Computers & Industrial Engineering 192 (2024) 110206

Contents lists available at ScienceDirect

Computers & Industrial Engineering

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

An effective graph-analysis method to schedule a continuous galvanizing line with campaigning boundary constraints

Segundo Álvarez-García [a,b,*], Nicolás Álvarez-Gil [a,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
Graph theory
Hybrid algorithms
Ant colony optimization

ABSTRACT

This work deals with the scheduling optimization of a Continuous Galvanizing Line (CGL) in the steel industry. We introduce a real-case extension of the CGL scheduling problem that is concerned with the linking of campaigns. In order to link the different campaigns planned at the CGL, the scheduler may fix the start coil of the sequence, the end coil, or both, introducing new *boundary constraints* (BC) to the sequencing problem. This shows to have a significant impact in the performance of the current Ant Colony Optimization (ACO) algorithms in use, failing to find feasible solutions with reliability. Our research aims at improving this reliability. Viewing sequences as paths in a directed weighed graph, in which coils are the nodes, and coil changes the edges, the BC problem is to find a minimum cost Hamiltonian path that respects the required start and end nodes. In this paper we study the negative impact brought by the BC in 30 challenging instances from the CGL, we discuss the reasons behind, and we propose a new algorithm able to robustly assure feasibility in all of them. We introduce a brand-new graph analysis method devised as an effective surrogate check for feasibility, which runs embedded in the ACO sequence construction heuristic. In the experimental analysis we see that it clearly boosts performance, increasing reliability by 20%, up to 99.67%. By teaming up with the Interval Reconstruction local search, the reliability and quality of the solutions shows to improve further. This graph analysis method we propose, though illustrated within ACO as a use case, is applicable to any constructive metaheuristic.

1. Introduction

Planning and scheduling operations is fundamental in the manufacturing industries, bringing efficiency to production, logistics or maintenance tasks, among others. Scheduling helps to coordinate and plan the use of resources, such as employees, equipment, and raw materials. Scheduling is critical for improving productivity, quality and service indicators, because it helps minimize downtimes, delays and unexpected disruptions in the production process. By improving planning and scheduling, inventories can also be reduced without impacting customer satisfaction (Maravelias & Sung, 2009).

Most scheduling problems are combinatorial optimization problems, many of which are NP-hard problems (Karp, 1972), that is, due to their formulation it does not exist any polynomial-time algorithm able to resolve them, assuming that P $\neq$ NP (Garey & Johnson, 1979). Approximate methods are used to solve this kind of problems, which means that finding the *global* optimum can never been assured. When developing optimization algorithms for the industry, computation time is very relevant: the final user needs to obtain a good quality solution in a reasonable time, so the efficiency of the algorithm is very important. A trade-off between solution quality and computation time is always in play.

The sequencing of jobs in one machine can be related to the famous Traveling Salesman Problem (TSP) (Applegate, Bixby, Chvatal, & Cook, 2006), and thus may be solved effectively by algorithms designed for this problem. This work deals with the scheduling of a Continuous Galvanizing Line (CGL) in a steel making factory, which has successfully been solved by different metaheuristics, among them ACO (Fernández-Alzueta et al., 2014). Álvarez-Gil et al. (2022a) studied the CGL scheduling problem in 30 real-world challenging instances, focusing on assuring feasibility –i.e., all constraints are respected– in very constrained scenarios, proposing a new ACO variant with a novel local search

* Corresponding author at: Business Management Department, University of Oviedo, Edificio Departamental Este, 1ª Planta, Campus de Gijón (33204), Gijón, Asturias, Spain.
E-mail addresses: segundo.alvarez-garcia@arcelormittal.com (S. Álvarez-García), alvareznicolas@uniovi.es (N. Álvarez-Gil), rosillo@uniovi.es (R. Rosillo), david@uniovi.es (D. de la Fuente).

https://doi.org/10.1016/j.cie.2024.110206
Received 6 September 2023; Received in revised form 25 March 2024; Accepted 6 May 2024
Available online 9 May 2024

(Interval Reconstruction, AS-IR) able to perform successfully where the algorithms so far in use failed.

The motivation of this work stems from the feasibility issues encountered when applying the ACO algorithms in the context of linking the planned campaigns, where additional constraints make even more difficult the optimization problem. Behind this important real-case problem, a higher motivation lies in the aspiration to devise an efficient mechanism able to steer constructive heuristics in complex graphs, endowing them with an effective look-ahead capability.

The contributions of this work are (1) the introduction of a new scheduling problem concerning the sequencing of steel coils in a CGL with boundary constraints (BC) related to the link between schedules; (2) the analysis of the performance of the ACO algorithms in use, which are not able to assure feasibility when these BC are introduced; and (3) the development of graph analysis (GA) techniques –designed to perform as a surrogate method for checking feasibility– and their streamlined hybridization with ACO to robustly overcome the feasibility issues in the benchmark instances. The GA method developed is not ACO-dependent, it can be applied to any constructive metaheuristic.

The rest of the paper is structured as follows: Section 2 presents a literature review of existing methods and problems related to our problem and line of research. Section 3 provides the description of the scheduling problem. In Section 4 we describe the GA method and the new algorithm developed to solve the problem. Section 5 shows and analyses the computational results achieved. Section 6 gathers the conclusions of the work and provides hints on future lines of research.

2. Literature review

Following the classification of production scheduling problems found in Graves (1981), our problem's paradigm is the flow shop problem (FSP), with one-stage one-processor, also called the one-machine problem. Regarding the scheduling criteria, we have all costs accounted in a single fitness function (single objective), without additional schedule performance objectives to optimize (level of production resources, percentage of late tasks, etc.). The problem consists in sequencing in a single facility the set of input items (coils) with minimum total cost.

In the book Principles of Sequencing and Scheduling (Baker & Trietsch, 2009), the authors provide a great introduction to scheduling optimization through operations research. The book goes over a set of classical scheduling problems, such as single-machine problem, earliness and tardiness costs, etc., and explains traditional techniques that can resolve them, including well-known metaheuristics like Simulated Annealing, Branch and bound, or Genetic Algorithms, to cite a few.

The scheduling optimization in the steel industry has received attention in the literature, though the number of publications is not abundant. Works in a steel factory can be found in Harjunkoski & Grossmann (2001). An exhaustive review for scheduling at Primary operations with uncertainty can be found in Iglesias-Escudero et al. (2019). The scheduling problem addressed in this work, though, belongs to Finishing Operations, for which those studies do not apply due to the very different nature of the scheduling at the finishing lines.

Regarding the CGL problem, Okano et al. (2004) describe a scheduling problem for the CGL plus other upstream facilities involved, aiming at a standard solution regarding selection and sequencing stages. They apply clustering methods that select homogeneous coils from the stock, in order to increase connectivity and therefore feasibility. They build this way the campaigns from downstream to upstream, attending at due dates, and then sequence from upstream to downstream, eventually repairing flow and due dates issues.

Kapanoglu & Koc (2006) address the problem of scheduling a CGL in highly constrained scenarios, but taking a wider scope. Instead of a fixed set of coils to be sequenced, they look for *primary coils* –coils from the targeted campaign– to build the sequence, picking them by due dates. In a second stage, unfeasible solutions are repaired by taking from the inventory *secondary coils* –coils from other campaigns– to fix violations. A very interesting Multi-population Parallel Genetic Algorithm is developed for the solution.

Our study has a more concise scope than this paper, focusing exclusively on avoiding constraints violations and not on fixing them. In the factory for which we are optimizing the schedules, fixing violations is a manual task: selection decisions can only be taken by experts who must ponder many factors for their choice –due dates, stock building strategies, quality, etc. As a remark, it is worth noting that sometimes the best decision for repairing an unfeasible sequence is to remove coils, not to add extra coils –if that is consistent with due dates.

Valls Verdejo et al. (2009) address scheduling of a CGL with the use of a Tabu Thresholding (TT) combined with a Tabu Search (TS) algorithm (Glover, 1989), that outperforms the manual schedules solutions. The problem addressed includes multi-coil conditions related to the coils' weights, which divides them in two sets heads and tails. The TT focuses on finding feasible sequences, and includes two local search methods for reducing and diversifying heads and improving fitness. The TS acts as a global optimizer.

Tang & Gao (2009) study a CGL scheduling problem in which the coils are classified into inner and outer coils (depending on the upper side chosen when run at the line), and must be sequenced separately, requiring to insert several inner coils between the outer coils. The problem objective is to minimize the total changeover cost, the number of inner coils inserted in the outer coils part, and the number of transitions coils needed to fix infeasibilities. The authors propose a TS algorithm for which 5 neighborhoods or kind of moves are defined: 2-opt, path insert, path swap, coil exchange and coil swap. The first 3 moves are only applied to the inner part of the schedule. The algorithm also uses intensification and diversification search, where diversification consists in performing shifts to the best solution so far obtained and reinitializing the TS.

Fernandez et al. (2014) address the problem of sequencing a CGL with the same approach we take as a (low) constrained ATSP, successfully adapting the AS algorithm, initially devised for the TSP, to handle scheduling constraints and optimize the schedules. Important savings are reported compared to the reference schedules.

Álvarez-Gil et al. (2022b) study highly constrained scenarios brought by the production of a more heterogeneous inventory mix, which challenges the AS performance. Analyzing a set of 30 especially challenging instances of the problem that prove to pose serious difficulties for the AS, they develop a novel local search especially designed for highly constrained scenarios, due to its *constructive* character. Their *Interval Reconstruction* local search procedure is described further below.

We have found scarce literature on sequencing metaheuristics that makes use of graphs properties or graph theory to improve their performance, which is our line of research.

Alba & Chicano (2007) propose a new algorithm ACO *huge graphs* (ACOhg) designed to solve problems with huge underlying graphs –some over 10^6 nodes. This brings important issues related to memory consumption and efficiency. ACOhg introduces very interesting ideas like building and evaluating partial solutions or reducing the memory consumption of the pheromone trails. ACOhg deals with a problem far from the one we are concerned with –we tackle complex but relatively small graphs.

Wang et al. (2014) present a graph based ACO for an integrated process planning and scheduling problem, in which they make use of an AND / OR graph structure to store alternative paths in the production route of the items. In this problem, not all nodes of the AND / OR graph structure need to be visited, i.e., we are not looking for a Hamiltonian path. The scope of this work is quite different from our study. The paper focuses especially in avoiding local optima, for which it makes relevant changes in the pheromone deposition method. The use of the graph here concerns the representation of the problem, but no properties or analysis of the graph are considered into the algorithm, as we intend to do.

