element [1, 2] ($C_{12}= 963$): cost of producing item 1 right before item 2.
- Matrix element [1, 3] ($C_{13}= 117$): cost of producing item 1 right before item 3.
- Matrix element [1, 4] ($C_{14}= 237$): cost of producing item 1 right before item 4.
- (...)

A possible solution for this instance is $S = [2, 0, 1, 3, 4, 5, 6, 7, 16, 11, 15, 10, 8, 12, 14, 13, 9]$. It is a valid sequence since all the items appears just once, and the length of the sequence is n. In order to get the final cost of the sequence, all the transition costs should be computed, and the final cost is the sum of them. For a sequence of length n, the number of transitions is $n - 1$. For sequence S, the cost C_S will be:

$$Cs = C_{2,0} + C_{0,1} + C_{1,3} + C_{3,4} + C_{4,5} + C_{5,6} + C_{6,7} + C_{7,16} + C_{16,11} + C_{11,15} + C_{15,10} + C_{10,8} + C_{8,12} + C_{12,14} + C_{14,13} + C_{13,9}$$

$$Cs = 952 + 300 + 117 + 41 + 804 + 71 + 1777 + 67 + 293 + 0 + 0 + 0 + 0 + 0 + 0 + 0 = 4422$$

It is worth mentioning that, for all the instances of the dataset, the sequence [0, 1, 2, 3, …, *n*-1] is a feasible sequence and can be used as a result reference. The only exception is instance *cgl_38.txt*, for which that solution contains one constraint violation.

2. Experimental Design, Materials and Methods

The instances come from the real-world problem of sequencing steel coils (e.g. final format of flat steel products) in a continuous galvanizing line. The galvanization is a finishing process that provides the coils with a zinc layer to protect them against air and moisture [9]. The line operators use a stand-alone scheduling application installed in their computer that receives, for each day, the set of coils that should be sequenced and their main properties (width, thickness, steel grades, zinc coating, etc.). This information is obtained connecting directly to the plant Manufacturing Execution System (MES). Then, the software calculates the cost of processing each pair of coils and creates the cost matrix, which is the only input required for the sequencing problem.

The galvanizing process is continuous, being all the coils welded to the others in order to create a continuous strip. The cost of sequencing two coils together depends on their properties, seeking to avoid sudden changes in the main properties (width, thickness, zinc coating, etc.) when processing a new coil. For example, if two consecutive coils have different widths, some meters of strip near the welding zone may not meet the quality requirements and may be sold as scrap at a much lower price. Additionally, if the change in some properties is very sharp, there exists risk of strip breakage and that transition is penalized with a high cost. The scheduling software also calculates the constraints. Some coils cannot be sequenced together because their steel grades are not weldable or because the difference in some of their properties exceeds a threshold defined by the expert schedulers.

In order to obtain the problem instances of the dataset, we looked for daily schedules processed in the line in the past, so we were sure that the set of is sequenceable. Then, we used the scheduling software to load that schedules, generate the cost matrices and exported them in the provided format.

Ethics Statement

N/A.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT Author Statement

Nicolás Álvarez-Gil: Conceptualization, Methodology, Software, Formal analysis, Investigation, Resources, Writing – original draft, Writing – review & editing, Data curation; **Segundo Álvarez García:** Conceptualization, Methodology, Software, Formal analysis, Investigation, Resources, Writing – review & editing, Data curation; **Rafael Rosillo:** Conceptualization, Methodology, Visualization, Investigation, Supervision, Writing – review & editing; **David de la Fuente:** Conceptualization, Investigation, Supervision.

Acknowledgments

N/A.

References

[1] N. Alvarez-Gil, S. Alvarez, R. Rosillo, D. de la Fuente, Sequencing jobs with asymmetric costs and transition constraints in a finishing line: a real case study, Comput. Ind. Eng. 165 (2022), doi:10.1016/j.cie.2021.107908.
[2] M. Sohel Rahman, M. Kaykobad, On Hamiltonian cycles and Hamiltonian paths, Inf. Process. Lett. 94 (2005) 37–41, doi:10.1016/j.ipl.2004.12.002.
[3] D.L. Applegate, R.E. Bixby, V. Chvátal, W.J. Cook, The Traveling Salesman Problem: A Computational Study, Princeton University Press, 2007, doi:10.1515/9781400841103.
[4] K. Helsgaun, An effective implementation of the Lin–Kernighan traveling salesman heuristic, Eur. J. Oper. Res. 126 (2000) 106–130, doi:10.1016/S0377-2217(99)00284-2.
[5] L.F. Escudero, An inexact algorithm for the sequential ordering problem, Eur. J. Oper. Res. 37 (1988) 236–249, doi:10.1016/0377-2217(88)90333-5.
[6] N. Ascheuer, M. Jünger, G. Reinelt, A branch & cut algorithm for the asymmetric traveling salesman problem with precedence constraints, Comput. Optim. Appl. 17 (200) (2000) 61–84, doi:10.1023/A:1008779125567.
[7] M.R. Garey, D.S. Johnson, R.E. Tarjan, The planar Hamiltonian circuit problems is NP-complete, SIAM J. Comput. 5 (1976) 704–714, doi:10.1137/0205049.
[8] M.R. Garey, D.S. Jhonson, Computers and Intractability A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, 1979.
[9] S. Fernández, S. Álvarez, D. Díaz, M. Iglesias, B. Ena, Scheduling a galvanizing line by ant colony optimization, in: ANTS 2014: Lecture Notes in Computer Science, 8667, 2014, pp. 146–157, doi:10.1007/978-3-319-09952-1_13.

5. Capítulo 4: Sequencing jobs with asymmetric costs and transition constraints in a finishing line: A real case study

Esta última publicación se centra en el diseño de un método de optimización capaz de asegurar soluciones factibles de mínimo coste para el problema anteriormente introducido de secuenciación de bobinas de acero en una línea de galvanizado. Para ello, primero se expone en detalle la complejidad del problema y varios aspectos interesantes detectados tras un estudio en profundidad que hacen que, en algunos casos, los métodos de optimización usados por la empresa no sean capaces de obtener soluciones factibles. Obtener soluciones factibles para un conjunto de bobinas a ser producidas, si existe, es fundamental dado el impacto económico que conlleva.

En el momento en que se inició esta investigación, se empleaban algoritmos metaheurísticos de la familia *Ant Colony Optimization* (ACO) para obtener las secuencias a producir por la línea acabadora. Los algoritmos ACO están inspirados en el comportamiento de las colonias de hormigas, y como son capaces de coordinarse para encontrar caminos óptimos a las fuentes de comida mediante el depósito de feromona (Dorigo and Stützle, 2004). Estos algoritmos metaheurísticos son una buena opción al proporcionar soluciones de calidad en tiempos de computación razonables para los operadores de la línea, y por tener una naturaleza constructiva que facilita la implementación de las restricciones bobina a bobina. Sin embargo, durante los últimos años, la imperante necesidad de reducir los niveles de inventario y la aparición de nuevos grados de acero trajo consigo un notable aumento en el número de restricciones, suponiendo un reto mayor para los algoritmos ACO en uso y provocando que, para ciertas situaciones, existiendo soluciones factibles, éstos no fueran capaces de encontrarlas.

Los algoritmos metaheurísticos constructivos como los ACO van añadiendo elementos a la solución hasta encontrar una solución completa. A la hora de añadir un nuevo elemento (p.ej. bobina) a la solución, los algoritmos ACO miran dos aspectos: el coste directo de añadir dicho elemento (a mayor coste, menor probabilidad de ser añadido) y la información aprendida durante el proceso de búsqueda, a través de un mecanismo de depósito y evaporación de feromona (decisiones con mayor feromona, significa que formaron parte de buenas soluciones pasadas, y por tanto tienen mayor probabilidad de ser elegidas).

Durante la investigación, se analizaron en profundidad varias instancias del problema y se detectó que, para ciertos conjuntos de bobinas seleccionados para ser producidos (selecciones principalmente basadas en los plazos de entrega), se obtienen distribuciones de costes especiales que hacen que, para encontrar soluciones factibles, sea necesario tomar varias decisiones locales de máximo coste. Para algoritmos constructivos como los ACO, al dar mayor probabilidad a las decisiones que aportan poco coste, este tipo de distribuciones de costes provocan que tengan grandes dificultades para obtener soluciones factibles.

Detectado el problema principal y la razón de que estos algoritmos, aunque en general obtengan buenas soluciones, en algunas situaciones no encontraran soluciones factibles, se diseñó una estrategia de exploración novedosa, llamada *Interval Reconstruction* (IR), que permitiera solventar los defectos de los ACO en uso, y asegurar la obtención de soluciones factible, objetivo principal para la empresa.

En la publicación se explican las características de esta nueva estrategia y todos los detalles necesarios para su implementación e incorporación a los algoritmos ACO. Principalmente, su exploración en dos niveles (primero atacando a las restricciones y luego buscando reducir costes), y su elevada eficiencia, hacen que el IR sea una estrategia muy potente, aplicable también para otros problemas de optimización combinatoria. Finalmente se demuestra como introduciendo esta nueva técnica de exploración en un algoritmo ACO, el *Ant System*, se obtuvo una metaheurística híbrida capaz de asegurar soluciones factibles y de mejor coste en todas las instancias estudiadas del problema.

Computers & Industrial Engineering 165 (2022) 107908

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

Sequencing jobs with asymmetric costs and transition constraints in a finishing line: A real case study

Nicolás Álvarez-Gil [a,*], Segundo Álvarez García [b], Rafael Rosillo [a], David de la Fuente [a]

[a] Business Management Department, University of Oviedo, Edificio Departamental Este, 1ª planta, Campus de Gijón (33204), Gijón (Asturias), Spain
[b] ArcelorMittal Global R&D Asturias, P.O. Box 90 (33400), Avilés (Asturias), Spain

ARTICLE INFO

Keywords:
Combinational optimization
Steel industry
Sequencing
Metaheuristics

ABSTRACT

Production scheduling plays a vital role in industrial manufacturing due to the potential impact on the production costs and service levels of a company. It consists in finding the best sequence in which some items should be produced, optimizing one or multiple performance indicators, such as the production cost or total time span. In this work we study the real-world problem of sequencing steel coils in a continuous galvanizing line and the challenges it poses. The production of new steel grades and the growing necessity or reducing the stock levels at the galvanizing line have brought an important increase in the number of sequencing constraints, challenging feasibility and the algorithms in use. We explain some issues of the current Ant Colony Optimization algorithms and introduce a new hybrid version, the Ant System with Interval Reconstruction (AS-IR), that notably enhances the feasibility performance. The new hybrid algorithm uses the Interval Reconstruction (IR), a novel *constructive local search* algorithm initially developed to solve constraint violations, and then extended to also help reduce the sequencing costs. All the key features of the IR and how it is used in the hybrid algorithm are explained in detail. The experiments conducted with 30 real instances show how the proposed AS-IR hybrid algorithm achieves much better results, guaranteeing feasible sequences when the set of coils is *sequenceable*, as well as finding lower-cost solutions.

1. Introduction

Task scheduling has numerous applications in very different business, industry and scientific domains, encompassing fields as diverse as logistics, maintenance, biology, robotics, aircraft design, etc. At industrial companies, the scheduling of production processes is an activity of critical importance since it directly impacts on efficiency, production rates, customer satisfaction and, hence, on the overall benefits of the company.

The scheduling problems can be defined as "the allocation of available production resources over time to best satisfy some set of criteria" (Graves, 1981). The production management of a company involves multiple decisions that differ on their scope, time and impact. These decisions are usually taken following a hierarchical structure: strategic, tactical and operational decisions (Framinan et al. 2014). Manufacturing scheduling are low-level operational decisions that provides a detailed production plan for a particular shop floor, and usually are short-term complex decisions that require well-defined data, constraints and objectives (Verderame and Floudas 2010). Most of the scheduling problems, that arise from the necessity of enhancing the efficiency in the use of the resources and the management of operations, are combinatorial optimization problems.

Many combinatorial problems are NP-hard problems, which means that it does not exist any polynomial-time algorithm that can solve them, assuming that $P \neq NP$ (Garey and Johnson 1979). Examples for this type of problems are the Traveling Salesman Problem (Applegate et al. 2006) or the Job Shop Scheduling Problem (Applegate and Cook 1991). If the problem is NP-hard, the methods that can ensure success in finding the optimal solution in bounded time might need exponential computation time in the worst-case (Blum and Roli 2003), which is impractical for most of the real-world applications. This is the reason why it is more common to use approximate methods such as heuristic or metaheuristics algorithms.

The trade-off between computation time and solution quality is very important in the industrial world, especially at the operational level, where the decisions should be made fast, being therefore acceptable to

* Corresponding author.
E-mail addresses: alvareznicolas@uniovi.es (N. Álvarez-Gil), segundo.alvarez-garcia@arcelormittal.com (S. Álvarez García), rosillo@uniovi.es (R. Rosillo), david@uniovi.es (D. de la Fuente).

https://doi.org/10.1016/j.cie.2021.107908
Received 22 January 2021; Received in revised form 7 December 2021; Accepted 20 December 2021
Available online 24 December 2021

sacrifice solution quality to reduce computation time. The approximate methods cannot guarantee the optimal solution, but they can provide reasonably good solutions in a reduced amount of time.

