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

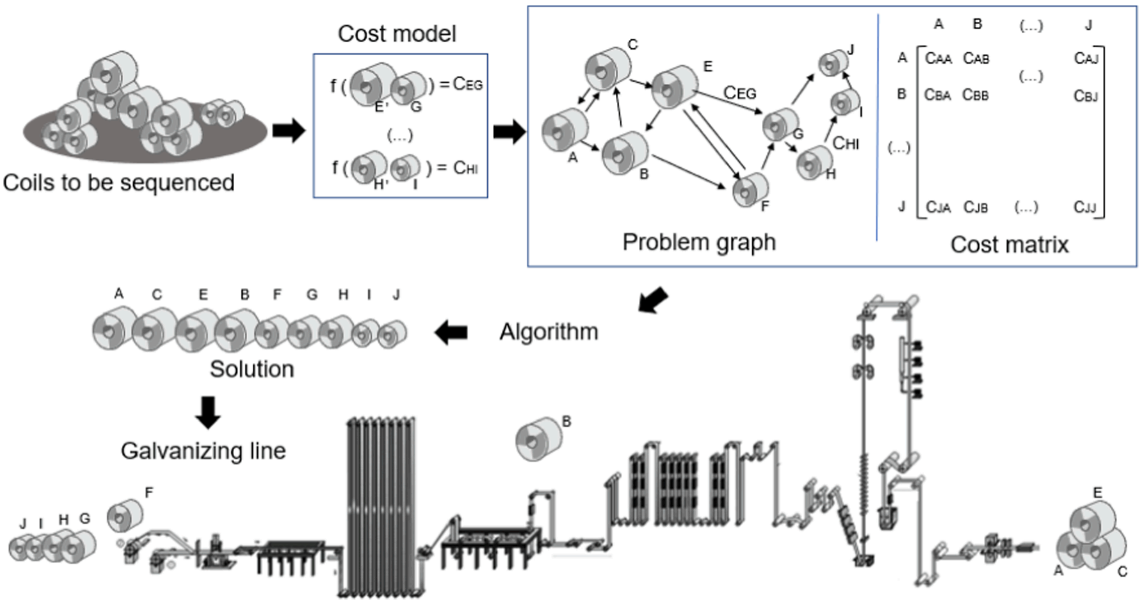


Fig. 1. Description of the problem and the context of the real case.

(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:

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

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

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

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, lets 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, lets 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.

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

$$\Delta\tau_{ij}^k = \begin{cases} \frac{1}{C^k}, ifarc(i,j)belongstoS^k \\ 0, otherwise \end{cases} \tag{8}$$

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, lets 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

$$
\begin{array}{c|cccccccc}
 & \text{Node 0} & \text{Node 1} & \text{Node 2} & \text{Node 3} & \text{Node 4} & \text{Node 5} & \text{Node 6} & \text{Node 7} \\
\hline
\text{Node 0} & 100 & 100 & 100 & 100 & 0 & 0 & 0 & 0 \\
\text{Node 1} & 100 & 100 & 100 & 100 & 0 & 0 & 0 & 0 \\
\text{Node 2} & 100 & 100 & 100 & 100 & 0 & 0 & 0 & 0 \\
\text{Node 3} & 100 & 100 & 100 & 100 & 0 & 0 & 0 & 0 \\
\text{Node 4} & -1 & -1 & -1 & -1 & 100 & 100 & 100 & 100 \\
\text{Node 5} & -1 & -1 & -1 & -1 & 100 & 100 & 100 & 100 \\
\text{Node 6} & -1 & -1 & -1 & -1 & 100 & 100 & 100 & 100 \\
\text{Node 7} & -1 & -1 & -1 & -1 & 100 & 100 & 100 & 100
\end{array}
$$

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

$$
\begin{array}{c|cccccccc}
 & \text{Node 1} & \text{Node 2} & \text{Node 3} & \text{Node 4} & \text{Node 5} & \text{Node 6} & \text{Node 7} & \text{Node 8} \\
\hline
\text{Node 1} & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{Node 2} & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{Node 3} & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\
\text{Node 4} & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 \\
\text{Node 5} & -1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 \\
\text{Node 6} & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 \\
\text{Node 7} & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 \\
\text{Node 8} & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1
\end{array}
$$

Sequence:

$$\begin{tikzcd}
\text{Node 1} \arrow[r] & \text{Node 3} \arrow[r] & \text{Node 5} \arrow[r] & \text{Node 7} \arrow[r, phantom, "\times"] & \text{Node 2} \arrow[r] & \text{Node 4} \arrow[r] & \text{Node 6} \arrow[r] & \text{Node 8}
\end{tikzcd}$$

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, $W_1$, $W_2$ ← SelectIntervals($S^{best}$, target_arc)
2  for reconstruction_try in range max_reconstruction_tries:
3      $B = W_1 \cup W_2$
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