As already mentioned, feasibility deals with finding Hamiltonian

paths in graphs. So, any theoretical approach for this problem in graph theory is of application for our problem.

Bang-Jensen & Gutin (2009) present a comprehensive study of graph theory putting their focus exclusively in undirected graphs –or digraphs–, which is our topic of interest for scheduling. The BC imposed by scheduling campaigning is already conceptualized in graph theory, as we can read in this paper: an *(x, y)*-Hamiltonian path is a Hamiltonian path from x to y. If such a path exists, we say that a digraph is Hamiltonian-connected. Asking for such a path in an arbitrary digraph is an even stronger requirement than asking for a Hamiltonian path: the theory confirms that the *(x, y)*-Hamiltonian path problem is NP-complete.

Sleegers & van den Berg (2022b) discuss the best backtracking algorithms for the Hamiltonian Cycle Problem (HCP) and argue that even if the problem is NP-hard almost all of the instances are not. All the algorithms in the study are based in depth-first search, some of them enhanced with edge-pruning techniques for efficiency. Especially interesting edge-pruning methods are four non-Hamiltonian checks: degree check, that verifies any edge with 1- or 0-degree, premature closure check, disconnectedness check, and one-connectedness check, which looks for articulation points –any vertex that if removed, breaks up the graph in two or more disconnected parts. An example of a difficult instance for the HCP can be found in Fig. 1.

Sleegers & van den Berg (2022a) use two evolutionary algorithms to generate hard instances of the HCP, evaluating hardness as the number of recursions required by Vandegriend–Culberson, the best-known exact backtracking algorithm for the problem (Vandegriend and Culberson, 1998).

These backtracking algorithms are very interesting and of inspiration for our solution. Nevertheless, we must remember that they address the HCP, which presents 4 differences with respect to our CGL scheduling problem: (1) we are looking for paths, not cycles; (2) we are imposing start and end vertices; (3) we are tackling directed graphs, not undirected; and (4) we are looking for minimum cost paths, not just paths. We must also recall that, though highly streamlined by pruning, backtracking algorithms are computationally expensive.

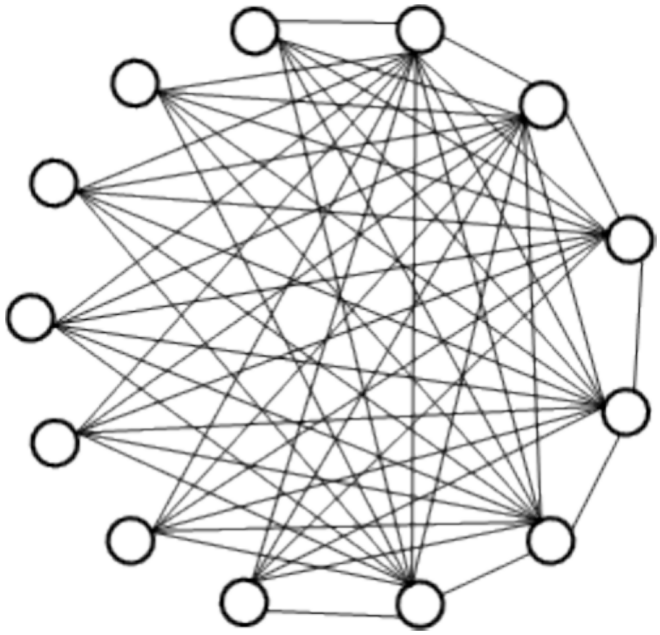

Fig. 1. Hardest Hamiltonian graph for size 13, non-Hamiltonian.

3. Problem description

3.1. Industrial context

In a steel factory, the planning and scheduling of all its units plays a critical role. The overall steel making process is complex, with many possible different routes and designs (Totten, Funatani, & Xie, 2004). Two big parts can be differentiated: Primary operations, where the pig iron is made from iron ore and coke at the blast furnace, refined into liquid steel by reducing its carbon content at the steel shop, and then solidified in the casters facilities into big blocks of steel (slabs, billets); and Finishing operations, where those slabs and billets are gradually subject to transformations in size (width, thickness) and mechanical properties (yield strength) along successive processes, and so transformed into a steel strip that is cut in sheets or wound in coils. The surface of the strip is usually protected against corrosion by electrochemical treatments such as galvanizing or tinning. In the Continuous Galvanizing Lines (CGL), coils are unrolled and welded into a never-ending steel strip that goes into a furnace for annealing treatments, a zinc bath for galvanizing the surface with a zinc coating layer, and a skin-pass machine for mechanical properties refinement and rugosity control. The strip is eventually oiled for further protection and recoiled to be sent to the customer Fig. 2.

The planning and scheduling of all these processes at the steel factory is hierarchical (Maravelias & Sung, 2009), from the high-level planning of volumes and flows of campaigns along the units to the most operational low-level scheduling of the items to be processed in each unit for the day. Items (i.e., slabs, coils) with similar properties are grouped in campaigns that are sequenced one after another attending to tactical and operational criteria, being service (customer dates) an important one. These decisions are taken by planning experts with the help of support tools. Finally, the schedulers at each facility need to decide the arrangement of the items of each campaign, usually for a horizon of 1 to 2 days. The arrangement must respect the production rules and minimize as much as possible losses related to quality, yield, productivity, etc. Finding the best arrangement is a complex combinatorial problem that needs the help of optimization techniques in order to build good quality schedules, and is the problem we are discussing in this paper. See the underlying graph of a difficult real-world instance for the CGL scheduling problem in Fig. 3.

In some manufacturing industries, the scheduling optimization problems are modeled as a make-span minimization problem (Hejazi & Saghafian, 2005).

In the CGL, however, the production time for each coil is constant no matter its position in the schedule, and no setup stops are needed between coils, that may increase the total make-span –setups appear only between campaigns. Quality costs appear instead at each coil change or *coil transition* (Fernandez et al., 2014). The fitness function to be minimized is the sum of all the transition costs. Aside to the costs, a wide set of production rules introduce scheduling constraints that forbid certain coils to be sequenced consecutively, because this brings issues or is not possible for certain machines at the CGL (for instance two coils that do not weld together, or two coils whose difference in strip width is over a given limit, or which have very different annealing temperatures targets in the furnace). If an infeasible sequence was to be processed in the CGL, there would be a risk of strip breakage, which represents a huge cost due to the unproductive long hours required to fix the issue. The strip has to be manually welded where it broke, and if this happens into the annealing furnace, the operation is very complicated, sometimes requiring more than 24 h to accomplish it.

3.2. The CGL problem

The real-world problem under study consists in arranging all coils for the CGL at an important steel company. We aim at minimizing the total production cost while respecting the production constraints. To these

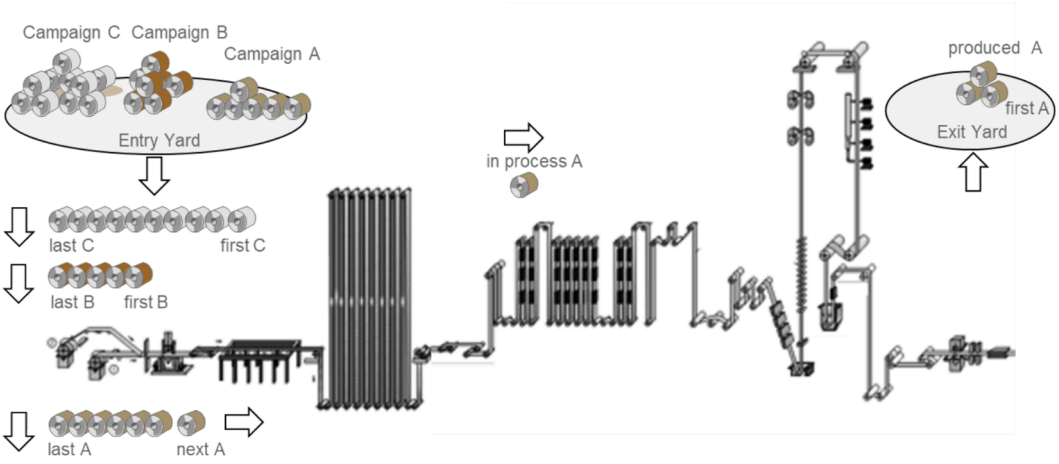

Fig. 2. Scheduling campaigns at a Continuous Galvanizing Line.

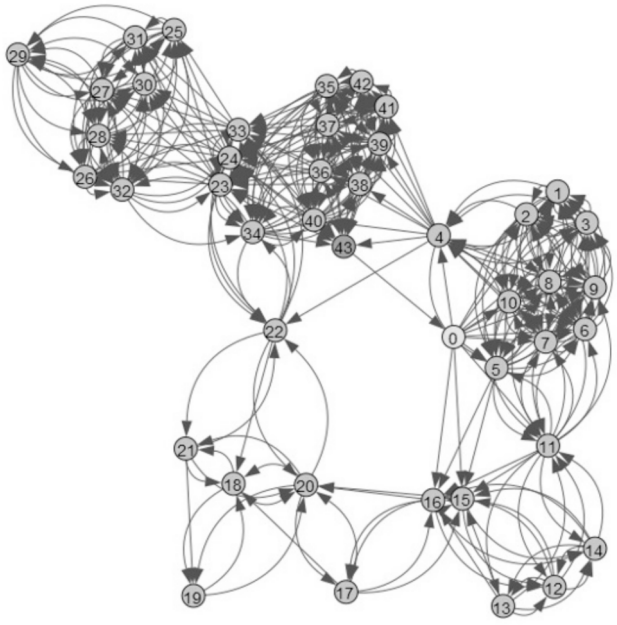

Fig. 3. Underlying graph for problem instance cgl_44, with the start node 0 and the end node 43.

production constraints, we must add *boundary constraints* related to the link between the campaigns.

It is not the subject of this work to go into the mathematical functions that are used to model the scheduling costs, which will be known and come in the input instances, but to describe the methods to solve the optimization problem. All the costs are defined in a square cost matrix that stores the cost of sequencing together each pair of coils. This cost is not symmetric, i.e., it is not the same cost to run coil 0 and then coil 1 than vice versa.

The first priority for the schedule is to have no constraints violated, the second priority is to have minimum cost. If a schedule does have one or more constraints violated, that is, is not *feasible*, different actions can still be taken by the CGL experts, like introducing linking auxiliary coils, looking for more coils to try to fix the issue, or eliminating coils from the schedule. All these actions have a huge cost for the factory in terms of yield and service, and a significant loss of time for the experts.