Among the approximate methods we found constructive metaheuristics (Blum & Roli 2003) a promising approach for the sequencing of the finishing lines. The Greedy Randomized Adaptive Search Procedure (Feo & Resende 1995) and Ant Colony Optimization (ACO) (Dorigo 1992) are two well-known constructive methods. This type of metaheuristics presents advantages for the studied problems (e.g. flexibility to add new constraint rules) but sometimes, as it is addressed in this paper, the solutions provided by the constructive methods should be enhanced.

As contributions of this work, we introduce the scheduling problem of sequencing steel coils in a continuous galvanizing line (CGL) and provide 30 different real instances, which can be used as new benchmark instances for algorithm testing. We explain our implementation of the current ACO algorithms, the modifications that were required to use them for this problem, and the main issues identified that make them fail in finding feasible sequences in some challenging instances. In addition, we present a new Ant System hybrid version that solves the feasibility issues. It uses the Interval Reconstruction (IR), a new algorithm developed for solving constraints violations, that also reduces costs and can be embedded in other metaheuristics. The key features of the IR and our implementation of the AS-IR are explained in detail, together with a demonstration of its performance on the real instances provided.

The rest of the paper is structured as follows: Section 2 provides a literature review of existing studies related to combinatorial optimization and scheduling in the steel industry. In Section 3 we provide a description of the problem and the instances. The two ACO algorithms analyzed, the main challenges detected, and the new hybrid algorithm are explained in Section 4. Section 5 shows the results and analysis of the computational study and finally Section 5 closes the paper with the conclusions and future research.

2. Literature review

There are numerous studies on combinatorial optimization problems (Korte and Vygen 2003, Blum and Roli 2003) and, particularly, on scheduling problems (Graves 1981, Zhang et al. 2017). As a high-level breakdown, we can differentiate between the papers that are focused on theoretical problems and the ones that present or analyze real-world problems. These two kinds of research are closely related, and each is benefitted by the other. Usually, some theoretical problems are inspired by real-world problems, but are formulated in a more generic and abstract form. At the same time, the research on real cases tends to use ideas, insights and approaches developed for the theoretical problems. In this case, we found very useful some papers belonging to both types.

Regarding the theoretical studies, we noticed that the presented problem shares some similarities with the following well-known state-of-the-art problems:

- The Traveling Salesman Problem (TSP). The TSP states as follows: given a set of cities and the distance or *cost of travel* between each pair of them, find the shortest or cheapest tour that visits all the cities exactly once. If costs of travel between cities are not symmetric, the problem is called the Asymmetric TSP (ATSP). According to Applegate et al. (2006), the TSP is, due to its simple formulation but hard resolution, the most studied problem in combinatorial optimization, being a challenging testbed to develop algorithms. Applegate gives a comprehensive history of the TSP and the evolutions of TSP solvers. Within the wide variety of approaches developed for the TSP, we would like to highlight the Lin-Kernighan-Helsgaun, the most effective implementation of the Lin–Kernighan traveling salesman heuristic (Helsgaun, 2000), and the Ant Colony System (ACO) presented in Dorigo and Gambardella (1997). The TSP is still the focus of many recent studies on new applications and solution methods (Nekovář et al. 2021, Twaróg et al. 2021; Pina-Pardo et al. 2021).

For scheduling the galvanizing line, if we compare the cities with the coils, the distances with the transition costs, and the tours with the production sequences, the presented problem is almost the same that the ATSP, except that in the ATSP no forbidden arcs or constraints are considered. Arc constraints can be modelled as high costs but, sometimes, it may not be the best approach for this problem as we will see in Section 4.

- The Hamiltonian Cycle Problem (HCP). The HCP states as follows: given an unweighted graph of nodes or vertices connected by directed edges, determine if there is a Hamiltonian cycle (a cycle along the graph that visits all nodes exactly once), or prove that that cycle does not exists. Determining this is in practice equivalent to find any Hamiltonian cycle in the graph. This problem has been shown to be NP-complete even if limited to planar graphs (Krishnamoorthy 1975, Garey et al. 1976, Akiyama et al., 1980). One of the best algorithms for the HCP is the Snake and Ladders, "(…) a polynomial-complexity algorithm inspired by, but distinctly different from, the k − opt heuristics" (Baniasadi et al., 2013).

The HCP is similar to the proposed problem in the sense that it is needed to find a tour or a sequence, given a graph in which not all the nodes are directly connected with the others. The main differences are that we need to find a minimum-cost Hamiltonian path, but in the HCP there is no cost or weight associated to each edge.

- The Sequencing Ordering Problem (SOP). The SOP states as follows: given a directed graph with weighted edges and nodes subject to precedence constraints, where the nodes stand for the jobs and the arcs for the production costs, find the minimum-cost Hamiltonian path. The SOP was first formulated by Escudero (1988) and many different methods has been developed for its resolution, from simple heuristics to sophisticated branch & bound algorithms (B&B). To name a few, Ascheuer et al. (2000) presented a branch & cut (B&C) algorithm for solving large SOP instances in a few minutes, based on a Mixed Integer Linear Programming (MILP) formulation. Another MILP approach by Montemanni et al. (2013) proposes a decomposition method (DEC). The goal was to split the original problem in different sub-problems, solve them separately and recombine the solutions. Gambardella and Dorigo (2000) introduced the HAS-SOP, the first ACO algorithm devised for the SOP that combines a constructive initial phase with a "lexicographic" local search named "SOP-3-exchange". Recently, the Lin-Kernighan extension by Helsgaun for solving many types of constrained problems, the LKH3, has obtained the best-known solutions for almost all the SOP TSPLIB instances, reporting also some new best solutions (Helsgaun, 2017).

We found the SOP the closest theoretical problem to our problem, since there are asymmetric costs and precedence constraints, but a slight difference in the nature of the constraints requires a totally different approach to handle them. In the SOP, if there is a constraint between node *i* and node *j*, it means that node *i* cannot be sequenced at any position of the sequence before node *j*. Differently, in our problem, the same constraint means that node *i* cannot be produced right before node *j*, but it can be produced at any position before node *j* as long as they are not together and all the other pairwise constraints are respected (similar to the HCP problem). This difference makes that in our problem the constraints cannot be backward propagated as in the SOP.

There are also works in the literature focused on real planning and scheduling solutions in the steel industry. Harjunkoski and Grossman (2001) presented a decomposition approach for the scheduling of the processes between the electric arc furnaces and the continuous caster, splitting the original problem into smaller subproblems that can often be

optimally solved, reducing the overall complexity of the problem. A combination of Lagrangian relaxation, dynamic programming and heuristics rules is used by Tang et al. (2002) for the scheduling of the steel-making process. Cowling et al (2004) used a multi-agent approach for the dynamic scheduling of steel milling and casting. Many other works focused on primary steel-making processes can be found in the literature (see exhaustive reviews in Lee et al. 1996, Tang et al. 2001, and Iglesias-Escudero et al. 2019).

However, when it comes to the scheduling of steel finishing lines, the existing works are scarce.

Okano et al (2004) present an ambitious project that involves multiple processes from the cold mill onward, including the CGL. The project consists of several phases, such as initial clustering, the creation and allocation of the campaigns, time windows management and the final sequencing of the coils for a one-month horizon. The sequencing problem is modelled as a traveling salesman problem with times windows (TSPTW) and they use a constructive heuristic and a local search algorithm to solve it. They consider the sequencing constraints in the objective functions and in the estimation of the distances between coils, though they do not address in their model the common use of transition coils to resolve constraints in case of not finding a feasible solution. Prior to sequencing, they try to guarantee the *sequenceability* (whether or not a feasible solution exists) of each campaign during the clustering and campaign creation phases, which facilitates the sequencing task. Eventually, the detailed coil sequences for a few days horizon are done by human experts who can manually fix the orderings. They manifest that "the CGL sequencer is regarded as the most difficult process in the finishing lines in terms of sequencing, … even finding a feasible solution is NP-Hard". Our paper is focused exactly on this latter phase of sequencing the coils, but with the objective of ensuring finding a low-cost feasible solution, if it exists, given fixed group of coils.

The scheduling of a Turkish steel company CGL is studied in (Kapanoglu and Koc, 2006). First, the campaign is created with the coils that better satisfy the due dates, predefined priorities and campaign tonnage. Then these selected coils are sequenced with a multi-stage genetic algorithm (MSGA). If a feasible solution was not found by the MSGA, all the violated constraints are solved one by one using a heuristic algorithm that tries to solve them using the minimum number of coils from the inventory. The possibility of using inventory coils for solving the constraints violations facilitates the problem. In addition, performing this repairing method after the sequencing may not provide the best solutions, since the sequence is not optimized again after the insertions. In our problem, the human experts generate the initial selection of coils according to the service level and due dates, and then a feasible solution should be found without using additional coils.

(Valls Verdejo et al., 2009) also studied the problem of scheduling a CGL. In their work, they differentiate between the campaign creation phase and the intra-campaign coils scheduling, being the latter the focus of the study. Within the same campaign, they managed three different types of coils depending on its surface quality requirements. The main challenge of their work is how to split and sequence one type of coils, so the existing constraints and requirements are met, for what they use a Tabu Search algorithm.

Lastly, Fernández et al. (2014) presented a very similar problem to the one studied on this paper and how they were able to notably enhance the productivity and efficiency of a CGL using an Ant Colony Optimization algorithm, reducing the sequencing costs by around 50% in average. However, in the last years, the development of new steel grades and the necessity of reducing the stock levels in this finishing line have brought an important increase in the number of sequencing constraints, challenging feasibility and the algorithms in use. This work provides a set of 30 challenging instances of the problem (some of them have proved to be of high difficulty for ACO), analyzes the performance on them of two ACO variants and introduces an alternative algorithm able to improve such performance.

Finally, there exists others works focused on real planning and scheduling cases in other fields such as the pharmaceutical and chemical industry (Stürtz & Marchetti, 2020), automotive industry (Gnonia et al., 2003) , plastic injection modelling industry (Klement et al, 2021) and construction industry (Ghiyasinasab, Lehouxa, Ménardb, & Cloutier, 2020).

3. Problem definition

3.1. Context of the real case

The production of steel involves several processes to transform the raw materials into final steel products such as coils or bars. First, the iron ore is converted into liquid iron in the blast furnace, and the liquid iron is converted into liquid steel in the converters. Then, in the continuous caster, the molten steel is transformed and cut into solid semi-products, called slabs in the production of flat products. Next, these slabs are rolled at the hot strip mill to obtain coils of steel with the required dimensions. These coils are the final format provided to the client but, depending on final order specifications, they must go through some of the finishing processes such as pickling, tempering, tinning, annealing or galvanizing.

This paper is focused on the scheduling a continuous galvanizing line (CGL) of a Spanish Steel Company, where the steel is coated with a zinc layer to protect it against air and moisture (Fernández et al. 2014). It is a complex continuous process in which coils pass through different phases (accumulator, furnace, zinc pot, etc.) and it is very sensitive to its ordering. That is why the scheduling of the CGL is a very important task in the steel industry. Since the process is continuous and cannot be interrupted, the tail of the coil being processed is welded to the next coil. For the line, it can be seen as an infinite strip, but its properties and characteristics change at some points.

Once the coils are welded, they go towards the accumulator that allows to change the speed of the line without running out of coils. The next stage is the furnace, where the coils are heated until they reach its target temperature. This temperature mainly depends on the steel composition, thickness and width. Afterwards, the strip is immersed into a zinc pot, which is followed by an air knives system that provides each coil which the required zinc coating weight uniformly spread.

Normally, the coils are grouped in campaigns with similar properties and due dates. This phase is done by the human experts before the scheduling process, since doing both phases together increases the complexity. The same campaign may last for several days, while the sequencing of coils to be finished is defined daily. This is the origin of the problem studied on this paper. Every day, the schedulers manually select a set of coils to be produced. They do it following their expertise and the coils due dates. Then, the coils have to be sequenced with minimum cost (i.e. minimum losses of strip meters). It is possible that, since they do the selections as a separated phase and there exists technical constraints, the set of coils is not *sequenceable* (i.e. there is not a Hamiltonian path that can visit all coils just once). Every time two sequenced coils cannot be produced together, an extra coil with no client should be produced in between to avoid the forbidden transition, being its cost huge in comparison to the others since that whole coil may be sold as scrap at a much lower price. Furthermore, more than one extra coil may be required to solve the forbidden transition. If an infeasible sequence is processed, there is a risk of strip breakage and the cost would be much higher due to the unproductive days required to restart the line (cooling down the furnace, removing the broken strip, heating up again the furnace and resuming production).

Hence, after the selection of the coils, the sequencing might face two different situations. The first situation is that the selected coils are not sequenceable, and then the sequencing should provide an ordering with the minimum number of constraints and the minimum possible cost. But, if the selected coils are sequenceable, the solution approach must be able to find a low-cost feasible solution. The latter situation is the aim of this work and it is not a trivial problem, it is equivalent to finding a

minimum cost Hamiltonian path, which is known to be an NP-hard combinatorial problem.