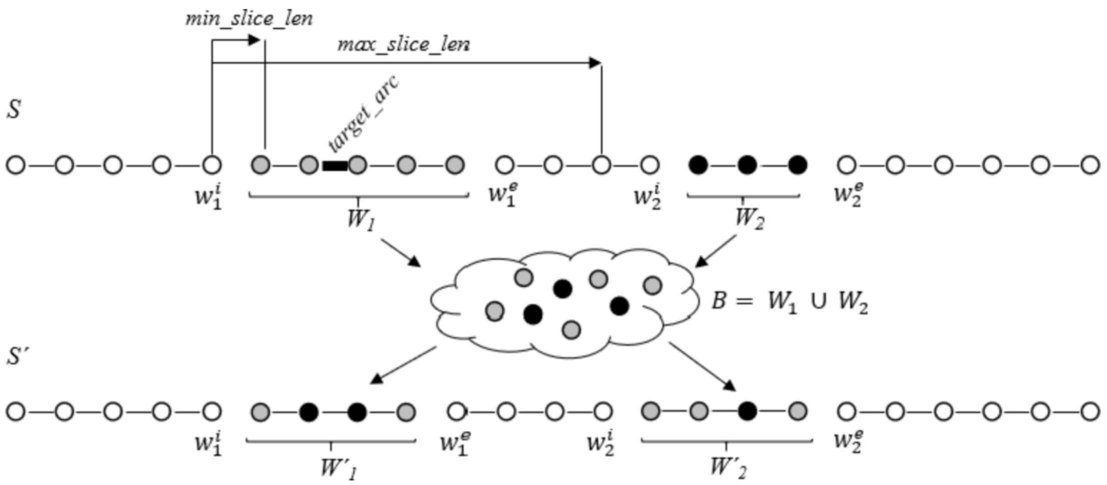


Fig. 4. One reconstruction try of the Interval Reconstruction.

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.

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

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).

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

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.

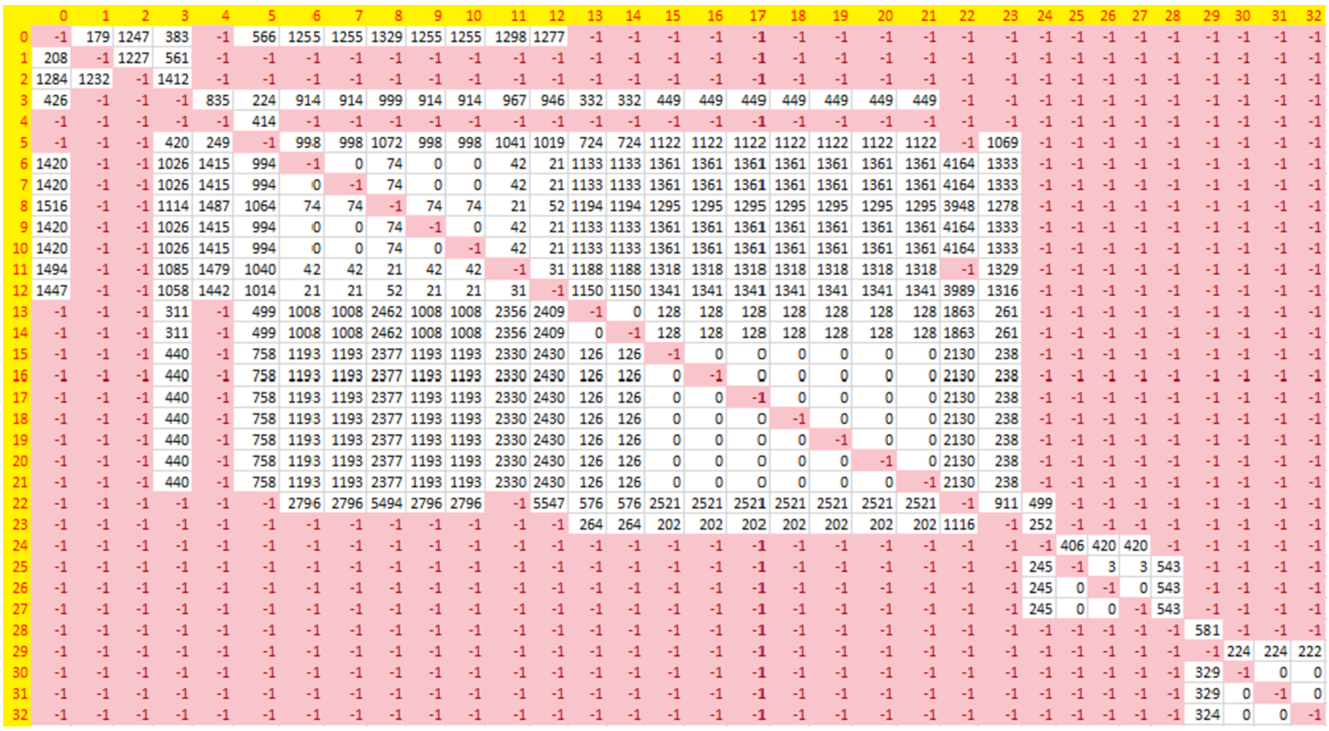
Another reason for the AS having a better performance than the ACS can be that the AS is more open to exploration due to its construction mechanism (the next node depends purely on the probability distribution given by classical *random proportional* rule, Ec.3) and the way the pheromone is updated (all 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.
- Lets̕ 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.

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%

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.

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.