We can easily make an analogy of the problem described with the Asymmetrical Travelling Salesman Problem (ATSP), in which the target is to visit a set of cities minimizing the total cost of the trip, knowing the cost or distance between each pair of cities. In the ATSP, the constraints would forbid to travel directly between certain cities, and the theoretical problem can be defined as a Constrained Asymmetric Travelling Salesman Problem (CATSP). It is important to note that in the TSP we look for a *cycle*, that is, we account for the cost of completing the tour from the last city to the initial city, while in the CATSP described we look for a *path*: the cost from last coil to first coil is not accounted. The problem described so far is addressed by Fernandez et al. (2014) and Álvarez-Gil et al. (2022a).

The problem extension we introduce in this work is related to the linking of the CGL campaigns. The order of the campaigns is of high importance and is decided by the plant planning experts, who must take decisions concerning coils due dates, flow of campaigns in upstream facilities, planned downtimes, etc. As the CGL process is continuous, the end coil of one campaign must link to the start coil of the next campaign, still respecting all the standard scheduling constraints. Even though the schedules for each campaign are made *independently*, they must be consistent with the campaign linking, making sure that consecutive campaigns do link. We will call this linking requirement *boundary constraints* (BC) to the CATSP and the problem CATSP-BC. The linking of schedules, as we will show in the computation analysis, has a big impact in the problem complexity. In fact, it poses a challenge to the algorithm that otherwise would perform fine. Three requirements are possible:

1. The start coil and the end coil are free to be chosen by the algorithm. This is the case with campaigns in which the coils are very similar and the link to other campaigns is not critical. The scheduler leaves the algorithm complete freedom to optimize.
2. The start coil of the schedule is imposed. This is the usual case when the coil properties in the campaign is very heterogeneous, and the scheduler must make sure that the start coil of the schedule respects all constraints with the end coil of the previous schedule.
3. Both the start and the end coils of the schedule are imposed. This is common when the expert knows that the next campaign has limited connectivity with the current campaign he is scheduling, or when a special start is required in the next campaign. For instance, some campaigns that are run “wide-to-narrow” regarding the strip width: because of this, the previous campaign must end in a coil with maximum width. It is always the decision of the scheduler to set the convenient end coil, which must have connectivity with the coils of the next campaign.

Fig. 2 illustrates option 3. Campaign A runs first, then B and then C. We must make sure that last coil of campaign A links with first coil of campaign B; we must assure that last coil of campaign B links with first

coil of campaign C, and so on.

3.3. Formal definition

A formal definition of the problem is: given a directed weighted graph G = (V, E), where V is the set of vertices or nodes and E is the set of edges or arcs between nodes, and two nodes $s,e \in V$, find the minimum-cost Hamiltonian path that starts in s and ends in e. The set V corresponds to the set of n coils to be sequenced ($|V| = n$). Each arc (i,j) in the set of arcs E is weighed with the transition cost c_{ij} between coils i and j, and the arc exists if and only if the transition between the two coils is allowed.

The adjacency matrix A of the graph stores the information of allowed transitions for each pair of coils, having $a_{ij} = 1$ for existing arcs between nodes i and j, and $a_{ij} = 0$ if there is no arc.

The cost matrix C associated to the graph stores the information of transitions costs for each pair of coils, that is, the weights of the arcs in set E, having a cost $c_{ij} \geq 0$ for existing arcs between nodes i and j. By convention, the cells of the matrix corresponding to forbidden transitions (whose equivalent cell in A has $a_{ij} = 0$), will have $c_{ij} = -1$. This practical convention allows an easy representation of the problem with a single cost matrix C from which we can easily obtain the adjacency matrix A and build the graph G.

It is important to note that in the CATSP-BC problem, the constraints only forbid two coils i and j to be sequenced together, that is, the pair (i, j) must never appear in the solution: a sequence $S = \{0, 1, 2, 3, \cdots i, j, \cdots n-1\}$ would be an infeasible sequence, violating a scheduling constraint. Contrarily to the Sequencing Ordering Problem (SOP) introduced by Escudero (1988), where precedence constraints forbid a coil i to be sequenced at any position before a coil j, here two coils i and j with no transition (i,j) between them can be arranged at any position as long as the sequence does not include the nodes i and j as consecutive in it. As the problem is asymmetric, transition (j,i) might be perfectly valid regardless that (i,j) is forbidden.

If we add the boundary constraints to the problem, the solution sequence must be then

$$S = \{start, 1, 2, 3, \cdots end\}$$

That minimizes the total cost $C = \sum c_{ij}$, with $a_{ij} = 1$ for each pair of nodes $(i, i+1)$ of the sequence.

The boundary constraints have total priority for the solution. This means that even if there are feasible solutions without these constraints, the start and end nodes must be respected no matter if that implies to deliver an unfeasible solution. The transition costs are provided for each pair of coils, matrix C, independently of the nodes chosen as start and end of the sequence. In this regard,

- We will find in E existing arcs outgoing from e, and incoming arcs to s.
- We may find in E the arc $(start, end)$ existing as well.

This only means that those coils can be produced together in the line, regarding the production rules. But the boundary constraints invalidate those arcs for each specific problem. That is, the BC are not reflected in the matrix C provided. This convention simplifies the data: the same cost matrix C can be used for any chosen boundary coils.

We must clarify that the CGL schedulers do not know in advance if the problem is feasible, that is, if there exist a sequence that respects all constraints. Determining this is actually an NP-hard problem. Should the problem be infeasible, i.e., no Hamiltonian path exists, the optimal solution is that one with minimum constraints violations. In this case, there are actions to be taken by the CGL schedulers, all of them costly to the company –in loss of yield and productivity, in service indicators, or in loss of time by the experts. This is a major reason to look for a feasibility-wise robust algorithm.

4. Algorithms

In this section we describe the algorithms under study, the reasons for their choice, and the new method proposed.

4.1. Choice of the algorithms and approach to the problem

The CGL original scheduling problem without boundary constraints has been already studied in Fernandez et al. (2014) and in Álvarez-Gil et al. (2022a). In these papers we can see detailed reasoning of why the algorithm of choice has been Ant Colony Optimization, in particular the Ant System (AS). In the latter, the authors present an evolution of AS called AS-IR embedding a local search named Interval Reconstruction (IR). The effectiveness of the IR local search assures feasibility in hard to solve instances, and is the algorithm in use at the CGL we are addressing in this work.

The IR local search is especially designed for feasibility and adds reliability in high-constrained scenarios while keeping the same parameterization that yields good performance of the base AS in medium or low-constrained scenarios. This parameterization has been tested and adjusted in all kinds of campaigns with very different costs and constraints structure. Our line of research intends to further evolve the AS for the CATSP-BC keeping this advantage in mind. We are aware of how parameters can improve performance, but we also want to avoid overfitting our AS, compromising the optimality in the regular scenarios.

For this reason, we conducted a preliminary analysis to assess the impact of parameterization and to get insights on our problem. Parameters α and β tune the influence of the pheromone information versus the heuristic, and ρ controls the pheromone evaporation and thus premature convergence. We set up the start and end nodes as described in Section 5. We discarded values of $\alpha > 1$ which is reportedly prone to stagnation (Stützle & Dorigo, 2004). We tested the AS and the AS-IR with all grid-like combinations for $\alpha = \{0.5, 0.7, 1\}$, $\beta = \{0, 0.5, 1, 1.5, 2, 4\}$, $\rho = \{0.5, 0.65, 0.8\}$, that is, 3x6x3 = 54 experiments, running them 30 times for each instance. This makes a total of 54x30x30 = 48,600 runs for each algorithm.

This preliminary analysis shows better performance in feasibility for certain parameterizations. In Table 1 we can see the results for $\beta = \{0, 2\}$, which are the most significant for our discussion; $\beta = 4$ gets the worst results by far and $\beta = \{0.5, 1, 1.5\}$ throw an intermediate performance between $\beta = 0$ and $\beta = 2$. We show (1) the total success rate per instance (%) –percentage of feasible solutions in all runs all instances–, (2) the number of instances in which more than 50 % of the runs were infeasible, and (3) the same latter indicator for a 90 % infeasibility.

We can observe that the best results are obtained with $\beta = 0$, an extreme setting that effectively removes the cost influence: all costs are equal to the algorithm, we are looking just for Hamiltonian paths, not for minimum cost Hamiltonian paths. This reveals that in these instances some high-costs arcs that belong to the Hamiltonian path are misguiding the construction heuristic –which usually selects costly arcs with very low probability, except if $\beta = 0$.

But, though setting $\beta = 0$ seems to avoid this *adverse-costs* issue, this is not necessarily the best configuration regarding minimization of costs. In Table 2 we can see a comparison in costs for the two configurations $\beta = 0$ and $\beta = 2$. For each instance we can see the best cost and the number of infeasible runs (*inf.*). Each cost of 100,000,000 stands for a constraint violation. The column *cost diff (%)* shows the difference in percentage of best cost for $\beta = 0$ with reference to $\beta = 2$. As expected, if the algorithm is less greedy on costs, feasibility improves but the optimal cost found increases notably. In 12 instances, the best costs found with $\beta = 0$ are over a 100 % higher than the best costs found with the greedier $\beta = 2$. We must remark that this is the recommended standard parameterization in Stützle & Dorigo (2004).

Now, we can easily imagine that if we had instances with the same nodes and arcs, but being the cheapest arcs only in the Hamiltonian

Table 1
Preliminary analysis of the impact of the AS parametrization on feasibility.

	β = 0									β = 2								
	ρ = 0.5			ρ = 0.65			ρ = 0.8			ρ = 0.5			ρ = 0.65			ρ = 0.8		
α	0.5	0.7	1	0.5	0.7	1	0.5	0.7	1	0.5	0.7	1	0.5	0.7	1	0.5	0.7	1
AS																		
success rate (%)	84	87	88	85	87	90	85	88	88	70	64	58	69	63	54	65	61	52
instances inf. ≥ 50 %	5	3	3	5	3	2	4	3	4	9	11	12	8	11	12	11	11	13
instances inf. ≥ 90 %	3	1	0	2	0	0	2	1	0	8	8	9	8	9	9	8	9	11
AS-IR																		
success rate (%)	88	92	92	90	92	92	89	93	92	82	80	76	81	79	75	81	77	74
instances inf. ≥ 50 %	3	3	3	3	2	1	3	2	2	6	6	8	6	6	8	6	7	8
instances inf. ≥ 90 %	1	0	0	1	0	0	1	1	0	4	6	5	4	6	4	4	6	4

Table 2
Comparison in costs for the AS-IR with parameter β = 0 and β = 2.