3.2. Definition

In order to provide a formal description of the problem, we can define the problem in a similar way to the SOP or the ATSP: given a directed weighted graph G = (V, A), being A the arc set and V the node set, find the minimum-cost Hamiltonian path (see Fig. 1). The node set V corresponds to the set of n coils to be sequenced ($|V| = n$). The arc set A represents the arcs that connect each pair of distinct nodes. Each arc(i, j) between two coils i and j has a weight c_{ij} associated, which represents the cost of producing those two coils together. The problem is not symmetric: the arc cost c_{ij} may be different to c_{ji}. Depending on the coils properties and technical limitations of the line, an arc weight c_{ij} can represent a cost or an ordering constraint:

- Cost, $c_{ij} \geq 0$, $c_{ij} \in \mathbb{R}$. In this case, c_{ij} represents the cost on processing coil i right before coil j. The exact value of these costs depends on several cost functions that take into account the coil properties and the impact of the transitions in the CGL process. The model used to obtain these costs for each pair of coils is beyond the scope of this paper, and thus we do not describe it in detail. Usually, these costs are translated into meters of strip lost because they do not meet the client quality requirements. As an illustrative example of what these costs represent, each coil requires a target temperature at the furnace. This temperature depends on the coil width, thickness and steel grade, among others. When the next coil enters the furnace, the temperature should change to the new target. Due to the inertia of the system, the higher the differences between the properties of two consecutive coils are, the higher the time required for reaching the new target temperature. Since the line does not stop, all the meters that passed through the furnace during the time the temperature was not at its target might not meet the quality requirements, and those meters cannot be sold. Other causes of losing strip meters are differences on the coating weight at the zinc pot or widths differences at the air knives systems. These costs can range from zero (i.e. the properties of two consecutive coils are identical) to any other $c_{ij} \in \mathbb{R}$ obtained by the cost model but, in any case, those transitions are acceptable by the line managers.
- Constraint, $c_{ij} = -1$. It means that coil i cannot be scheduled right before coil j (i.e. they cannot be welded together), but coil i can be processed at any other position before coil j. This is different to the ordering constraints of the SOP. Usually, there are two main reasons for a constraint between two coils. One reason is that certain coils or steel grades are not weldable. The other situation is when the difference between the values of two coils properties exceeds a certain threshold, defined by the expert schedulers, and there exists risk of strip breakage. If a feasible sequence (i.e. a sequence without any constraint) is not found, then a coil with no client assigned must be produced in between, or sometimes more than one. The cost of doing it belongs to a different cost range to the sequencing costs, and thus the reduction of the number of constraints plays a much more important role than the reduction of the sequencing costs. Once the minimum number of constraints is found, then the cost should be reduced.

It is difficult to know in advance if a selected group of coils is sequenceable or not. That is equivalent to determine if, given a graph, there exists a Hamiltonian path or not, what is itself a NP-hard problem. Normally, the line schedulers do the selection guided by the coils dues dates, and this may harm the sequenciability of the selected coils. If the selection is not sequenceable, then providing a sequence with one or multiple constraints may be acceptable since, if that is the minimum possible number of constraints, it is still the best sequence given that selection. If the selection is sequenceable, the algorithm must assure a feasible solution. But in complex selections like some of the instances introduced here, this is not always the case. That would not be acceptable since the cost of an unfeasible solution is much higher, and hence it is fundamental and the aim of this work to find an algorithm that, for sequenceable selections, will always provide a low-cost feasible solution.

3.3. The instances

Finding an algorithm that can guarantee to find a low-cost feasible solution, if it exists, in a reasonable amount of time is not a trivial task. For testing the algorithms, we first needed some instances of the problem that we already knew to be sequenceable. We took 30 different real daily schedules processed in the past at the studied CGL and obtained their cost matrix using the same cost model. That cost matrix is the only input of the problem. For an instance of n coils, the cost matrix will be a $n \times n$ matrix where a value (i, j) represents the cost c_{ij} of sequencing coil i right before coil j. Each instance is named as "*cgl*" followed by its size n

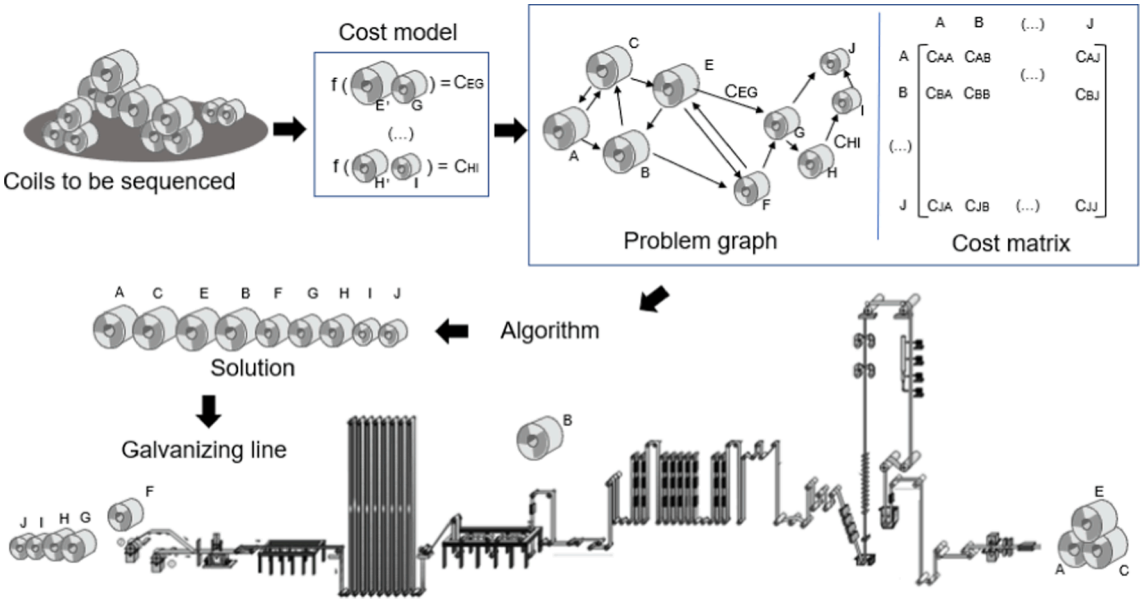

Fig. 1. Description of the problem and the context of the real case.

(i.e. the total number of coils to be sequenced). The size of the instances ranges from 17 coils to 114 coils. All instances can be freely accessed in the following repository.[1]

4. Algorithms

In this section we explain the algorithms used in this study. We implemented two different Ant Colony Optimization (ACO) algorithms: an Ant System algorithm (AS) and an Ant Colony System algorithm (ACS). These two well-known state-of-the-art algorithms were successfully applied to solve famous combinatorial optimizations problems such as the TSP (Dorigo et al. 1991, Dorigo & Gambardella, 1997) or the SOP (Gambardella & Dorigo 2000). Additionally, we developed a new hybrid algorithm to improve their performance.

The reason for using these ACO algorithms is that they have been applied for the sequencing of this finishing line for the last years with very good results (Fernández et al, 2014). Nevertheless, mainly due to the development of new steel grades and the necessity of reducing the stock levels, the complexity of sequencing the line has remarkably increased, making more difficult to the current versions of the algorithms to find feasible sequences. It finally led us to the development of an enhanced AS hybrid version introduced at the end of this section, able to solve these feasibility issues.

One of the advantages of constructive metaheuristics, such as the ACO algorithms, to schedule the finishing lines of the plant, is the fact that they facilitate the handling of multi-coil constraints sometimes encountered. Although in this particular galvanizing line only node-to-node or *transition* constraints are present for the current scheduling rules, in many galvanizing lines and in other finishing lines (continuous annealing, tinning, pickling, etc.) some constraints are related to multiple nodes and/or depend on the rest of the sequence (non-strict grouping constraints, coil properties limited by their previous evolution, coils that should be scheduled between certain positions/time windows, etc.). This kind of constraints can be handled more easily in the constructive metaheuristics than in other methods, such as search algorithms based on k-opt moves or the non-constructive metaheuristics (Genetic Algorithms, Tabu Search, etc.), where time-consuming feasibility checks would be constantly required after each movement/recombination, or special strategies should be defined for each particular line. Another reason is that, from time to time, a new constraint should be added to the model (i.e. a new product will be produced), and we found it easier to add new constraints to a constructive metaheuristic than modifying the whole strategy to perform movements or recombinations.

Algorithm 1. (*ACO algorithms basic structure*)

```
1  Set the parameters
2  Initialize the pheromone values
3  while (termination criteria not met) do
4    PerformAntsSequenceConstruction
5    UpdatePheromoneValues
6  end while
```

ACO is a swarm intelligence metaheuristic inspired by the foraging behavior of the ants and how they can coordinate themselves by *stigmergy*, an indirect way of communication based on modifications of the environment. Ants can deposit pheromone on their paths to increase the probability that other ants will follow the same trial, being able to achieve self-organization as a colony. Most of the theoretical information and formulae about ACO discussed in this section were taken from the complete ACO-book by Dorigo and Stützle (2004). All ACO algorithms share the same basic structure based on an initialization phase followed by a main loop with a construction phase and a pheromone update phase (see Algorithm 1). From that basic skeleton, many successful ACO variations have been developed and they mainly differ in how construction decisions are taken and how the pheromone is updated.

In order to apply the ACO algorithms to the presented problem, we modelled it as an ATSP, assigning a high cost (higher than the rest of the transition costs) to the forbidden arcs. This may seem to be the most straightforward way. But as we will see later on, simply managing the constraints as normal arcs with a high cost associated sometimes generates issues during the search, namely difficulties for finding feasible sequences.

4.1. Ant colony system

Due to its successful application to other combinatorial problems, we first implemented an ACS. It is an extension of the AS that includes three different features to improve its performance (Dorigo and Stützle, 2004): a more aggressive action choice rule strongly biased by the accumulated experience, a pheromone evaporation/deposit only performed by the best-so-far ant and the update of pheromone every time an arc is chosen during the construction (local update). In the ACS, the construction of each ant is guided by the following formula:

$$j = \begin{cases} argmax_{l \in N_i^k} \left\{ \tau_{il} [\eta_{il}]^\beta \right\}, if q \leq q_0 \\ J, otherwise \end{cases} \quad (2)$$

where J is a random variable obtained from the probability distribution given by classical *random proportional* rule of the original AS, but with $\alpha = 1$:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta}, if j \in N_i^k \quad (3)$$

Initially, a random start node or coil is assigned to each of the m ants. At each construction step, an ant k located at node i will choose the next coil j from the set of non-selected coils N_i^k(Ec.2). With probability q_0 the selection is made *greedily*, since the next node would be the one with better product $\tau_{il}[\eta_{il}]^\beta$: the choice with the highest amount of pheromone and heuristic information (i.e. lower cost), favoring exploitation. On the other hand, with probability (1 - q_0) the next node would be chosen according to the probabilistic action choice rule (Ec.3), where τ_{ij} is the amount of pheromone on the arc(i, j) and η_{ij} its heuristic information. β is the parameter that controls the influence of the heuristic information in the construction process, while α controls the influence of the pheromone ($\alpha = 1$ in ACS). The heuristic information of an arc(i, j) is constant during the iterations and it is related to the cost of sequencing coil i right before coil j, $\eta_{ij} = 1/c_{ij}$. When the arc(i, j) represents a forbidden link (c_{ij} = -1), we assign to that arc a high cost BC some orders of magnitude greater than the average transition costs obtained by the cost model. This makes very small the heuristic information of choosing an arc that represents violating a constraint, and thus makes very low probability of selecting it.

In the ACS, there are two different mechanisms for updating the pheromone values: the global pheromone update and the local pheromone update. The global pheromone update is performed at the end of each iteration (after all the ants have constructed their paths) and it is done only by the best-so-far ant (Ec.4).

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \rho \Delta \tau_{ij}^{best}, \forall (i,j) \in S^{best} \quad (4)$$

The best-so-far ant is the one that, until that moment, has found the best solution S^{best} (i.e. the sequence with the minimum cost C^{best}). This ant deposits to all the arcs of its sequence an amount of pheromone equal to $\rho \Delta \tau_{ij}^{best}$, where ρ is a parameter that represent the pheromone evaporation and $\Delta \tau_{ij}^{best} = 1/C^{best}$. At the same time, evaporation is performed

[1] https://data.mendeley.com/datasets/v357z2ncbh/2

on S^{best} arcs controlled by ρ. If a feasible solution has not been found yet, the best-so-far solution will be the sequence with the lower number of constraints, but its cost will still be very high because it will have at least one cost $c_{ij} = BC$. When this happens, the solution with the lower number of violated constraints S^{best} will have more pheromone than the others, but the pheromone deposited after each iteration will be very low to avoid stagnation and premature convergence on an infeasible solution.