	AS IR β = 0		AS-IR β = 2		
Instance	best cost	inf.	best cost	inf.	cost diff (%)
cgl_17.txt	5602	0	5602	27	0
cgl_26.txt	6613	0	6522	30	1
cgl_28.txt	3953	0	3654	0	8
cgl_32.txt	7388	0	7128	26	4
cgl_33.txt	12,027	1	10,068	27	19
cgl_37.txt	10,476	0	5673	17	85
cgl_38.txt	7253	0	7253	0	0
cgl_43.txt	9079	0	7939	30	14
cgl_44.txt	15,444	17	100,009,102	30	
cgl_45.txt	9462	0	8596	19	10
cgl_47.txt	15,634	0	6061	1	158
cgl_48.txt	16,263	0	10,733	30	52
cgl_48b.txt	12,390	0	6013	4	106
cgl_50.txt	9186	15	6641	0	38
cgl_51.txt	26,988	0	12,668	0	113
cgl_51b.txt	13,599	0	5469	0	149
cgl_57.txt	22,391	0	9795	8	129
cgl_58.txt	8835	0	5093	0	73
cgl_60.txt	20,074	16	100,010,795	30	
cgl_66.txt	14,568	0	9368	5	56
cgl_70.txt	34,221	0	11,232	0	205
cgl_70b.txt	14,298	0	7258	3	97
cgl_72.txt	17,333	14	13,186	30	31
cgl_73.txt	18,952	0	8655	30	119
cgl_76.txt	31,602	0	11,752	1	169
cgl_78.txt	22,049	5	100,010,101	30	
cgl_81.txt	26,502	0	9191	6	188
cgl_88.txt	33,354	0	11,185	4	198
cgl_107.txt	34,086	0	7377	0	362
cgl_114.txt	38,135	0	11,341	0	236

path, a better configuration both for feasibility and costs would be β = 2. This tells that trying to solve the problem just by adjusting the parameterization does not seem to be enough. We would be overfitting the AS to our test problems. Following these considerations, we have dismissed applying automatic parameterization methods (Huang, Li, & Yao, 2020) like IRACE (López-Ibáñez et al., 2016) to address the BC problem.

An alternative solution is to try to characterize the input instances, classifying them in order to apply different optimal AS parameters for each type of instance. This is another interesting line of research that we did not follow, because (1) it is very difficult to get or to generate a heterogeneous and representative set of instances –easy and hard to solve, with diverse types of graphs and connectivity, with diverse cost structure– adequate to feed the automatic classifier, and (2) in our opinion the results shown cast doubts on finding a parameterization especially good both in feasibility and costs for *adverse-cost* scenarios, should we succeed in this classification problem.

Our line of research followed a different path: try to bring intelligence into the AS by looking at the underlying graph, adding searching mechanisms especially designed for feasibility and for the BC problem. In this sense, the algorithm we propose in this work performs the same way that the base AS in regular scenarios and, when feasibility is at stake, a graph analysis method guides the sequence construction –without any need to characterize the input in advance for this. This design strategy assures that the quality is never lost in the regular instances, and is the reason why we will not insist more on the AS parameters hereafter.

In the next three sections we describe the AS and the AS-IR, both slightly modified for the BC, and the new algorithm AS-GA we propose.

4.2. Ant System

The algorithms under study are built on the Ant Colony Optimization (ACO) framework, a population-based algorithm that performs stochastic constructive heuristics gradually improved by means of positive feedback (pheromone).

Algorithm 1 ACO algorithms framework

```
1. Set the parameters
2. Initialize the pheromone values
3. while (termination criteria not met) do
4.    PerformAntsSequenceConstruction
5.    PerformLocalSearch (optional)
6.    UpdatePheromoneValues
7end while
```

The Ant System (AS) (Dorigo, Maniezzo, & Colorni, 1996) is the first developed variant of the ACO family. We will formulate the AS metaheuristic as applied to the scheduling problem, following the framework in Algorithm 1. We have an input cost matrix C describing the problem, and a pheromone matrix T –for trails. A set of m ants perform the construction of a sequence for n iterations, starting with a random coil and continuing with next coil based on a pseudo-random decision. The probability of a candidate coil j to be chosen as the next one after coil i is given by

$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k}[\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}}, \; if \; j \in N_i^k \tag{1}$$

where τ_{ij} is the amount of pheromone on the arc $(i.j)$ stored in T and η_{ij} its heuristic information inversely proportional to the transition cost stored in C for each arc $(i.j)$, $\eta_{ij} = 1/c_{ij}$. The parameters α *and* β control the influence of the heuristic information versus the pheromone. For forbidden transitions, we still leave them as candidates, but assign a penalty cost (some huge cost HC) that yields the probability virtually zero. At some point during the sequence construction, if no candidate coils remain, the candidates will have all of them a heuristic value $\eta_{ij} = 1/HC$ and therefore the final choice for those forbidden transitions will depend only on the pheromone value τ_{ij} that each candidate transition has at that iteration. With this formulation, only valid transitions are chosen unless no other option exist.

After all the ants have completed their sequences, the pheromone matrix is updated. Before doing the pheromone deposition, we perform

the pheromone evaporation for all arcs (i,j) in T according to the equation:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij},\ \forall (i,j) \in T \quad (2)$$

where ρ is the parameter that controls the rate of evaporation. After the evaporation, we lay pheromone only for the arcs belonging to the best sequences. The number of best sequences n_{best} is a parameter of the algorithm. See Stützle & Dorigo (2004) for details on pheromone deposition and initialization recommendations.

For the CATSP-BC, a modification is needed to assure complying with the BC. The start node is always chosen as the first node, and the end node is never used as a candidate until the last step of the construction. See Fig. 4.

In the base AS, no local search is performed. Our choice for an effective local search is the AS variant described in next section. But for the analysis done in this work, we will also use the base AS as a reference and as a measure for the difficulty of the instances.

There is another variant of AS called Ant Colony System (ACS), with a greedier action choice and a slightly different pheromone deposition, that performs well in bigger instances for problems like TSP, but has serious problems of stagnation (Stützle & Dorigo, 2004). In general, as observed in small difficult instances, it is not able to explore adequately the solution space compared to AS. Because of this poorer performance, observed computationally in Álvarez-Gil et al. (2022a), we will not include ACS in our study.

4.3. Ant system with interval reconstruction

The Ant System with Interval Reconstruction (AS-IR) described in Álvarez-Gil et al. (2022a) is an evolution of the AS that embeds a local search especially designed to look for feasibility. The method focuses on feasibility but also allows to reduce costs. It is an innovative approach to local search because it addresses the local moves using a partial constructive mechanism. This proves to be very efficient for highly constrained scenarios, where other local search kind of moves fail to improve the sequence.

The base algorithm is the AS, to which the IR local search step is added, as in Algorithm 1. The IR is performed over the best sequence S^{best} of each iteration, before the pheromone update. The IR targets an arc with a constraint violation or a high cost, and defines one interval window W_1 around the nodes it links; a second window W_2 is placed elsewhere randomly. The length of both intervals is selected randomly over a predefined minimum and maximum length interval (parameterized). Then all the nodes in the two intervals (or windows) are put together in a bag $B = W_1 \cup W_2$ from where they are redistributed in the now empty intervals.

With probability $p = 0.5$ a window W_i is chosen, and a node is selected from B among all the nodes that are adjacent to the last node in W_i. If at some point no adjacent nodes remain at B, a random node is taken, which means a violation will be inevitably added. The node is removed from the bag and the process is repeated until emptying it, getting a new complete sequence S'. Then cost is computed, without forgetting to include the arc from the windows ends to the next nodes in the sequence, right after the windows. If cost (including penalizations due to violations) is reduced, the new S' is consolidated as the best improved sequence so far. The reconstruction is performed several times. Fig. 5 illustrates the method described.

The IR local search implementation is efficient, as it only computes differential improvements for its evaluations. And, being focused on violations to be fixed until a feasible sequence is found, the method is very effective in resolving constraints issues, as it proves in the computational analysis.

Concerning the BC, a modification is needed in the IR: if the second interval chosen includes the end node, we must make sure that it is never sent to the first interval, ant never placed at any other position than the last one. The logic for this is straightforward and deserves no further explanation. Although this requirement limits the moves of the IR, the method is still very powerful.

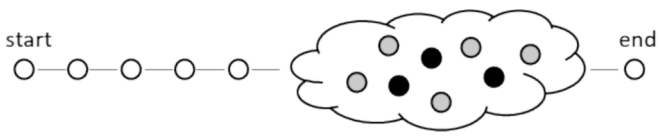

Fig. 4. The end node is always reserved for the last node.

4.4. Ant system with graph analysis

In this section we introduce the new Graph Analysis method and algorithm AS-GA developed for the CATSP-BC.

4.4.1. Motivation

The CGL problem seeks to find (minimum cost) feasible sequences. Being restricted to finish in a given node makes the task much harder, because the number of feasible solutions diminishes notably. To achieve feasibility, a constructive heuristics must take all the correct decisions from the early stages rather than just go appending adjacent nodes. The motivation of this work is to provide to the constructive heuristic the ability to look ahead, steering it in the correct direction towards the end node.

This is not an easy task: building a Hamiltonian path, as we have seen, is an NP-Hard problem. Checking the Hamiltonian condition for each candidate node at each construction step, for each ant, at each iteration, would be itself an NP-hard problem to solve every time. This panorama poses a real challenge. We need to come up with an effective yet efficient analysis able to guide the heuristic. We describe our solution in the next two sections.

4.4.2. A surrogate for feasibility applying graph analysis techniques

Our approach for guiding the heuristic to make right decisions regarding feasibility is based in the analysis of the graph containing the nodes still unused in the sequence under construction. We want to select at each step a candidate node that leaves a good remaining graph –a good set of remaining nodes– ahead, increasing the chances of feasibility.

Before going into details, we will give a few basic definitions (Faudree, 2003) related to graphs. A Hamiltonian path is a path that visits every vertex of the graph exactly once. In directed graphs there are two important types of connectivity. A directed graph is called weakly connected (WC) if replacing all of its directed edges with undirected edges produces a connected (undirected) graph. A vertex v is reachable from a vertex u if there is a directed path from u to v. The digraph is called strongly connected (SC) if each pair of distinct vertices is reachable from each other. We call WC components and SC components to subgraphs in a graph that are WC and SC respectively.

Once these notions set, we will see how we can make use of them to help find Hamiltonian paths.