On the other hand, the local pheromone update is performed by each ant every time an arc is chosen during the sequence construction:

$$\tau_{ij} \leftarrow (1-\xi)\tau_{ij} + \xi\tau_0 \tag{5}$$

where ξ is a parameter that normally equals to 0.1 and τ_0 is the initial value of the pheromone. During the initialization phase of the algorithm, the same value τ_0 is assigned to each arc in the pheromone matrix. A good option for this initial pheromone value is $1/nC^{nn}$, being n the size of the instance and C^{nn} the cost of a sequence obtained using the nearest-neighbor approach (Dorigo and Stützle, 2004).

Focusing on this case study, the nearest-neighbor solution tends to contain one or multiple constraints, computed as penalties or high costs. Knowing this, for the initialization of the pheromone we use $1/nC^{nn^*}$, where $C^{nn^*} = C^{nn}/n_c^{nn}BC$ and n_c^{nn} is the number of violated constraints in the nearest-neighbor solution. In other words, C^{nn^*} is the cost of the nearest-neighbor solution without considering the high costs associated to the violated constraints. This generates that the initial value of the pheromones would probably be higher than the expected amount to be deposited in the first iterations.

According to Dorigo and Stützle (2004), if the initial amount of pheromone is higher than the expected quantity of pheromone deposited in one iteration, the first iterations are not influenced by the pheromone and the decisions of the next ants until the initial values of the pheromones reach the adequate levels are guided only by the heuristic information, *losing* the first iterations. In our problem, this effect is interesting. For example, let́s assume the usual situation of a problem instance in which the first ants, mainly guided by the heuristic information (Ec.1), are not able to find feasible solutions. If initially the value of the pheromone is low (as it would be if we use $1/nC^{nn}$), then the solution with the smaller number of violated constraints will have more pheromone than the others and it exists the risk of biasing the search to the region of that infeasible solution and finally not finding a feasible one. But, on the contrary, if the initial value of pheromone is higher (using $1/nC^{nn^*}$, $C^{nn^*} \ll C^{nn}$), the pheromone deposited by infeasible solutions during the first iterations will not have much effect on the next iterations and the next ants will better explore the search space. Obviously, if there is not a feasible solution, we prefer a solution with only one constraint than other with two and, if that is the case, as the initial pheromone is reduced then the solution with fewer constraints will start to have much more pheromone than the others and probably the final solution will be the one with lower number of constraints. But, if there exists a feasible solution, we do not want an infeasible solution (even being the best of the iteration) to bias the search in the first iterations, reducing the possibility on finding the feasible one. This is the issue we want to avoid with our different initialization method, after first getting poor results with the original initialization.

4.2. Ant system

After some tests of the ACS showing not very good results, we decided to also implement an AS and compare their performance. The main reason was that we noticed that greedy cost construction decisions may difficult the task of finding feasible solutions.

Indeed, oftentimes the selection of low-cost transitions during the construction can generate feasibility problems in the following steps. For example, let́s suppose that one coil a can link to just two coils b and c, with $c_{ab} \ll c_{ac}$. There are multiple coils that can be sequenced right before coil b, but the only coil that links before coil c is coil a. Once coil a is chosen during the construction, if the decision for the next coil is highly influenced by the costs, then coil b will be selected and automatically the sequence will contain at least one constraint (because there is no other coil that links to coil c). Many of the instances of the problem have similar but more complex relations between cost and feasibility, which makes the search even harder.

The ACS, because of the aggressive action choice rule in which with probability q_0 the option with the highest $\tau_{il}[\eta_{il}]^{\beta}$ value is selected, is prone to misguide the search towards low cost arcs that eventually lead to unfeasible solutions, especially during the first iterations where the influence of the pheromone is lower. But in the AS, the construction decisions are more open to exploration since they are always guided by the random proportional rule (Ec.3), less greedy, and thus there are more choices of avoiding situations like the one explained above.

In the AS, differently to in the ACS, the pheromone update is performed by all ants after each iteration. First, all pheromone values are reduced according to the following equation:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij}, \forall (i,j) \in A \tag{6}$$

After the evaporation, all arcs belonging to the sequence constructed by each ant are reinforced with an amount of pheromone $\Delta\tau_{ij}^k$ inversely proportional to the cost C^k of its sequence S^k (Ec.7 and Ec.8). The arcs that were not chosen by any ant do not received any amount of pheromone during that iteration.

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^k, \forall (i,j) \in A \tag{7}$$

$$\Delta\tau_{ij}^k = \begin{cases} \dfrac{1}{C^k}, if arc(i,j) belongs to S^k \\ 0, otherwise \end{cases} \tag{8}$$

For initializing the pheromone, we use the approach suggested by Dorigo and Stützle (2004) of using m/C^{nn} but again with C^{nn^*} instead of C^{nn} as explained for the ACS, and for the same reasons.

4.3. Hybrid version of the ant system

4.3.1. Motivation

As we will see in the results and as anticipated, the AS, with a less greedy construction mechanism, achieves a better performance for this problem in terms of finding feasible sequences compared to the ACS. Nevertheless, the AS still faces difficulties in finding feasibility when the costs distribution drives the search to shortcuts where eventually adding a constraint becomes unavoidable. This has been the motivation for the developing our Ant System - Interval Reconstruction algorithm (AS-IR).

Let us illustrate the issues found in ACS and AS performance with two different examples, exaggerated and simplified to facilitate the explanation, but present in similar forms in some of the studied instances.

Looking at Fig. 2, we can clearly differentiate two groups of coils. We refer to Nodes 0, 1, 2 and 3 as Group A, and Nodes 4, 5, 6 and 7 as Group B. As it can be seen from the cost matrix shown in Fig. 2, first all nodes from Group A should be sequenced together, and then all nodes from Group B. Nevertheless, with that cost distribution where going from a Group A node to a Group B node has cost zero, the ACS and the AS will probably face difficulties in finding that ordering because their construction decisions are strongly biased by the costs.

For example, let́s assume that Node 0 has been selected at some point during the construction of the sequence, and the rest of nodes [1, 2, …, 7] are available (i.e. they have not been added to the sequence yet). In order to get a feasible sequence, only Node 1, 2 and 3 can be selected as next nodes. If any other node from Group B is selected instead, automatically at least one constraint will be generated: none of the nodes from Group B can link back to nodes from Group A. When the construction is at Node 0, all the nodes from Group B (i.e. wrong choice) link with a very low cost, while the nodes from Group A have a much higher

	Node 0	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6	Node 7
Node 0	100	100	100	100	0	0	0	0
Node 1	100	100	100	100	0	0	0	0
Node 2	100	100	100	100	0	0	0	0
Node 3	100	100	100	100	0	0	0	0
Node 4	-1	-1	-1	-1	100	100	100	100
Node 5	-1	-1	-1	-1	100	100	100	100
Node 6	-1	-1	-1	-1	100	100	100	100
Node 7	-1	-1	-1	-1	100	100	100	100

Fig. 2. Example to illustrate adverse cost distributions.

cost (i.e. the correct choice). Hence, a construction mechanism as the one used in ACS and AS, that gives more probability to the nodes with lower costs, has a high probability of making a wrong choice and not finding a feasible sequence. This effect is amplified when the algorithm should select nodes with higher cost many consecutives times (i.e. as in this case, all the nodes from Group A should be sequenced together before Group B nodes).

Another interesting situation that we noticed is when, to be able to solve or avoid just one single constraint, many arcs should be modified -sometimes even the whole sequence. Let us see this in the next example.

The sequence [1, 3, 5, 7, 2, 4, 6, 8] shown in Fig. 3 has just one constraint, between nodes 7 and 2. It may seem *close* to be feasible (in terms of movements) because it has just one constraint but, looking at the cost matrix, it can be seen how all the arcs should be changed to fix the constraint. The only feasible sequence for this extreme example is in strict growing order: [1, 2, 3, 4, 5, 6, 7, 8]. For constructive algorithms like the AS and the ACS, the probability of selecting any of the adjacent nodes is the same since all them have equal cost. If during the first iterations the ACO algorithms do not find the correct sequence -what is very likely since the correct decision should be made several consecutive times without a cost distribution that helps the search-, pheromone will be deposited in unfeasible sequences like the one shown in Fig. 3, and the chances to explore a completely different sequence are gradually reduced with the iterations.

A simple test shows that an instance with a cost matrix as described above and of size only 40 nodes is not resolved to feasibility by ACS nor by AS.

Thus, we have seen that the ACO algorithms, specially the AS, have many positive aspects that make them desirable for scheduling the finishing lines, but fail in extreme situations like the ones explained above. This is the reason why we needed to improve their performance and ability to find feasible sequences. Finally, we came out with a new *local search* that we named as Interval Reconstruction.

4.3.2. The ant system with interval reconstruction

We call Ant System with Interval Reconstruction (AS-IR) to an enhanced version of the AS in which we introduce a sequence improvement mechanism, the Interval Reconstruction (IR). The IR was initially developed to solve constraints, but it also can be used to reduce costs. It works selecting two *intervals* or *windows* of nodes from a given sequence, and then rearranging those nodes with the aim of improving the sequence.

Algorithm 2. (*Ant system with interval reconstruction*)

```
1  Set the ACO and IR parameters
2  Initialize the pheromone values
3  while (termination criteria not met) do
4    PerformAntsSequenceConstruction
5    PerformIntervalReconstructionLocalSearch
6    UpdatePheromoneValues
7  end while
```

To our understanding, we have not seen a similar *constructive local search* in the literature, though in part the development of this improvement mechanism was inspired by the Greedy Randomized Adaptive Search Procedure (Feo & Resende, 1995) and the Iterated Greedy (Ruiz & Stützle, 2007). We refer to the IR as a *local search* mainly

	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6	Node 7	Node 8
Node 1	-1	0	0	0	0	0	0	0
Node 2	-1	-1	0	0	0	0	0	0
Node 3	-1	-1	-1	0	0	0	0	0
Node 4	-1	-1	-1	-1	0	0	0	0
Node 5	-1	-1	-1	-1	-1	0	0	0
Node 6	-1	-1	-1	-1	-1	-1	0	0
Node 7	-1	-1	-1	-1	-1	-1	-1	0
Node 8	-1	-1	-1	-1	-1	-1	-1	-1

Sequence:

Node 1 → Node 3 → Node 5 → Node 7 ✖ Node 2 → Node 4 → Node 6 → Node 8

Fig. 3. Example to illustrate a sequence where all the arcs should be changed to fix one constraint.

because we use it as an improvement mechanism applied to complete sequences constructed by the ants, and because it can be embedded in any other metaheuristic, but it can expand the search more than locally.

The IR is applied at the end of each iteration, after all the ants have constructed their sequence and before the pheromone update (Algorithm 2). Although different strategies can be used (apply it only to the best so far ant, to some of the best ants of the iteration, etc.), we decided to only apply it to the best ant of the iteration due to performance reasons.

The IR gets as input a complete sequence S^{best} built by the best ant and tries to improve it several times. If no improvement is found, the same input sequence is returned. First of all, the IR looks for all the arcs of the input sequence (S^{best}, Algorithm 3) that represent a constraint violation (forbidden arcs). At each improvement try, one such target arc is selected, around which one of the intervals for partial reconstruction will be defined. With a target arc chosen from the list of forbidden arcs in S^{best}, the goal of the IR is to solve that constraint (or reduce the number of constraints at least by one). In case there are not constraints violations in S^{best}, that is, the list is empty, we select as target arc any random arc, and the goal of the IR is now to reduce the cost of the sequence.

Algorithm 3. (*PerformIntervalReconstructionLocalSearch*)

```
1  Get the best solution S^best of the iteration
2  for improvement_try in range max_improvement_tries:
3    target_arc ← SelectTargetArc
4    S^new ← PerformReconstruction(S^best, target_arc)
5    if cost (S^new) < cost (S^best) then:
6      S^best ← S^new
7      break
8  end for
9  return S^best
```

With this approach, we are not treating the constraints as normal arcs with high cost, but instead we give total preference to solving the constraints, targeting directly the forbidden arcs. In addition, to avoid the costs misguide the reconstruction procedure, when we are targeting a constraint (that is, if the list of forbidden arcs is not empty), the reconstruction procedure does not pay attention to the costs, it only looks at the adjacency. This is a key feature of the algorithm.

Once we have chosen the target arc for a particular improvement try, two windows are defined. One window (W_1) is defined around the target arc, so that the two nodes involved are selected. The other window is selected at any other part of the sequence, randomly chosen. Both windows are controlled by two length parameters, *min_slice_len* and *max_slice_len*, allowing us to decide how much of the sequence will be modified. If those parameters are set to low values, the IR will perform a local refinement, and if they are high, the windows reconstructed will be bigger, and the change in the sequence could be radical, what sometimes may be interesting. The length of each window is randomly generated between *min_slice_len* and *max_slice_len*.