We will consider hereafter that the BC applies to the problem, that is, the start and end nodes are fixed. To comply with this, we modify the underlying graph removing all arcs incident to the start node and all arcs incident from the end node. Hereafter we will always talk about directed graphs, referred indifferently as digraphs or simply graphs.

The construction of the sequence begins with the start node and continues selecting next nodes among the incident nodes from the last node of the sequence, at each step. In the highly constrained scenarios, though, at a certain point during the sequence construction, we may run out of incident nodes from last node: to continue we can only take one node not adjacent, violating a constraint. This lack of adjacent nodes is a consequence of all the previous decisions taken steps before. Involuntarily, we have left a remaining graph without possible Hamiltonian paths. Our target is to avoid arriving to this situation by trying to

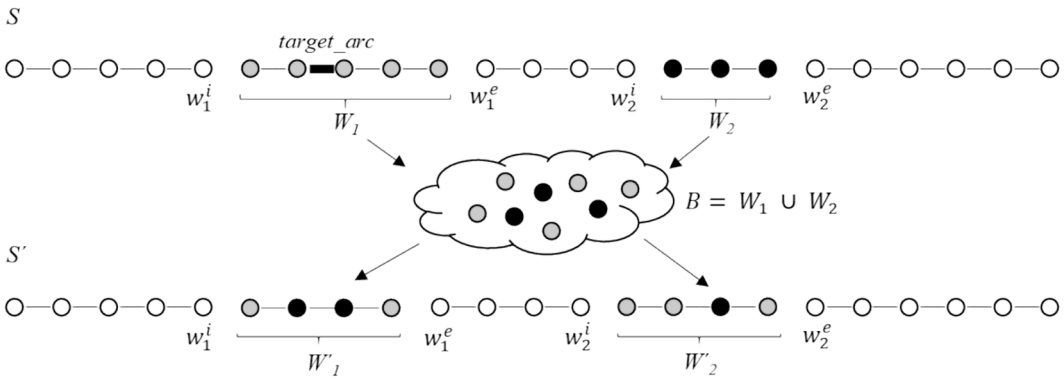

Fig. 5. One reconstruction try of the Interval Reconstruction.

identify in advance those bad decisions.

When we go adding nodes to the sequence, those nodes are removed from the remaining graph, together with all their arcs: the rest or nodes lose arcs, the graph becomes less and less connected. One idea for avoiding connectivity issues could be to prefer selecting nodes with poor adjacency –that is, low degree nodes–, before they eventually become 1- or 0-degree nodes. This stands both for incoming and outcoming arcs. The target is to avoid generating leaf nodes like node 2 in Fig. 6. This simple strategy, though, proves to be not very effective in computational tests. One reason is that it is usually too late to fix these issues when we detect them, often due to concurrent cases. This strategy would not detect either a disconnected remaining graph, like in Fig. 7.

A slightly more sophisticated strategy would be to analyze the minimum degree of the remaining graph. This idea has also shown to yield a poor performance in complex scenarios.

For a surrogate feasibility check to be reliable enough, we must look not at node level but at the big picture of the whole graph. A trivial necessary Hamiltonian condition is to always require a WC graph, avoiding getting a disconnected graph like in Fig. 7. This basic condition is correct but clearly insufficient. The graph depicted in Fig. 6 is WC but does not have a Hamiltonian path.

However, to require a SC graph as a condition for feasibility is too strong a condition. We can see in Fig. 8 how we could easily trace a Hamiltonian path from start to end, yet the graph is not SC: there are two SC components, depicted in color white and grey.

Inversely, SC graphs do not assure the existence of a Hamiltonian path, like we can see in two examples in Fig. 9. This is because in the definition of a SC graph, each pair of nodes is reachable from each other, but we can repeat nodes to do this.

Our target is to devise a surrogate for feasibility able to rule out graphs that spoil the Hamiltonian condition for sure.

The key to an effective method lies in the closing of a cycle between end and start, by adding an auxiliary arc from end to candidate node. With this arc, we allow going through all nodes from start to end and finally get back to the start node. As a result of closing the cycle, graphs become SC when a Hamiltonian path exists, i.e., drawing an arc (*end*, *start*) in Fig. 8. Although we cannot assure feasibility, i.e., cases like in Fig. 6, we are using a condition much stronger than WC and getting closer to the Hamiltonian condition.

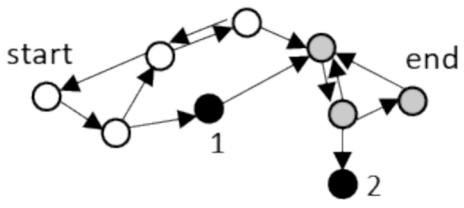

Fig. 6. Weakly connected graph with 4 strongly connected components.

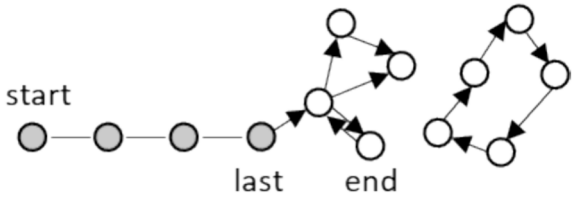

Fig. 7. Disconnected graph during the sequence construction.

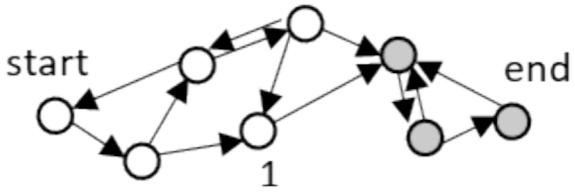

Fig. 8. Weakly connected graph with 2 strongly connected components.

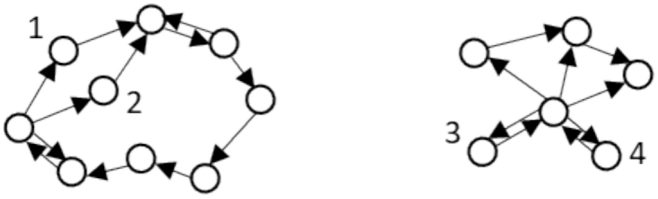

Fig. 9. Strongly connected graphs without Hamiltonian paths.

This first requirement of a SC graph, after adding the arc (*end*, *candidate*), is illustrated in Fig. 10. The nodes {start, … last} belong already to the sequence, and now we have 3 candidates {1, 2, 3} incident to last, from which to choose. Our analysis identifies not which candidate will assure a Hamiltonian path, but which one will surely spoil it, in order to reject it as a valid node to continue the sequence. We remove last node and its arcs and draw an arc from the end node to the candidate node being checked. We can see to the left how choosing candidate 1 renders the graph WC but not SC. Instead of one SC graph, we get three SC components. We see in black and grey color candidates 2 and 3 that cannot be visited starting from 1. This means that there not exists any Hamiltonian path from node 1 to end, and therefore we can reject node 1 as a good candidate. At the figure in the bottom center, we see the impact of choosing candidate 2, drawing an arc (*end*, 2). In this case, we get two SC components, and we can easily see that node 3 cannot be visited after node 2. Thus, we also reject this candidate. Finally, the graph to the right corresponds to the choice of candidate 3. We draw an arc (*end*, 3) and the result is a SC graph, in which all nodes are reachable

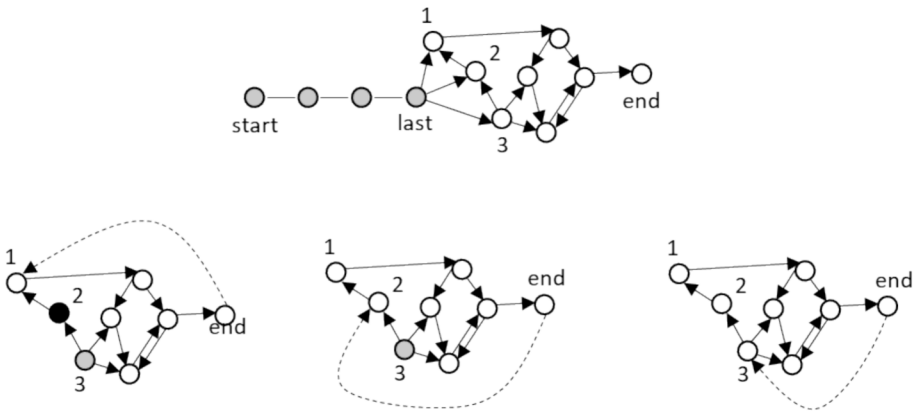

Fig. 10. Three candidates for last node, two of them (1 and 2) rendering the sequence unfeasible.

from all the rest. This is the only valid candidate in this simple example. Without much difficulty, we can spot a Hamiltonian path starting from node 3, continuing to node 2 and to node 1, and then on easily towards the end node.

Of course, the complexity of the graphs goes beyond simple cases like this, used only for illustration. Our computational tests have led us to our final surrogate feasibility check, which involves repeating this analysis one step further with the nodes that are incident from our candidate –that is, with the next step's future candidates. This, though of course far from a perfect check, proves to be sufficient to effectively guide the heuristics towards the end node reliably, as we will show in the computational analysis section.

The main drawback of this surrogate feasibility check is its computational cost: doing this analysis for each candidate at each step of the construction, for all the ants, in all the iterations, has a polynomial increase in the computation time. The performance slows down drastically, making unviable a brute-force application of the method. In the next section, we will show how we have embedded the method in the AS algorithm in an efficient way. Yet this method is not devised for very big sizes in short computation time. It will work efficiently below 150 nodes, which meets the requirements of our use case, where the size of the schedules is usually under 100 nodes.

The pseudo-code of our surrogate feasibility check, performing the graph analysis described, is shown in Algorithm 2 and Algorithm 3. The input to the surrogate Hamiltonian check is the partial sequence S, the remaining graph G, together withthe candidate and the end nodes. We will see in algorithm 5 where the check is called.