Algorithm 4. (*PerformReconstruction(S^{best}, target_arc)*)

```
1  S ← S^best
   Ś, W1, W2 ← SelectIntervals(S^best, target_arc)
2  for reconstruction_try in range max_reconstruction_tries:
3    B = W1 U W2
4    while B not empty:
       Ś, B ← InsertOneNode(Ś, B)
5    if cost (Ś) ≤ cost (S) then:
6      S ← Ś
7      break
8  end for
9  return S
```

The two windows define two subsets of nodes, W_1 and W_2, comprised

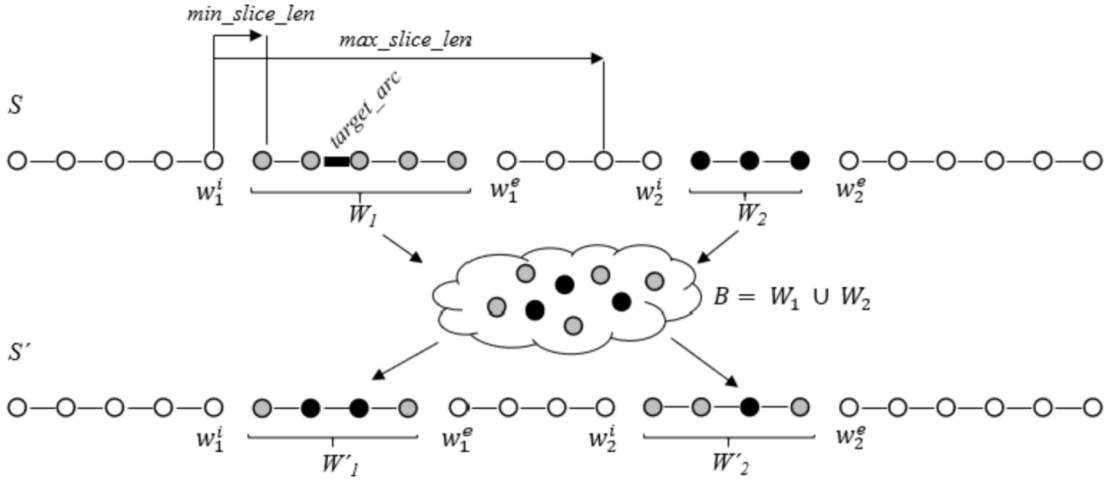

Fig. 4. One reconstruction try of the Interval Reconstruction.

between the initial (w_1^i, w_2^i) and end positions (w_1^e, w_2^e) of each window. Those nodes will be removed from the sequence and then reinserted trying to build two new windows, $\acute{W}_1$ and $\acute{W}_2$, with lower cost or lower number of constraint violations (see Fig. 4).

The reinsertion procedure works by inserting one by one the nodes from the set $B = W_1 \cup W_2$. Before inserting a node, one of the two new windows is selected according to a given probability ($p = 0.5$). To decide the next node to be added to that window, we first look for the set of nodes that has not been selected yet and are adjacent to the last added node i (D_i). If it is the first node to be added to the new window, we look for nodes adjacent to the node positioned at w_1^i or w_2^i. We say a node j is adjacent to node i if they can be sequenced together ($c_{ij} \neq$ -1). When the set of adjacent nodes is not empty ($D_i \neq \varnothing$), the next node j will be selected from D_i.

If we are targeting a forbidden arc (i.e. trying to solve one constraint), the next node is selected randomly form D_i, having all the candidate nodes the same probability (Ec.9). This way, to avoid situations as the ones presented in Section 4.3.1, we do not allow the decision be influenced by the costs, increasing the exploration and avoiding potential biases of the AS construction mechanism and the accumulated pheromone.

$$p_{ij} = \frac{1}{size(D_i)}, if j \in D_i \tag{9}$$

On the other hand, if the target arc is not a forbidden arc (i.e. the input sequence is already feasible), the next node is selected according to a probability distribution that linearly bias the selection towards the nodes that add lower costs (Ec.9).

$$p_{ij} = \frac{\frac{1}{c_{ij}}}{\sum_{l \in D_i} \frac{1}{c_{il}}}, if j \in D_i \tag{10}$$

If at some construction step the set of adjacent nodes is empty ($D_i = \varnothing$), we select a random node from the set of non-adjacent nodes, directly adding an unavoidable violated constraint to the sequence.

Once a node is reinserted into one of the new windows, it is removed from B, and the same process is repeated until all the nodes from B have been inserted (Algorithm 4). Finally, we compute the two *closing costs*, that are the cost of linking the last added node of each window with the node located at w_1^e or w_2^e. The reconstruction procedure is very efficient since we only compute partial costs (removed arcs vs added arcs), and only requires quick access to the cost matrix. This allow us to try multiple reconstructions.

For a given W_1 and W_2, we allow several tries to get a better solution (*max_reconstruction_tries*). If a better solution is found, both the reconstruction tries (same target arc and windows) and the improvement tries (different target arcs) are interrupted, and the new solution is returned to the AS. To know if the new windows are better, we first compute the number of violated constraints and the cost of W_1 and W_2, and then those of $\acute{W}_1$ and $\acute{W}_2$ during the reconstruction. If the number of constraint violations is reduced, the new sequence is considered better. In the case of equal number of violated constraints, the new sequence is considered better if the cost is reduced.

The use of two windows is, together with the dismissal of costs to drive the search over unfeasible sequences, another key feature of the algorithm, allowing a wide exploration of the search space. Using only one window, the interval reconstruction would be restricted to simply shuffle certain nodes in a part of the sequence, whereas the usual case is that other nodes from other parts of the sequence may be fundamental to resolve the constraint. With two random windows, the exploration is not limited anymore in this regard, allowing a much broader search.

5. Results

The main goal of this work was to assess the effectiveness of the different algorithms in finding feasible and low-cost solutions in a reasonable amount of time for industrial scenarios. For the assessment, we ran the three algorithms (ACS, AS and AS-IR) 30 times for each of the real instances explained in Section 3, in order to obtain representative results. These instances were known to be feasible in advance, so we could determine the success rate of the algorithms in finding feasible solutions. We limited the execution time to 120 s for each run, which is a decent amount of time for the scheduling crew to run the algorithm on their laptop.

For the galvanizing line, the most important target is to get a feasible sequence of the coils to be produced each day, which in fact is the main challenge. Once a feasible sequence is obtained, then the lower the costs, the better. Hence, we divided the assessment in two phases: first, we compared the performance of the algorithms in finding feasible solutions, and then we analyzed their performance in reducing costs.

For the ACO algorithms, we used the parameters setting suggested in Dorigo and Stützle (2004). In our implementation of the ACS we used $q_0 = 0.9$, $\beta = 2$, $\rho = 0.1$, $\xi = 0.1$ and $m = 10$, being m the number of ants. For the AS, we used $\alpha = 0.1$, $\beta = 2$, $\rho = 0.5$ and $m = n$, being n the size of the instance. The parameters used for the IR in the AS-IR were: *max_window_len* = $n/3$, *min_window_len* = 0, *max_improvement_tries* = 10 and *max_reconstruction_tries* = 30.

5.1. Feasibility performance

For the feasibility analysis, we used only the nine instances of Table 1. For the rest of the instances, finding feasible solutions was not an issue for any of the three algorithms. Some instances are harder than others in terms of feasibility depending on the attributes of the set of coils to be scheduled. Sometimes, most of the coils have similar attributes and they link together easily. In other cases, the set of coils to be

Table 1
Feasibility Analysis.

Inst.	Alg.	n_unfeas	Success (%)	Inst.	Alg.	n_unfeas	Success (%)
cgl_51	AS	0	100%	**cgl_72**	AS	1	96.7%
	AS-IR	0	100%		AS-IR	0	100%
	ACS	21	30%		ACS	7	76.7%
cgl_48	AS	4	86.7%	**cgl_78**	AS	0	100%
	AS-IR	0	100%		AS-IR	0	100%
	ACS	27	10%		ACS	25	16.7%
cgl_60	AS	0	100%	**cgl_33**	AS	27	10%
	AS-IR	0	100%		AS-IR	0	100%
	ACS	26	13.4%		ACS	27	10%
cgl_26	AS	3	90%	**cgl_73**	AS	0	100%
	AS-IR	0	100%		AS-IR	0	100%
	ACS	28	0.7%		ACS	5	83.4%
cgl_44	AS	0	100%				
	AS-IR	0	100%				
	ACS	5	83.4%				

scheduled are very different and it is really difficult just to find a feasible sequence.

Ideally, if an algorithm is run 30 different times on the same instance, it should find a feasible sequence all the times (if it exists). This is what we call success (Table 1), and this ideal situation would have a success rate of 100%. If the success rate of an algorithm on a particular instance is lower than 100%, it means that it exists a chance of the operator running the tool for a sequenceable set of coils and not getting a feasible schedule, due to stochastic nature of the studies algorithms

As it can be seen from Table 1, the ACS with the parameter values adopted in Dorigo and Stutzle (2004) is the algorithm with the worst feasibility performance. The maximum success rate of the ACS was 83.4% in *cgl_44*, and it got a success rate lower than 20% in five out of the nine instances. The AS obtained much better results with a success rate over 85% in eight out of the nine instances (a 100% in five of them). This result goes against other theoretical problems in the literature where the ACS tends to perform better than the AS, especially for larger instances (Dorigo and Stützle, 2004), but it is not the case in the not so big instances studied from this problem, where finding feasible solutions is crucial.

One reason for the poor feasibility performance of the ACS in this problem may lay in the fact that its construction decisions are strongly guided by the costs, selecting with probability $q_0 = 0.9$ the node with lower cost. This fact, together with the more aggressive action choice rule strongly biased by the accumulated experience, restricts its exploration capacity, and if it does not find a feasible sequence during the first iterations, it is unlikely that it will do it latter no matter for how long it is run.

Another reason for the AS having a better performance than the ACS can be that the AS is more open to exploration due to its construction mechanism (the next node depends purely on the probability distribution given by classical *random proportional* rule, Ec.3) and the way the pheromone is updated (all the ants deposit pheromone proportionally to the cost of its sequence, while in the ACS the pheromone deposit is done only by the best-so-far ant).

Nevertheless, the AS still obtained a success rate lower than the 100% in four instances, what means that it exists a probability of returning an unfeasible sequence for a set of sequenceable coils. This probability is extremely high for *cgl_33* instance, for which both the AS and the ACS obtained a success rate of only 10%.

Although it is out of the scope of this paper to go into a deep analysis of the cost matrices and the adjacency graphs underlying these instances, we can take a quick look to the cost matrix of *cgl_33* to see how similar situations and cost distributions to the ones commented in Section 4.3.1 make this instance very hard for ACO algorithms. Just to highlight some details that can be easily seen from Fig. 5, among others:

- Node 4 is a very special coil because it only links before Node 5. At each iteration, each ant starts at a random node. If at some point Node 4 is selected, and Node 5 was already added to the sequence, it automatically generates one constraint violation.
- Another difficult situation that can be easily detected is related to Nodes 0, 1 and 2. These three nodes should be scheduled together in order to get a feasible sequence. If the last added node was Node 0, and Nodes 1 and 2 have not been selected yet, it is mandatory to select one of them to avoid a future constraint violation. If any other adjacent node is selected instead right after Node 0 (Node 3, 5, 6, 7, …, 12), automatically a constraint violation is generated since none of the rest of the nodes link before Node 1 or Node 2.
- Leťs suppose that the last added node was Node 0, and Node 1 was already selected. The only right choice for next node is Node 2, but is has a cost of 1247. On the other hand, selecting as next node Node 3 has a cost of 383 (higher probability for ACO algorithms), but it will generate one constraint violation (no other available node links before Node 2)

Some input instances presented in this work have cost distributions showing similarities with the one explained above, making likely for decisions guided by the costs to generate feasibility issues - as we have assessed in our tests. Since the ACO algorithms do not take totally greedy decisions (i.e. as it would do the Nearest Neighbor heuristic), they may at some point make the correct decision by chance, selecting a right node with a less cheap cost. But when such many unlikely decisions should be done along the construction, the ACO algorithms can eventually fail in