Algorithm 2 SurrogateFeasibility(S, G, candidate_node, end_node)

```
1. last_node ← LastNodeInSequence(S)
2. G' ← RemoveNode(G, last_node)
3. feasible ← IsAdjacent(G, last_node, candidate_node)
4. if feasible and | V' | ≤ 2: # cannot apply the surrogate check for less than 3 nodes
5.    return True
6.      feasible ← IsStronglyConnected(G', candidate_node, end_node)
7.    if feasible:
8.    next_feasible ← False
9.      NextCandidates ← GetAdjacentNodes (G', candidate_node)
10.     if NextCandidates ≠ Ø:
11.     next_feasible ← False
12.       for next_candidate in NextCandidates:
13.       next_feasible ← IsStronglyConnected (G', next_candidate, end_node)
14.         if next_feasible:
15.         Break
16.           end for
17.       feasible ← next_feasible
18.       return feasible
```

Algorithm 3 IsStronglyConnected(G, candidate, end_node)

```
1. G' ← GetSubgraph(G, candidate)
```

(*continued on next column*)

(*continued*)

Algorithm 3 IsStronglyConnected(G, candidate, end_node)

```
2. G' ← DeleteOutgoingArcsEndNode(end_node)
3. G' ← AddArcEndNodeToCandidateNode(end_node, candidate)
4. is_connected ← IsStonglyConnected(G')
5. return is_connected
```

4.5. Embedding graph analysis in the ACO framework

To embed the graph analysis into the ACO framework we only modify the ants sequence construction, inserting our surrogate feasibility check. The new framework, shown in Algorithm 4, barely differs from the one in Algorithm 1.

Algorithm 4 AS-GA algorithm framework

```
1. Set the parameters
2. Initialize the pheromone values
3. while (termination criteria not met) do
4.    PerformAntsSequenceConstructionGraphAnalysis
5.    PerformLocalSearch (optional)
6.    UpdatePheromoneValues
7. end while
```

The main challenge for applying the feasibility check deals with doing it without an exponential growth in computation. In a first scheme, we would simply apply the feasibility check to all adjacent candidates, rejecting those with a negative result of the feasibility check, and then apply the AS selection method to the accepted nodes. But clearly this is a bad option in terms of computation, especially for the low constrained scenarios in which all or almost all the candidates are adjacent and therefore should be checked. Several key modifications allow to achieve a satisfactory performance in terms of computation time:

- The first modification to streamline this first rough design is to change the exhaustive check to all candidates, that would filter out those unfeasible, into a selection round that gets a candidate and then checks it, and finishes as soon as a candidate gets a positive result from the feasibility check. That is, we first perform the standard AS selection action to all adjacent nodes, and then apply the feasibility check, not the other way round. If the result is an acceptance we validate the candidate, otherwise we remove that node from the candidates list and repeat the *selection and check* procedure again.

This design leads to the AS-GA sequence construction logic shown in Algorithm 5. We are building sequence S, initialized to the start node. We first apply the feasibility check to the start node, stopping it any further if the result is negative. Once we have confirmed that there may

exist a *(start, end)*-Hamiltonian path we continue with next nodes. We first perform the standard selection method for next candidate from a bag B^{adj} with all the adjacent candidates to the last node currently in the sequence, and then perform the feasibility check for that selected node. If the node is accepted, we continue to next step; if it is rejected, we remove it from B^{adj} and repeat this selection logic with the rest of candidates until the feasibility check returns a positive result. In case there is no candidate acceptance for any of the nodes in B^{adj}, we continue with a random non-adjacent node, and we do not perform any more the graph analysis for this ant.

Algorithm 5 PerformAntsSequenceConstructionGraphAnalysis(G, start_node, end_node)

```
1. S ← {start_node}
2. feasible ← False
3. if SurrogateFeasiblity(S, G, start_node, end_node)
4.    feasible ← True
5. G ← RemoveNode(G, start_node)
6. while S not complete
7.    last_node ← LastNodeInSequence(S)
8.    Bag^adj ← GetAdjacentNodes(G, last_node)
9.    Bag^adj ← Remove(Bag^adj, end_node)
10.   if Bag^adj = Ø
11.      feasible ← False
12.      candidate ← SelectRandomNodeNotAdjacent(V)
13.   else
14.      candidate ← SelectNextNode(Bag^adj)
15.   if feasible
16.      while (not SurrogateFeasiblity(S, G, candidate, end_node) and Bag^adj ≠ Ø)
17.         Bag^adj ← Remove(Bag^adj, candidate)
18.         candidate ← SelectNextNode(Bag^adj)
19.         if Bag^adj = Ø
20.            feasible ← False
21.            candidate ← SelectRandomNodeNotAdjacent(V)
22.   G ← RemoveNode(G, last_node)
23.   S ← Append(candidate)
24. S ← Append(end_node)
25. return S
```

The efficiency improvement achieved with this scheme is significant but still insufficient.

- A second important streamlining choice lies in the number of iterations needed. The algorithm can be stopped at a target time or be run for a fixed number of iterations. In the base AS, a factor of success is to run efficiently numerous iterations in short time, hoping to eventually converge to a feasible solution. If a more robust approach like AS-GA achieves feasibility already in the first iterations, as we will see, only a few iterations suffice, focused mainly on cost reduction.
- A third streamlining choice is that with this method we need fewer ants and, besides, not all ants need perform the graph analysis. This is possible only because the GA is effective enough: as we will see in the computational tests, in all instances and all runs, a feasible solution is achieved by a confident percentage of the ants. With this good performance, we do not need all the ants perform the analysis. Some percentage of the ants, i.e., 50 %, may be devoted to scout for feasible solutions with the GA capabilities, while the rest of the ants do not need perform the analysis; they just rely on the AS learning process (*stigmergy*) to build good solutions, exploiting the results of their companion GA ants.
- A fourth implicit streamlining mechanism comes built-in, derived by the infeasibility cases: whenever we run out of candidates at a certain construction step, the feasibility check is not useful anymore, and is deactivated. This means that once an ant fails in feasibility, the selection loop is not performed anymore.

The AS-GA run in our computational analysis includes all these efficiency methods. Yet there is still room for more future streamlining. Parallel computation should significantly reduce the computation time. Ants running in parallel in different cores allow an easy reduction of the total computation time; there are no drawbacks for the AS-GA design regarding this. But for the purpose of this work, to carry out a fair comparison with the other algorithms, no parallel computation has been implemented in any of the AS variants under study.

A clear improvement to the AS-GA is to add the IR local search, which at first does not have any drawback other than adding extra computation. The computational cost is small compared to the GA surrogate feasibility check, and the implementation is straightforward, just adding the IR mechanism as the local search of choice in the scheme shown in Algorithm 4. We have called this variant the AS-GA-IR.

We want to remark that this feasibility graph analysis is valid for any constructive heuristic that builds sequences from start to end –as different to insertion constructive heuristics. That is, it is suitable to be embedded into constructive metaheuristics such as Greedy Randomized Adapted Search Procedures (GRASP) (Feo & Resende, 1995). In this method, a greedy constructive heuristic is randomized and run throughout multiple iterations, adding local search improvements after each construction phase. The GRASP construction phase builds solutions from scratch by selecting next candidate from a Restrictive Candidate List (RCL). Embedding our graph analysis method into the GRASP would be similar to its embedding into the use case AS: in the GRASP construction phase, we substitute each (one-time) candidate selection from the RCL for a selection round, in which we iteratively select a tentative candidate and perform on it the graph-based feasibility check, until the check is positive. Only then the tentative candidate is effectively selected.

5. Experimental analysis

In this section we show and analyze the results of the computational tests, comparing the 3 algorithms described in section 4 plus a fourth algorithm combining the IR local search and the GA method. Aside to the summary results, we illustrate the GA method at work with a couple graphs analyzed during the construction of the sequences. We also analyze details in the performance of the algorithms regarding feasibility.

For this study, we have chosen to use the 30 instances published in Alvarez-Gil et al. (2022-april), which have been selected because of their difficulty from daily schedules run at the line. Each instance is a cost matrix named as *cgl_n*, where *n* is the size of the problem (number of coils to be sequenced), ranging from 17 coils to 114 coils.

For the campaign BC, we have chosen in all instances start node = *0*, that is, the first node of the matrix, and end node = *n-1*, that is, the last node, because we know in advance that there is at least one *(0, n-1)*-Hamiltonian path. This is easy to see at first glance looking at the cost matrix, where we can check that none of the elements over the diagonal are constraints, which makes the trivial solution $S = \{0, 1, 2, 3 \ldots n-1\}$ feasible. The only exception for this is instance *cgl_38*, in which only node 1 is reachable from 0 while visiting all nodes, and so we have set end node = 1 for this instance.

We want to remark that we can set multiple different BC problems using the same instance, but for the experimental analysis we have chosen one specific BC.

5.1. Experiment setting

Our aim is to analyze what improvement in performance our graph analysis method brings. We have a common implementation of the AS in which we can activate or deactivate the GA method and the IR local search. For the computational analysis, thus, we will be comparing 4 algorithms: the base algorithm AS, the AS-IR algorithm with the IR local search, the AS-GA hybridized algorithm we propose in this work, and the AS-GA-IR which embeds both the GA capabilities and the IR local search.

We have run our computational analysis in an Intel(R) Xeon(R) CPU E5-2695 v4 @ 2.20 GHz machine with 32 GB of RAM.

The 4 algorithms have the same AS parameterization. We have set up the standard parameters for the AS defined in Stützle & Dorigo (2004), which are $\alpha = 1$, $\beta = 2$, $\rho = 0.5$, and $m = n$, being m the number of ants and n the size of the instance. The number of ants n, though, as discussed in the previous section, is limited for the AS-GA and AS-GA-IR to a maximum of 40 ants in the instances with more than 40 nodes, as a streamlining strategy. This means that the AS-GA and the AS-GA-IR always run with fewer or equal of ants than the AS and the AS-IR. The number of best ants set up is $n_{best} = 1$.

The parameters used for the IR in the AS-IR are the ones used in Álvarez-Gil et al. (2022a): *max_window_len* $= n/3$, *min_window_len* $= 0$, *max_improvement_tries* $= 10$ and *max_reconstruction_tries* $= 30$. We can see a summary of the parameters used in Table 3.

For assuring statistical significance, we have run the 4 algorithms 30 times each on every instance. All runs have the same fixed budget time of 180 s, no matter the size of the instances. This is an assumable computation time for the final user. It is higher than the budget time of 120 s set in Álvarez-Gil et al. (2022a) due to the higher complexity of the problem.

Additionally, to have a reference of the optimum value for each problem, we have run the 30 instances on an exact solver, in our case IBM ILOG CPLEX Interactive Optimizer 22.1.1.0, in a machine with a processor 11th Gen Intel(R) Core(TM) i7-1185G7 @ 3.00 GHz 3.00 GHz and with 32 GB of memory. The solver is stopped after 1 h. We have formulated the problem as an Integer Linear Programming (ILP) problem following the formulation by Miller, Tucker, & Zemlin (1960), with minor adaptations to look for a Hamiltonian path with given start and end nodes, instead of a Hamiltonian cycle.