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
0	-1	179	1247	383	-1	566	1255	1255	1329	1255	1255	1298	1277	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	208	-1	1227	561	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
2	1284	1232	-1	1412	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
3	426	-1	-1	-1	835	224	914	914	999	914	914	967	946	332	332	449	449	449	449	449	449	449	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
4	-1	-1	-1	-1	-1	414	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
5	-1	-1	-1	420	249	-1	998	998	1072	998	998	1041	1019	724	724	1122	1122	1122	1122	1122	1122	1122	-1	1069	-1	-1	-1	-1	-1	-1	-1	-1	-1
6	1420	-1	-1	1026	1415	994	-1	0	74	0	0	42	21	1133	1133	1361	1361	1361	1361	1361	1361	1361	4164	1333	-1	-1	-1	-1	-1	-1	-1	-1	-1
7	1420	-1	-1	1026	1415	994	0	-1	74	0	0	42	21	1133	1133	1361	1361	1361	1361	1361	1361	1361	4164	1333	-1	-1	-1	-1	-1	-1	-1	-1	-1
8	1516	-1	-1	1114	1487	1064	74	74	-1	74	74	21	52	1194	1194	1295	1295	1295	1295	1295	1295	1295	3948	1278	-1	-1	-1	-1	-1	-1	-1	-1	-1
9	1420	-1	-1	1026	1415	994	0	0	74	-1	0	42	21	1133	1133	1361	1361	1361	1361	1361	1361	1361	4164	1333	-1	-1	-1	-1	-1	-1	-1	-1	-1
10	1420	-1	-1	1026	1415	994	0	0	74	0	-1	42	21	1133	1133	1361	1361	1361	1361	1361	1361	1361	4164	1333	-1	-1	-1	-1	-1	-1	-1	-1	-1
11	1494	-1	-1	1085	1479	1040	42	42	21	42	42	-1	31	1188	1188	1318	1318	1318	1318	1318	1318	1318	-1	1329	-1	-1	-1	-1	-1	-1	-1	-1	-1
12	1447	-1	-1	1058	1442	1014	21	21	52	21	21	31	-1	1150	1150	1341	1341	1341	1341	1341	1341	1341	3989	1316	-1	-1	-1	-1	-1	-1	-1	-1	-1
13	-1	-1	-1	311	-1	499	1008	1008	2462	1008	1008	2356	2409	-1	0	128	128	128	128	128	128	128	1863	261	-1	-1	-1	-1	-1	-1	-1	-1	-1
14	-1	-1	-1	311	-1	499	1008	1008	2462	1008	1008	2356	2409	0	-1	128	128	128	128	128	128	128	1863	261	-1	-1	-1	-1	-1	-1	-1	-1	-1
15	-1	-1	-1	440	-1	758	1193	1193	2377	1193	1193	2330	2430	126	126	-1	0	0	0	0	0	0	2130	238	-1	-1	-1	-1	-1	-1	-1	-1	-1
16	-1	-1	-1	440	-1	758	1193	1193	2377	1193	1193	2330	2430	126	126	0	-1	0	0	0	0	0	2130	238	-1	-1	-1	-1	-1	-1	-1	-1	-1
17	-1	-1	-1	440	-1	758	1193	1193	2377	1193	1193	2330	2430	126	126	0	0	-1	0	0	0	0	2130	238	-1	-1	-1	-1	-1	-1	-1	-1	-1
18	-1	-1	-1	440	-1	758	1193	1193	2377	1193	1193	2330	2430	126	126	0	0	0	-1	0	0	0	2130	238	-1	-1	-1	-1	-1	-1	-1	-1	-1
19	-1	-1	-1	440	-1	758	1193	1193	2377	1193	1193	2330	2430	126	126	0	0	0	0	-1	0	0	2130	238	-1	-1	-1	-1	-1	-1	-1	-1	-1
20	-1	-1	-1	440	-1	758	1193	1193	2377	1193	1193	2330	2430	126	126	0	0	0	0	0	-1	0	2130	238	-1	-1	-1	-1	-1	-1	-1	-1	-1
21	-1	-1	-1	440	-1	758	1193	1193	2377	1193	1193	2330	2430	126	126	0	0	0	0	0	0	-1	2130	238	-1	-1	-1	-1	-1	-1	-1	-1	-1
22	-1	-1	-1	-1	-1	-1	2796	2796	5494	2796	2796	-1	5547	576	576	2521	2521	2521	2521	2521	2521	2521	-1	911	499	-1	-1	-1	-1	-1	-1	-1	-1
23	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	264	264	202	202	202	202	202	202	202	1116	-1	252	-1	-1	-1	-1	-1	-1	-1	-1
24	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	406	420	420	-1	-1	-1	-1	-1
25	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	245	-1	3	3	543	-1	-1	-1	-1
26	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	245	0	-1	0	543	-1	-1	-1	-1
27	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	245	0	0	-1	543	-1	-1	-1	-1
28	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	581	-1	-1	-1
29	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	224	224	222
30	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	329	-1	0	0
31	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	329	0	-1	0
32	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	324	0	0	-1

Fig. 5. Cost matrix of *cgl_33* instance.

Table 2
Results of the cost performance experiments.

Inst.	Alg.	C_{best}	C_{avg}	Std. Dev.	Inst.	Alg.	C_{best}	C_{avg}	Std. Dev.
cgl_51	AS	12,677	14374.83	1042.93	*cgl_28*	AS	2856	2877.90	23.99
	AS-IR	**12,670**	**14340.70**	977.05		AS-IR	**2833**	**2833.76**	4.19
cgl_60	AS	10,604	11261.43	507.19	*cgl_81*	AS	6984	8056.76	686.65
	AS-IR	**10,409**	**10892.53**	362.40		AS-IR	**6866**	**7853.40**	689.53
cgl_38	AS	3887	3887	0	*cgl_58*	AS	3652	3661.20	8.41
	AS-IR	3887	3887	0		AS-IR	3652	**3656.06**	6.84
cgl_44	AS	10,542	10690.33	156.74	*cgl_70*	AS	**9848**	11587.53	1660.33
	AS-IR	**10,070**	**10280.10**	79.52		AS-IR	9956	**10912.73**	681.00
cgl_78	AS	11,282	13261.26	1089.77	*cgl_114*	AS	9774	**10552.76**	477.31
	AS-IR	**10,158**	**13050.63**	1114.47		AS-IR	9774	10622.96	466.10
cgl_88	AS	11,020	**11638.10**	560.11	*cgl_107*	AS	**5676**	**6396.16**	364.58
	AS-IR	**10,952**	11735.23	858.25		AS-IR	6103	6503.66	228.16
cgl_45	AS	8275	8817.70	263.70	*cgl_47*	AS	5333	6353.80	545.09
	AS-IR	**8089**	**8436.26**	187.00		AS-IR	**5046**	**6063.23**	539.69
cgl_76	AS	**10,683**	12109.10	844.51	*cgl_51b*	AS	4520	4831.16	188.20
	AS-IR	10,740	**11902.06**	918.38		AS-IR	**4394**	**4761.53**	197.16
cgl_17	AS	4422	4525.56	221.39	*cgl_43*	AS	5372	5539.66	123.91
	AS-IR	4422	**4422**	0		AS-IR	5372	**5510.10**	118.27
cgl_73	AS	6833	7795.03	500.62	*cgl_32*	AS	**4211**	4692.26	353.00
	AS-IR	**6557**	**7574.03**	429.04		AS-IR	4260	**4390.66**	89.51
cgl_70b	AS	5885	6454.16	199.34	*cgl_37*	AS	4883	5358.33	475.52
	AS-IR	**5728**	**6416.33**	245.35		AS-IR	4883	**4968.43**	135.51
cgl_66	AS	8370	**8885.26**	283.46	*cgl_48b*	AS	**4528**	**4778.86**	141.67
	AS-IR	**8327**	8946.13	359.86		AS-IR	**4483**	**4623.16**	105.91
cgl_50	AS	6066	6423.13	307.90	*cgl_57*	AS	7999	**8706.53**	541.01
	AS-IR	**5611**	**6173.23**	278.84		AS-IR	7999	8728.73	527.08

finding a feasible sequence.

These feasibility issues of the ACO algorithms can be solved by adding the IR as a sequence improvement mechanism. When the IR is trying to solve one constraint violation (i.e. the target arc is a forbidden link), the reconstruction decisions are only based on adjacency, and they do not pay attention to the costs. This allows to increase the exploration capacity of the algorithm and the chances to find the correct order of coils. In addition, the simplicity of the IR and its time-efficiency allow to try multiple reconstructions to solve one constraint violation. From Table 1, the AS-IR obtained a 100% of success in all the 30 studies instances, satisfying the main objective of ensuring feasibility when it exists. We find the combination of the AS and the IR a very robust hybrid algorithm in terms of feasibility, that uses the intelligence of the classical AS (pheromone deposit and evaporation) and the ability of the IR to try several recombinations without cost bias.

5.2. Cost performance

Since the feasibility success is considered the most important criteria, we dismissed the ACS for the cost analysis, and we compared the ability to find low-cost sequences of the AS and the AS + IR for the 26 instances for which both algorithms achieved a 100% of feasibility success rate. With this second analysis we wanted to see if the IR also helps the AS to find cheaper solutions, in addition to the contributions in the search of feasible orderings.

Table 2 shows the detailed results of the experiments conducted: the best cost (C_{best}), the average cost (C_{avg}) and the standard deviation (Std. Dev). Before analyzing the results of the AS-IR, it is worth to highlight that, despite of the difficulties explained in the previous sections, when the sequencing of the daily set of coils is not very challenging in terms of feasibility, the AS tends to perform very well and finds low-cost solutions.

The AS-IR obtained a better C_{best} in 15 instances and a better C_{avg} in 20 instances, while the AS outperformed the AS-IR with a lower C_{best} in 4 instances and a lower C_{avg} in 5 instances. Both algorithms obtained the same C_{best} in 7 instances and the same $C_{avg\ t}$ in just one instance. With a C_{best} improvement in the 58% and a C_{avg} improvement in the 79% of the tested instances, the effect of adding the IR to the AS is clearly positive, finding lower cost solutions in the majority of the cases.

Table 3
Cost improvement of the AS-IR vs AS.

Inst.	C_{best}	C_{avg}	Inst.	C_{best}	C_{avg}
cgl_52	0.06%	0.24%	*cgl_29*	0.81%	1.53%
cgl_61	1.84%	3.28%	*cgl_82*	1.69%	2.52%
cgl_39	0.00%	0.00%	*cgl_59*	0.00%	0.14%
cgl_45	4.48%	3.84%	*cgl_71*	−1.10%	5.82%
cgl_79	9.96%	1.59%	*cgl_115*	0.00%	−0.67%
cgl_89	0.62%	−0.83%	*cgl_108*	−7.52%	−1.68%
cgl_46	2.25%	4.33%	*cgl_48*	5.38%	4.57%
cgl_77	−0.53%	1.71%	*cgl_51b*	2.79%	1.44%
cgl_18	0.00%	2.29%	*cgl_44*	0.00%	0.53%
cgl_74	4.04%	2.84%	*cgl_33*	−1.16%	6.43%
cgl_70b	2.67%	0.59%	*cgl_38*	0.00%	7.28%
cgl_67	0.51%	−0.69%	*cgl_48b*	0.99%	3.26%
cgl_51	7.50%	3.89%	*cgl_58*	0.00%	−0.25%

In the AS-IR, when the IR is called to try to improve the sequence of the best ant of the iteration, it only returns a new solution if the sequence has a lower cost than the input sequence. If not, the AS continue with the solutions obtained by the ants until the next try of the IR. Hence, the IR will never worsen the AS solutions. For those instances for which the AS-IR obtained a worst average cost (19.2% of the tested instances), the reason may be that the IR does not find lower cost solutions and, since the maximum allowed time was of 120 s, it consumed some time that reduced the number of constructions of the ants in the AS.

We noticed that the instances for which the AS-IR did not improved the average cost are some of the larger ones (*cgl_114, cgl_107, cgl_88, cgl_66*). We believe that this may be because the length of the IR windows is related to the size of the instances, and if the windows are very large, many nodes should be reinserted and it is more difficult to find better new windows due to the simple construction mechanism that just uses the random proportional rule (Ec.10), a simplicity that helped the feasibility search. A deeper study on these instances should be conducted and a smaller *max_window_len* should be set for large instances, but for this test we wanted to set the same value for all the instances. In any case, the worst C_{avg} improvement of the AS-IR was of −1.68% in *cgl_107* (i.e. the AS-IR obtained a C_{avg} 1.68% greater than the AS) , while the better improvement of the AS-IR was of 7.28% or 6.43% for instances *cgl_37* and *cgl_32*, respectively (see Table 3).

Table 3 shows the improvement in C_{best} and C_{avg} of the AS-IR compared to the original AS version. We calculated this improvement as $(C_{AS} - C_{AS\text{-}IR})/ C_{AS} * 100$. If this value is positive, it means that the AS-IR obtained a lower (better) cost than the AS. Considering all the tested instances -the ones for which the AS-IR obtained a better, lower and equal cost compared to the AS- the average improvement was of 1.36% (C_{best}) and 2.08% (C_{avg}). If we only look at the instances for which the AS-IR got better results, the average improvement was 3.04% in C_{best} and 2.91% in C_{avg}. These values may seem to be small improvements but, for this line, where the total production cost of each sequence is considerable, and a new sequence is produced daily, the potential savings can be very important.

After the results obtained from the cost performance analysis, and considering the great success of the AS-IR in the feasibility analysis, we can state that the addition on the IR to the AS is undoubtedly an important enhancement, making the AS-IR a more robust algorithm for this problem.

6. Conclusions

In this study we present the real-world problem of sequencing coils in a continuous galvanizing line of a steel making facility. This problem is similar to well-known combinatorial problems as the ATSP and the HCP, but in the daily activity of this line there are situations in which the set of coils to be produced make the sequencing really challenging, specially finding a feasible sequence. Due to the properties of the steel coils and the technical limitations of the line, some of them cannot be sequenced together, generating node-to-node constraints. Finding feasible sequences is crucial for the line; only once the feasibility is guaranteed, the production cost should be reduced.

The current algorithms scheduling this line, which have provided very good results during the last years, are lately facing difficulties in finding feasible sequences because of an important increase in the number of scheduling constraints, brought by the development of new steel grades and the growing necessity of reducing the stock levels. These algorithms are the ACS and the AS - two algorithms of the Ant Colony Optimization (ACO) family that have proved to be state-of-the art (Dorigo and Stützle, 2004) -, for which the problem is modelled as an ATSP. This seems to be the more straightforward approach, assigning a high cost to the forbidden transitions. However, sometimes the set of coils to be sequenced underly a cost distribution that makes finding feasible sequences very difficult for these algorithms, especially when the costs strongly influence the search and no special strategies for handling the constraints are used.

We briefly explain why the ACO constructive metaheuristic is a good choice for the scheduling of the finishing lines, the main characteristics of these two ACO algorithms, and the modifications required to apply them to this problem (i.e. initialization and update the pheromone for unfeasible sequences). In addition, we show how the ACO algorithms can fail in finding feasible sequences in some especially hard instances provided by the scheduling crew, analyzing and extracting two key issues that the instances share.

This increasing complexity of the sequencing instances and the limitations of the current algorithms led us to look for a new algorithm able to ensure feasibility. The new algorithm devised, the Ant System with Interval Reconstruction (AS-IR), embeds a novel own-developed *local search* named Interval Reconstruction (IR) into the AS, as explained in detail in Section 4.

As we demonstrate with the experiments conducted, using 30 real instances of the galvanizing line, the AS-IR successfully ensures feasibility (100% of success in all the instances), achieving its main goal. The key features of the IR are the focus on feasibility by a real handling of the constraints (targeting directly the forbidden links and avoiding cost-based biased decisions), the use of two random reconstruction windows (allowing to explore many different recombinations) and its efficiency. These features, together with its constructive nature, make the IR a very good local search algorithm to be hybridized with the AS, solving the issues detected when only using the latter.

Finally, with the main objective of improving the feasibility performance of the current algorithms satisfied, we also analyze the ability of the AS-IR to reduce costs. One advantage of the IR approach is that it can be used both to solve one constraint violation and to reduce the cost of a feasible sequence, by adding cost checks in a second phase. The experimental results show how the combination of the IR and the AS not only ensures feasibility but also helps in enhancing the cost performance.

Further research will be focused on improving the current solutions for other finishing lines where more complex constraints are encountered (i.e. multi-coil constraints). The presence of these type of constraints was one of the reasons for using constructive metaheuristics in their scheduling, and the IR may be a good improvement for these lines as well, since it also uses a constructive procedure in its reconstruction mechanism. Another future line of study will be the analysis of the underlying graph of the instances, which may give us further insights and help find new strategies to improve the feasibility performance of the current scheduling solutions

CRediT authorship contribution statement

Nicolás Álvarez-Gil: Conceptualization, Methodology, Software, Formal analysis, Investigation, Resources, Writing – original draft, Writing – review & editing, Data curation. **Segundo Álvarez García:** Conceptualization, Methodology, Software, Formal analysis, Investigation, Resources, Writing – review & editing, Data curation. **Rafael Rosillo:** Conceptualization, Methodology, Visualization, Investigation, Supervision, Writing – review & editing. **David de la Fuente:** Conceptualization, Investigation, Supervision.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This article was financially supported by the Spanish State Research Agency of the Spanish Ministry of Science and Innovation (MCIN/AEI/10.13039/501100011033), via the project 'SPeeding Up the transition towards Resilient circular economy networks: forecasting, inventory and production control, reverse logistics and supply chain dynamics' (SPUR, grant ref. PID2020-117021 GB-I00).

This article was financially supported by the ERASMUS+ Program via the project Academic System Resource Planning: A Fully-Automated Smart Campus (ASRP) (UE-19-ASRP-598757). The authors deeply appreciate the constructive comments and valuable feedback offered by the anonymous reviewers.

References

Akiyama, T., Nishizeki, T., & Saito, N. (1980). NP-Completeness of the Hamiltonian Cycle Problem for Biparite Graphs. *Journal of Information Processing, 3*(2), 73–76.

Applegate, D. L., et al. (2006). *The traveling salesman problem: A computational study.* Princeton University Press.

Applegate, D., & Cook, W. (1991). A computational study of the Job-shop scheduling problem. *Journal of Computing, 3*(2), 149–156.

Ascheuer, N., et al. (2000). A Branch & Cut Algorithm for the Asymmetric Traveling Salesman Problem with Precedence Constraints. *Computational Optimization and Applications, 17*.

Baniasadi, P., Ejov, V., Filar, J. A., Haythorpe, M., & Rossomakhine, S. (2013). Deterministic "Snakes and Ladders" Heuristic for the Hamiltonian cycle problem. *Math. Prog. Comp, 6*, 55–75.

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys, 35*(3), 268–308.

Cowling, P. I., Ouelhadj, D., & Petrovic, S. (2004). Dynamic scheduling of steel casting and milling using multi-agents. *Production Planning & Control, 15*(2), 178–188.

Dorigo, M. (1992). *Optimization, learning and natural algorithms (in italian)* (p. 140). Politecnico di Milano, Italy: DEI. Ph.D. thesis.
Dorigo, M., & Stützle, T. (2004). *Ant Colony Optimization*. Boston, MA: MIT Press.
Dorigo, M., Maniezzo, V., & Colorni, A. (1991). *The Ant System: An autocatalytic optimizing process. Technical report 91–016 revised, Dipartimento di Elettronica*. Milan: Politecnico di Milano.
Dorigo, M., & Gambardella, L. M. (1997). Ant Colony System: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation, 1*(1), 53–66.
Escudero, L. F. (1988). An inexact algorithm for the sequential ordering problem. *European Journal of Operational Research, 37*(2), 236–249. https://doi.org/10.1016/0377-2217(88)90333-5
Fernández, S., Álvarez, S., Díaz, D., Iglesias, M., & Ena, B. (2014). Scheduling a Galvanizing Line by Ant Colony Optimization. *ANTS 2014: Lecture Notes in Computer Science*, 8667. Springer, Cham.
Feo, T., & Resende, M. (1995). Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization, 6*(2), 109–133.
Framinan, J. M., Leisten, R., & Ruiz-García, R. (2014). *Manufacturing Scheduling Systems.*. https://doi.org/10.1007/978-1-4471-6272-8
Gambardella, L. M., & Dorigo, M. (2000). An Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem. *INFORMS Journal on Computing, 12*(3), 237–255. https://doi.org/10.1287/ijoc.12.3.237.12636
Graves, S. (1981). A review of Production Scheduling. *Operations Research, 29*(4), 646–675.
Garey, M. R., Johnson, D. S., & Tarjan, R. E. (1976). The planar Hamiltonian circuit problems is NP-Complete. *SIAM J. Computing, 5*(4), 704–714.
Garey, M. R., & Johnson, D. S. (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness. *Freeman*.
Ghiyasinasab, M., Lehouxa, N., Ménardb, S., & Cloutier, C. (2020). Production planning and project scheduling for engineer-to-order systems- case study for engineered wood production. *International Journal of Production Research*. https://doi.org/10.1080/00207543.2020.1717009
Gnonia, M. G., Iavagnilioa, R., Mossaa, G., Mummoloa, G., & Di Levab, A. (2003). Production planning of a multi-site manufacturing system by hybrid modelling: A case study from the automotive industry. *International Journal of Production Economics, 85*, 251–262.
Harjunkoski, I., & Grossmann, I. E. (2001). A Decomposition Approach for the Scheduling of a Steel Plant Solution. *Computers and Chemical Engineering, 25*(11–12), 1647–1660.
Helsgaun, K. (2000). An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research, 126*, 106–130. https://doi.org/10.1016/S0377-2217(99)00284-2
Helsgaun, K. (2017). An Extension of the Lin-Kernighan-Helsgaun TSP Solver for Constrained Traveling Salesman and Vehicle Routing Problems. *Technical report*.
Iglesias-Escudero, M., Villanueva-Balsera, J., Ortega-Fernandez, F., & Rodriguez-Montequín, V. (2019). Planning and Scheduling with Uncertainty in the Steel Sector: A review. *Applied Sciences, 9*, 2692. https://doi.org/10.3390/app9132692
LKH-3 (Keld Helsgaun): http://webhotel4.ruc.dk/~keld/research/LKH-3/. Accessed: 2019-01-16.
Kapanoglu, M., & Koc, I. O. (2006). A multi-population parallel genetic algorithm for highly constrained continuous galvanizing line scheduling. *Lecture notes in computer science, 4030*, 28–41.
Klement, N., Abdeljaouad, M. A., Porto, L., & Silva, C. (2021). Lot-Sizing and Scheduling for the Plastic Injection Molding Industry—A Hybrid Optimization Approach. *Applied Sciences, 11*, 1202. https://doi.org/10.3390/app11031202
Krishnamoorthy, M. S. (1975). An NP-hard problem in bipartite graphs. *SIGACT News, 7*(1), 26.
Korte, B., & Vygen, J. (2003). *Combinatorial Optimization. Algorithm and Combinatorics book series* (p. 21). Berlin, Heidelberg: Springer.
Lee, H. S., Murthy, S. S., Haider, S. W., & Morse, D. V. (1996). Primary production scheduling at steelmaking industries. *IBM Journal of Research and Development, 40*(2), 231–252.
Montemanni, R., et al. (2013). A decomposition-based exact approach for the Sequential Ordering Problem. *Journal of Applied. Operational Research, 5*.
Nekovář, F., Faigl, J., & Saska, M. (2021). Multi-Tour Set Traveling Salesman Problem in Planning Power Transmission Line Inspection. *IEEE Robotics and Automation Letters, 6*(4), 6196–6203.
Okano, H., Davenport, A. J., Trumbo, M., Reddy, C., Yoda, K., & Amano, M. (2004). Finishing line scheduling in the steel industry. *IBM Journal of Research and Development, 48*(5/6), 811–830.
Pina-Pardo, J. C., Silva, D. F., & Smith, A. E. (2021). The traveling salesman problem with release dates and drone resupply. *Computers & Operations Research, 129*. https://doi.org/10.1016/j.cor.2020.105170
Ruiz, R., & Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research, 177*(3), 2033–2049.
Stürtz, J. A., & Marchetti, P. A. (2020). Efficient Scheduling of a Real Case Study of the Pharmaceutical Industry Using a Mathematical-Algorithmic Decomposition Methodology. In *ICPR-Americas 2020: International Conference of Production Research – Americas Conference Proceedings* (pp. 171–176).
Tang, L., Liu, J., Rong, A., & Yang, Z. (2001). A review of planning and scheduling systems and methods for integrated steel production. *European Journal of Operational Research, 133*, 1–20.
Tang, L., Luh, P. B., Liu, J., & Fang, L. (2002). Steel-making process scheduling using Lagrangian relaxation. *International Journal of Production Research, 40*(1), 55–70.
Twaróg, S., Szwarc, K., Wronka-Pośpiech, M., Dobrowolska, M., & Urbanek, A. (2021). Multiple probabilistic traveling salesman problem in the coordination of drug transportation—In the context of sustainability goals and Industry 4.0. *Journal Plos one.*. https://doi.org/10.1371/journal.pone.0249077
Valls Verdejo, V., Pérez Alarcó, M., & Lino Sorlí, M. (2009). Scheduling in a continuous galvanizing line. *Computers & Operations Research, 36*, 280–296.
Verderame, P. M., & Floudas, C. A. (2010). Integration of Operational Planning and Medium-Term Scheduling for Large-Scale Industrial Batch Plants under Demand and Processing Time Uncertainty. *Industrial & Engineering Chemistry Research, 49*(10), 4948–4965.
Zhang, J., Guofo, D., Zou, Y., Shengfeng, Q., & Fu, J. (2017). Review of job shop scheduling and its new perspectives under Industry 4.0. *Journal of Intelligent Manufacturing, 30*, 1809–1830.

6. Conclusiones

Los problemas de optimización de la programación de la producción están presentes en todas las empresas del ámbito industrial, y tienen un impacto muy importante en los resultados económicos de estas. Por ello, dada la elevada complejidad de este tipo de problemas, la alta frecuencia con la que deben tomarse este tipo de decisiones operativas, su gran impacto en los costes de producción y en la satisfacción de los clientes, y la imperante necesidad de sumarse a la transformación digital de las empresas para mantenerse competitivas, es fundamental tener herramientas inteligentes capaces de aportar buenas soluciones a este tipo de problemas.

Entre las técnicas más adecuadas para las empresas industriales están las metaheurísticas, dada su capacidad de obtener soluciones de calidad para este tipo de problemas combinatorios de gran complejidad en un tiempo razonable para la naturaleza de estas decisiones operativas. Tanto los problemas de *scheduling* como las metaheurísticas han sido objeto de estudio por parte de la comunidad académica y científica durante años pero, sin embargo, su aplicación a los problemas reales de las empresas actuales todavía supone grandes retos. El objetivo principal de la presente tesis doctoral es acercar todo este conocimiento a un ámbito más práctico y cercano a las empresas, diseñando y aplicando técnicas metaheurísticas inteligentes capaces de afrontar dichos retos. Este objetivo fue alcanzado a lo largo de una investigación dividida en varios objetivos más específicos y presentando los resultados a través de diversas publicaciones científicas.