5.2. Results

5.2.1. Summary comparison of the algorithms

In Table 4 we can see the results of the computational test. For each instance we can see the *best cost* and the number of infeasible runs (*inf.*) for the 4 algorithms. Each cost of 100,000,000 stands for a constraint violation. In Table 5 we show a summary of the feasibly success rate (%).

The requirement from the CGL is to run a robust algorithm able to assure feasibility in every run. The scheduler does not know in advance if there is a feasible arrangement for the coils of the campaign to be schedule. It will run the algorithm just once, and then will fix violations in case there are. Many times, the set of coils is not feasible, and no other option exists. But a robust algorithm able to reduce violations at maximum in the hardest scenarios will make the difference and save scheduling costs and time to the CGL.

This is the reason why we look at the number of runs that have not found a feasible sequence, additionally to the best cost found in all the 30 runs.

We can also see the result from the exact solver CPLEX, showing the lower bound (LB) and upper bound (UB) obtained after running a maximum of 3600 s. This gives us a reference of how difficult the instances are, and how close to the known optimum values the algorithms studied are. In the cases in which the best cost is found, we show it in column *best cost, otherwise we write* "-". The exact solver finds the solution in 6 of the 30 instances (the three smallest and other three of small-medium size), all of them obtained in a computation time below 180 s. The rest of instances are not resolved in the budget time of 3600 s.

Table 3
List of parameters for the experimental analysis.

parameter	value	description	used in algorithm
α	1	coefficient for pheromone weight	all
β	2	coefficient for heuristic weight	all
ρ	0.5	coefficient for pheromone evaporation	all
max_window_len	n/3	maximum window length for IR move	AS-IR, AS-GA-IR
min_window_len	0	minimum window length for IR move	AS-IR, AS-GA-IR
max_improvement_tries	10	maximum improvement tries for IR move	AS-IR, AS-GA-IR
max_reconstruction_tries	30	maximum reconstruction tries for IR move	AS-IR, AS-GA-IR

The AS is only competitive in occasional runs in a few instances. In Table 4 we can see that the base AS fails to find feasible sequences in some run in 22 out of the 30 instances: only 8 instances get 0 infeasibility runs. Its total success rate is only 56.89 %.

The AS-IR, specifically designed for feasibility in complex instances (without BC), finds more feasible sequences, succeeding in 79.67 % of the total runs. Yet in 3 instances it never finds a feasible solution (30 infeasibilities in the 30 runs) and in 1 more instance it only gets a feasible solution in 2 runs out of 30 (see 28 infeasibilities). We must recall that if we do not set a start and end node, in these instances the AS-IR is robust, feasibility-wise (Álvarez-Gil et al., 2022a).

The AS-GA reaches a 99.67 % feasibility success rate, failing only in 3 runs out of the total 900 runs, 30 per instance. Regarding costs it is very competitive, though the AS-IR gets to find better cost solutions in 8 instances.

As we were expecting, combining both hybrid AS algorithms in the AS-GA-IR results in the best of the 4 algorithms: the feasibility success rate is 99.78 %. Hybridizing them is straightforward. Although the GA method has an important computational cost, the IR local search is quite efficient in comparison, so there is no major drawback, even computationally, in applying the IR local search to improve the already good-quality solutions built with the GA method.

Regarding efficiency, we want to remark that in the budget time chosen of 180 s, the number of iterations run for the bigger instances when using GA decreases with size.

Regarding costs, in Table 6 we show the number of best cost results for all instances, both in average (only feasible sequences considered) and in minimum value of all runs. Minimum values are those in Table 4, the table for average values is omitted for simplicity. We can observe that the IR local search, as expected, helps improve the sequences built by the ants, reducing costs. Yet the AS-GA is competitive in costs, getting minimum best cost in 11 out of the 30 instances without the help of any local search improvement. This is because the best costs achieved do not sometimes differ notably.

5.2.2. Two examples of the graph analysis method at work

In Fig. 11, we can see the GA method in action in instance *cgl_28*. The end node is node 27. The only good candidate here is node 2, because no remaining node is incident to it except last node. In the graph (following the GA method described) we cannot see last node, only the candidate being checked; but we just need to know that in this case last node links to all the nodes seen in the graph –all nodes are candidates, making the decision at this stage very difficult. The selection loop chooses several times some node that renders the graph not SC, like the case depicted to the left. Selecting node 9 splits the graph in two SC subgraphs: one with node 2, and the other one with the rest of nodes. The surrogate feasibility check detects this issue, discards the candidate, and tries other candidates: nodes 11, 5, 4, etc., with the same negative result from the check. Only when candidate node 2 is selected and checked the graph evaluated is SC, picture to the right. We can see that in this case, following the described method, an arc is drawn from end node 27 to node 2 because the latter is the candidate node; this arc renders the graph SC. When choosing other candidate nodes, no arc enters node 2 anymore, which makes it impossible to visit node 2 in the future.

In Fig. 12, we can see in action a counterexample of how the surrogate feasibility check is not equivalent to the Hamiltonian check. To the left, we detect that candidate node 23 spoils the SC condition, splitting the graph in two SC components. We correctly reject this node 23. To the right, we see that choosing node 42 fixes the issue, getting a

Table 4
Best cost and number of infeasible runs for each instance and algorithm.

	CPLEX			AS		AS-IR		AS-GA		AS-GA-IR	
Instance	LB	UB	best cost	best cost	inf.	best cost	inf.	best cost	inf.	best cost	inf.
cgl_17.txt	5602	5602	5602	**5602**	27	**5602**	0	**5602**	0	**5602**	0
cgl_26.txt	6522	6522	6522	100,005,242	30	**6522**	0	6535	0	**6522**	0
cgl_28.txt	3654	3654	3654	**3654**	0	**3654**	0	**3654**	0	**3654**	0
cgl_32.txt	5482	7128	–	8342	26	**7128**	0	7147	0	**7128**	0
cgl_33.txt	10,067	10,068	–	10,874	27	**10,068**	0	10,083	0	**10,068**	0
cgl_37.txt	5673	5673	5673	6677	17	**5673**	0	5841	0	**5673**	0
cgl_38.txt	6089	7253	–	**7253**	0	**7253**	0	**7253**	0	**7253**	0
cgl_43.txt	5758	7343	–	100,004,005	30	7939	18	**7351**	0	7618	0
cgl_44.txt	10912,9	10,914	–	100,009,102	30	100,009,102	30	11,132	3	**11,122**	2
cgl_45.txt	8596	8596	8596	8622	19	**8596**	8	8619	0	8619	0
cgl_47.txt	5841	5992	–	**6061**	1	**6061**	0	6323	0	**6061**	0
cgl_48.txt	10,322	10,604	–	100,010,180	30	10,733	7	**10,729**	0	11,017	0
cgl_48b.txt	5073,9	5578	–	6256	4	6013	1	5990	0	**5900**	0
cgl_50.txt	6340	6610	–	6664	0	6641	0	**6622**	0	6624	0
cgl_51.txt	12666,7	12,668	–	12,705	0	**12,668**	0	12,996	0	12,868	0
cgl_51b.txt	4825	4991	–	5812	0	**5469**	0	5503	0	5496	0
cgl_57.txt	8371,5	9722	–	11,209	8	9795	5	**9739**	0	**9739**	0
cgl_58.txt	5093	5093	5093	**5093**	0	**5093**	0	**5093**	0	**5093**	0
cgl_60.txt	9762,1	11,601	–	100,010,923	30	100,010,795	30	**12,073**	0	12,112	0
cgl_66.txt	9191	9290	–	10,257	5	**9368**	0	9465	0	9420	0
cgl_70.txt	10,236	10,237	–	11,391	0	11,232	0	11,104	0	**10,445**	0
cgl_70b.txt	5798,7	6618	–	7345	3	7258	2	7470	0	**7104**	0
cgl_72.txt	8069	10,370	–	100,008,447	30	**13,186**	28	13,686	0	**13,186**	0
cgl_73.txt	6824	7978	–	100,006,715	30	8655	18	8227	0	**7985**	0
cgl_76.txt	9916,4	10,411	–	11,896	1	11,752	0	12,376	0	**11,224**	0
cgl_78.txt	9641,2	12,027	–	100,010,101	30	100,010,101	30	**14,272**	0	14,746	0
cgl_81.txt	7883	8365	–	10,979	6	9191	3	**8716**	0	8929	0
cgl_88.txt	10144,7	11,092	–	11,506	4	**11,185**	3	12,284	0	12,334	0
cgl_107.txt	4419	7057	–	**7718**	0	7377	0	8076	0	7859	0
cgl_114.txt	10,930	11,287	–	11,592	0	**11,341**	0	12,577	0	12,833	0

Table 5
Success rate (%) of feasible sequences found in all instances, all runs.

	AS	AS-IR	AS-GA	AS-GA-IR
feasibility success rate (%)	56.89	79.67	99.67	99.78

Table 6
Best cost summary scores per algorithm, for average cost (feasible sequences only) and minimum cost.

	AS	AS-IR	AS-GA	AS-GA-IR
Best avg cost	2	16	5	14
Best min cost	5	17	11	17

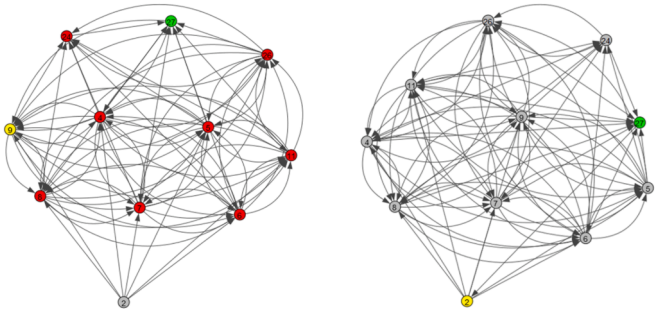

Fig. 11. Example of selection round for the GA method.

SC graph. But we can easily see that, whatever node we actually select, the path towards the end node is doomed to eventually fail in subsequent steps: node 22 is clearly dividing the graph in 2 sets (articulation point), being the only way to visit the nodes {15, 19, 21} at the bottom. Once it gets selected at some step ahead, we will not be able to go back to end

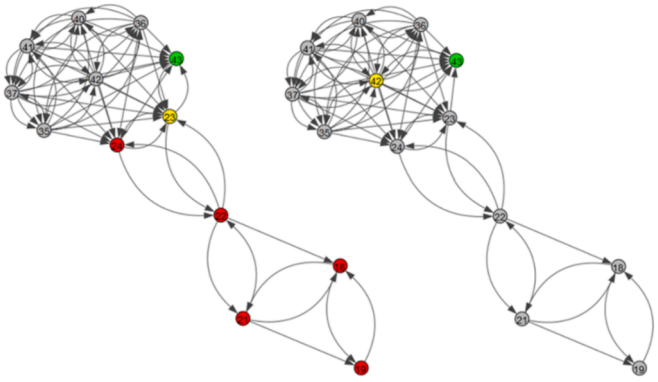

Fig. 12. Construction step correctly rejecting a surely wrong node, but doomed to fail eventually because of an articulation point, an issue originated in previous steps of the heuristic.

node 43. That is, our surrogate deems the remaining graph Hamiltonian, but it is actually not the case.