En las dos primeras publicaciones (Capítulos 2 y 3), directamente relacionadas, se introducen todas las características del *Firefly Algorithm* (FA) y se propone una implementación de este algoritmo para la resolución de un conocido problema de *scheduling* adaptado a las necesidades actuales de las empresas. Además, se presentan varias mejoras del algoritmo, basadas en una buena inicialización de la población y la introducción de varias estrategias de búsqueda local, obteniendo una versión mejorada del FA. A través de varios experimentos, comparando el método de optimización propuesto con otras metaheurísticas conocidas y aplicándolo a varias instancias del problema adaptado, se demuestra como el método propuesto es una buena opción para la optimación de problemas de programación de tareas en los que la flexibilidad en la producción, la satisfacción de los clientes y la reducción de los costes de producción deben satisfacerse simultáneamente.

En la segunda parte de la investigación, desarrollada a través de las dos publicaciones expuestas en los Capítulos 3 y 4, se continua con la búsqueda de métodos de solución capaces de resolver problemas de programación de la producción centrándose en una situación real de una empresa industrial de producción de acero. El problema analizado es la programación de una línea de galvanizado, en el que debe se encontrarse el orden óptimo en el que procesar varias bobinas de acero, reduciendo los costes de producción y evitando posibles paradas de la instalación. Este problema suponía un gran reto para la empresa, dado que la necesidad de reducir los niveles de inventario y la aparición de nuevos grados de acero supusieron una mayor dificultad para los algoritmos en uso para obtener secuencias de producción adecuadas. Primero, se analizaron en profundidad estos algoritmos en uso, algoritmos de la familia *Ant Colony Optimization* (ACO), y las características del problema para detectar cuales eran las razones que hacían que no se obtuvieran soluciones adecuadas en ciertas situaciones. Pese a que estos algoritmos ACO son una muy buena elección como métodos de resolución dada la naturaleza del problema, se detectó que la forma en la que realizan la exploración del espacio de soluciones, mediante un proceso de construcción altamente influenciado por los costes de las decisiones locales, hacía que no fueran capaces de obtener soluciones factibles de bajo coste para las distribuciones de costes generadas por ciertos conjuntos de bobinas a ser programadas.

Una vez detectado el problema, se desarrolló una novedosa estrategia de búsqueda capaz de suplir las deficiencias detectadas en los algoritmos ACO. Este nuevo algoritmo, en *Interval Reconstruction*, permite mejorar las soluciones generadas por los algoritmos ACO mediante una reparación muy eficiente y separada en dos niveles. Mediante varios experimentos con diversas instancias reales de la línea de galvanizado se demuestra como combinando el *Interval Reconstruction* con un algoritmo metaheurístico *Ant System*, se obtiene un método de optimización mejorado capaz de asegurar soluciones factibles de bajo coste, objetivo principal del problema. Aunque originalmente el *Interval Reconstruction* fue diseñado para el problema se secuenciación de la línea de galvanizado, puede ser utilizado para otros muchos problemas de optimización combinatoria, tanto como algoritmo único como en combinación con otros métodos de optimización.

Los métodos de optimización como las metaheurísticas siguen teniendo a día de hoy un gran potencial para mejorar la producción de las empresas, y su uso es cada vez más necesario para mantenerse competitivo en el mundo industrial actual en el que la digitalización y el empleo de este tipo de herramientas marca la diferencia. En esta tesis se muestra cómo una correcta implementación de estas técnicas metaheurísticas, en concreto el *Firefly Algorithm* y una versión mejorada del *Ant System*, puede impactar notablemente en la obtención de mejores resultados en el proceso de programación de tareas de producción, para un problema teórica adaptado y para un problema encontrado en una empresa real.

7. Extensiones y líneas de trabajo futuro

Como continuación a la investigación realizada para la presente tesis doctoral se estaría interesado en dos líneas de investigación complementarias: analizar el uso del *Interval Reconstruction* en otros problemas y el uso de técnicas de inteligencia artificial para la autoconfiguración de los parámetros de los algoritmos estudiados.

El algoritmo *Interval Reconstruction*, presentado por primera vez en una de las publicaciones de esta tesis, es una novedosa estrategia de destrucción parcial y posterior reconstrucción de gran eficiencia que puede ser utilizada como único método de resolución o en conjunto con otras metaheurísticas para mejorar la calidad de las soluciones. Por ello, es interesante estudiar su aplicación a otros problemas de programación de tareas o de optimización combinatoria dado que, aunque fue diseñado para el problema de secuenciación de bobinas en una línea de galvanizado, su esquema general de funcionamiento es lo suficientemente flexible para ser aplicado a otros problemas. Además, este algoritmo puede usarse para refinar soluciones propuestas por otras metaheurísticas, mejorando su actuación como se demostró en esta investigación combinándolo con un algoritmo *Ant System*.

Por otro lado, todos los métodos aproximados de optimización estudiados constan de varios parámetros que deben ser configurados y que tienen un impacto directo en su actuación (Xu et al, 2008; Lindauer et al, 2015). Una buena configuración de los parámetros de un algoritmo para un problema determinado puede no ser la más adecuada para otros problemas o incluso para diferentes instancias del mismo problema. Es de gran interés estudiar métodos de autoconfiguración capaces de, analizando las características de cada problema o de cada instancia particular, obtener automáticamente la mejor configuración mediante sistemas de aprendizaje basados en inteligencia artificial.

8. Bibliografía

Applegate, D., Cook, W. (1991) A Computational Study of the Job-Shop Scheduling Problem. ORSA Journal on Computing 8 (2). https://doi.org/10.1287/opre.8.2.219

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Computing Surveys, 35(3), 268–308.

Brettel, M., Friederichsen, N., Keller, M., Rosenberg, M. (2014) How virtualization, decentralization and network building change the manufacturing landscape: an Industry 4.0 perspective. Int J Mech Aerosp Ind Mech Manuf Eng 8:37–44

Çaliş, B., Bulkan, S. (2015) A research survey: review of AI solution strategies of job shop scheduling problem. J Intell Manuf 26, 961–973. https://doi.org/10.1007/s10845-013-0837-8

Dorigo, M., Stützle, T. (2002) The ant colony optimization metaheuristic: Algorithms, applications and advances. In Handbook of Metaheuristics. Kluwer Academic Publishers, pp. 251–285.

Dorigo, M., & Stützle, T. (2004). Ant Colony Optimization. Boston, MA: MIT Press.

El-Ghazali, T. (2009) Metaheuristics: From Design to Implementation. Wiley (ISBN: 978-0-470-27858-1)

ElFar, O. A., Chang, C.K. , Leong, H. Y., Peter, A. P. , Chew, K. W., Show, P. L. (2020) Prospects of industry 5.0 in algae: Customization of production and new advance technology for clean bioenergy generation, Energy Conversion and Management.

Farmer, J.D., Packard, N., Perelson, A. (1986) The immune system, adaptation and machine learning. Physica D (2), 187–204.

Feo, T. A., Resende, M. G. C. (1995) Greedy randomized adaptive search procedures. Journal of Global Optimization 6, 109–133.

Fogel, L.J. (1962) Toward inductive inference automata. In Proceedings of the International Federation for Information Processing Congress, Munich, 395–399.

Framinan J.M., Leisten R., Ruiz García R. (2014a) Overview of Manufacturing Scheduling. In: Manufacturing Scheduling Systems. Springer, London. https://doi.org/10.1007/978-1-4471-6272-8_1

Framinan J.M., Leisten R., Ruiz García R. (2014b) Overview of Scheduling Models. In: Manufacturing Scheduling Systems. Springer, London. https://doi.org/10.1007/978-1-4471-6272-8_3

Framinan J.M., Leisten R., Ruiz García R. (2014c) Overview of Scheduling Methods. In: Manufacturing Scheduling Systems. Springer, London. https://doi.org/10.1007/978-1-4471-6272-8_7

Gao K., Cao Z., Zhang L., Chen Z., Han Y., Pan Q. (2019) A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems. IEEE/CAA Journal of Automatica Sinica 6(4), 904-916. DOI: 10.1109/JAS.2019.1911540.

Gen, M., Lin, L. (2013) Multiobjective evolutionary algorithm for manufacturing scheduling problems: state-of-the-art survey. Journal of Intelligent Manufacturing 25, 849–866

Glover, F. (1986) Future paths for integer programming and links to artificial intelligence. Computers and Operations Research (13), 533–549.

Glover, F. (1989) Tabu search: Part I. ORSA Journal on Computing 1(3), 190–206.

Graham R.L., Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G. (1979) Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. Annals of Discrete Mathematics (5), 287-326. https://doi.org/10.1016/S0167-5060(08)70356-X

Groetschel, M., Lovasz, L. (1993). Geometric algorithms and combinatorial optimization. Springer, Berlin [a.o.]. 2th, corrected edition.

Güçdemir, H., Selim, H. (2017) Customer centric production planning and control in jobs shops: a simulation optimization approach. J Manuf Syst 43:100–116

Holland, J.H. (1962) Outline for a logical theory of adaptive systems. Journal of the ACM 3, 297–314.

Kennedy, J., Eberhart, R.C. (1995) Particle swarm optimization. In IEEE International Conference on Neural Networks, Perth, Australia, 1942–1948.

Khan, A., Chaudhry, I. (2015) A research survey: review of flexible job shop scheduling technique. International Transactions in Operational Research 23 (3). https://doi.org/10.1111/itor.12199

Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P. (1983) Optimization by simulated annealing. Science 220(4598), 671–680.

Koza, J.R. (1992) Genetic Programming. MIT Press, Cambridge, MA.

Lindauer, M.T, Hoos, H.H, Hutter, F., Schaub T. (2015) AutoFolio: Algorithm configuration for algorithm selection. In B. Bonet and S. Koenig, editors, AAAI. AAAI Press.

Lohrer, M. (2013) A comparison between the firefly algorithm and particle swarm optimization. PhD thesis

Maddikuntaa, P., Phamb, Q., Prabadevi Ba, P., Deepaa, N., Kapal Devc, K., Gadekallua, T.R., Rubyd, R., Madhusanka Liyanagee, M (2021) Industry 5.0: A Survey on Enabling Technologies and Potential Applications. Journal of Industrial Information Integration, DOI: 10.1016/j.jii.2021.100257

Martí, R., Laguna M., Glover, F. (2006). Principles of scatter search, European Journal of Operational Research 169 (2), 359-372. https://doi.org/10.1016/j.ejor.2004.08.004.

Martin, O., Otto, S.W., Felten, E.W. (1991) Large-step Markov chains for the traveling salesman problem. Complex Systems 5(3), 299–326.

Mladenovic, M (1995) A variable neighborhood algorithm—a new metaheuristic for combinatorial optimization. In Abstracts of Papers Presented at Optimization Days.

Morinaga,Y., Nagao, M., Sano, M. (2014) Optimization of flexible job-shop scheduling with weighted tardiness and setup-worker load balance in make-to-order manufacturing. In: Joint 7th international conference on soft computing and intelligent systems (SCIS) and 15th international symposium on advanced intelligent systems (ISIS). https://doi.org/10.1109/scis-isis.2014.7044681

Parunak, H.V.D (1991) Characterizing the manufacturing scheduling problem. Journal of Manufacturing Systems 10(3), 241-259. https://doi.org/10.1016/0278-6125(91)90037-3.

Rechenberg, I. (1965) Cybernetic solution path of an experimental problem. Technical Report, Royal Aircraft Establishment Library Translation No. 1112, Farnborough, UK.

Sörensen, K., Glover, F. (2013) Metaheuristics. Encyclopedia of Operations Research and Management Science, 960-970. DOI:10.1007/978-1-4419-1153-7_1167

Voudouris, C. (1998). Guided local search: An illustrative example in function optimization. BT Technology Journal 16(3), 46–50.

Wang, S., Wan, J., Li, D., Zhang, C. (2016a) Implementing smart factory of Industrie 4.0: an outlook. Int J Distrib Sens Netw 12(1):3159805

Wang S, Zhang C, Li D (2016b) A Big Data centric integrated framework and typical system configurations for smart factory. In: Wan J, Humar I, Zhang D (eds) Industrial IoT technologies and applications. Industrial IoT 2016. Lecture notes of the institute for computer sciences, social informatics and telecommunications engineering, vol 173. Springer, Cham, pp 12–23

Xu, L., Hutter, F., Hoos, H.H, Leyton-Brown, K (2008) SATzilla: portfolio-based algorithm selection for SAT. Journal of Artificial Intelligence Research, 32:565–606.

Yonezawa, Y., Kikuchi, T. (1996) Ecological algorithm for optimal ordering used by collective honey bee behavior. In 7th International Symposium on Micro Machine and Human Science, 249–256.

Yang, X.S. (2008) Nature-inspired metaheuristic algorithm. Luniver Press, Bristol.

Yazdani, M., Amiri, M., Li, P., Zandieh, M. (2009) Flexible job-shop scheduling with parallel variable neighborhood search algorithm. Exp Syst Appl 37:678–687