5.2.3. *Feasibility and convergence*

Finally, in this section we briefly analyze the convergence of the algorithms, by looking at the number of violations the ants obtain at each iteration. We will show two examples focusing on the AS-IR and the AS-GA, which are the two methods we are most interested in comparing in our work: the reference algorithm in use, and the new algorithm we propose. We focus on the AS-GA instead of the AS-GA-IR so as to analyze the goodness of the graph analysis method introduced, regardless the local search refinement that the IR method can add up. The objective of this paper is to introduce this graph analysis, which can be applied in other *meta*-heuristics, rather than just to look for the best performant algorithm for this use case.

To illustrate their behavior, we will look at two plots: (1) the number

of violations in each ant per iteration (characterized through three values: minimum number of violations among all ants, average number of violations, and maximum number of violations); and (2) number of ants getting feasible solutions per iteration, that is, sequences without violations. We zoom in on the first 50 iterations for a better view.

In Fig. 13 we can see an example of the performance of the AS-IR in instance *cgl_60*. In the right chart we can see that the AS-IR has not found any feasible sequence in any run. In the left chart we look at the number of violations made in each ant's sequence. At the first iteration the 60 ants of the AS-IR build sequences within a range of 3 and 8 violations, and an average of slightly over 5 violations. No ant gets a feasible sequence (0 violations). We can see that the AS learning process makes the ants improve and get ever decreasing violations along time, but the convergence does not reach 0 violations eventually. We are only showing the 50 first iterations here, but the curve does not go eventually down to 0, as Table 4 shows. We see that the complexity of the problem impacts the convergence, because the minimum violations perform somewhat erratic for this value. To the right, we confirm that no ant gets feasible solutions at any iteration.

In Fig. 14 we see an example of the performance of the AS-GA in the same instance *cgl_60*. What we first notice is that the number of total iterations does not reach even 20, due to the computational cost of the GA method. In the right chart, we can observe that the AS-GA finds feasible sequences (no violations) from the first iteration. We are running 40 ants every iteration, among which only 20 run the GA method. The number of feasible ants increases the along iterations from 3 ants at iteration 0 to 13, 15, 17… in subsequent iterations, and stabilize in an oscillating value that never surpasses 20 ants. This shows how difficult the instance is: the ants not performing the GA method do not get to find feasible sequences. Regarding the number of violations, in the left chart, the maximum number goes down to 3 quickly: the pheromone feedback helps reduce violations to those ants; compare with the AS-IR where the maximum goes down much more slowly and stabilizes at 4 or 5.

In Fig. 15 and Fig. 16 we compare again two runs of the both algorithms for the instance *cgl_44*. In these runs not only do both algorithms converge to a low average number of violations per ants, with minimum 0 violations (feasible sequences found), but they do it in almost all ants.

This convergence in almost all ants finding feasible sequences does not occur in the 30 runs, though. In Fig. 17 we can see a different run of the AS-GA for instance *cgl_44*, in which only a fraction of the ants achieves feasibility, while other ants explore without success. We are running 20 GA ants, and we can observe how the number of ants with feasible solutions stagnates in 20, from the total of 40 ants.

In our opinion, the fact that not all runs converge into all ants getting feasible sequences is a measure of the high complexity of the instances. It also highlights the effectiveness of the GA method we propose in this work, which adds strong reliability to the AS algorithm: the 20 ants running the GA method do find feasibility.

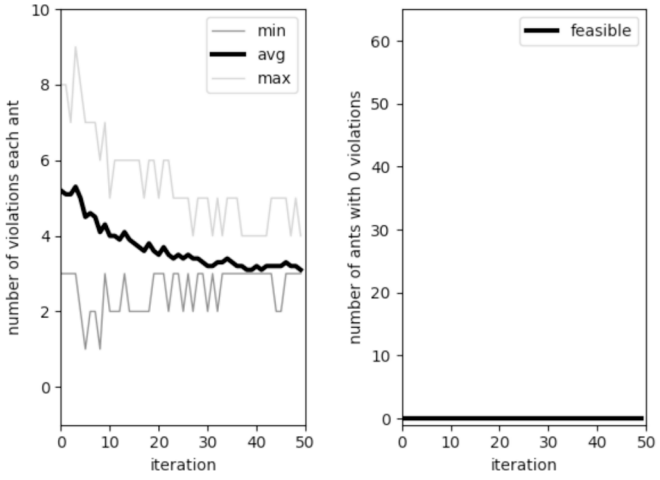

Fig. 13. Performance of the AS-IR: run #18 for instance cgl_60.

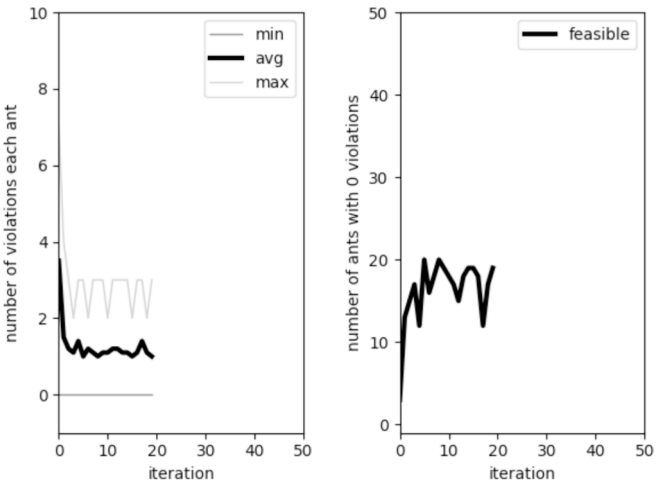

Fig. 14. Performance of the AS-GA: run #5 for instance cgl_60.

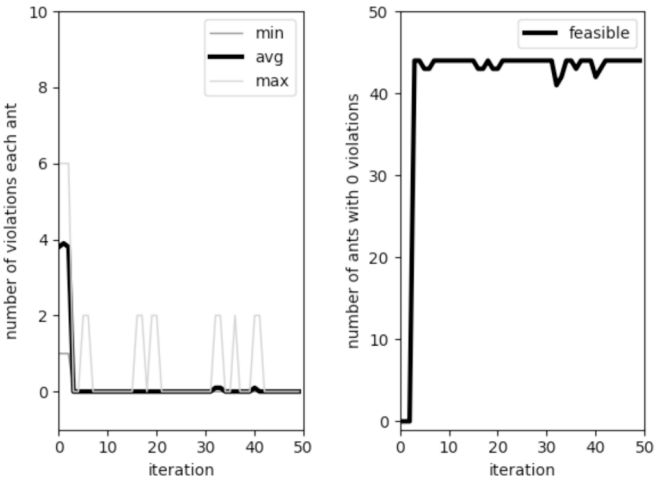

Fig. 15. Performance of the AS-IR: run #12 for instance cgl_44.

To sum up our analysis of the results, we can conclude that the AS-GA presented in this work is an algorithm very robust for the CATSP-BC for which it has been designed. The AS-GA finds a feasible solution in 99.67 % of the runs in all instances. The AS-IR alone is not so reliable: it fails to find a feasible sequence in 3 instances, and almost fails in other 2. Joining forces from both methods, the AS-GA-IR gets to improve the cost performance, which is what we expected when adding an effective local search over already feasible solutions.

Concerning efficiency, we must observe that the bigger instances run ever fewer iterations for the AS-GA and AS-GA-IR, due to the computational cost of the GA method. Nevertheless, the cost pays off because those few iterations suffice to assure feasibility and compete in costs.

6. Conclusions

This work addresses the scheduling optimization for a Continuous Galvanizing Line (CGL) at a steel making plant, in which we are required to link the different campaigns. This real-world problem we introduce can be translated into finding a minimum cost *(start, end)*-Hamiltonian path in the graph underlying to the scheduling costs matrix, where the imposed *start* and *end* nodes of the path are related to the campaign linking. This requirement significantly reduces the number of valid

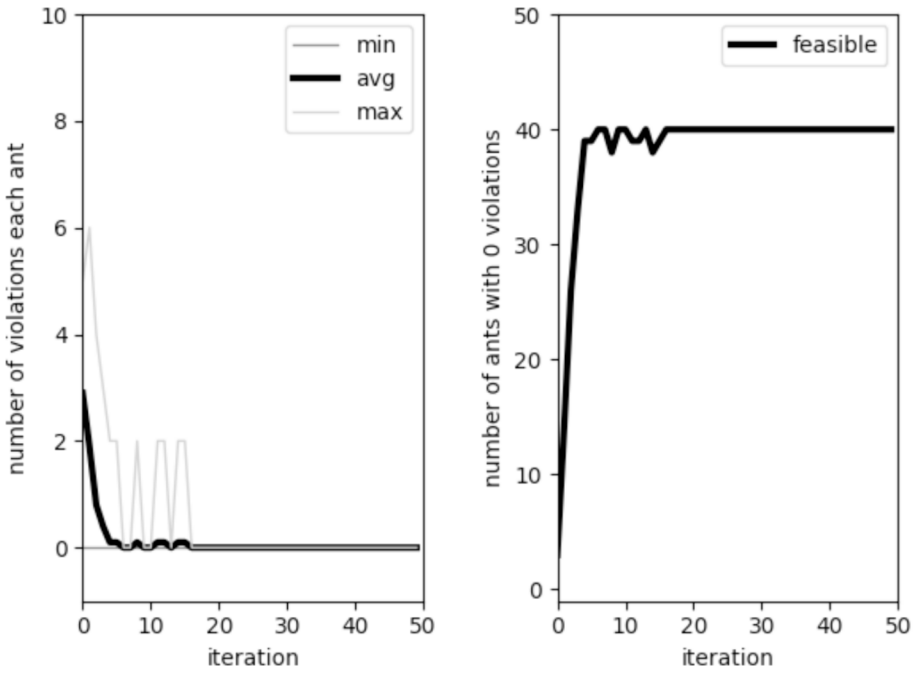

Fig. 16. Performance of the AS-GA: run #26 for instance cgl_44.

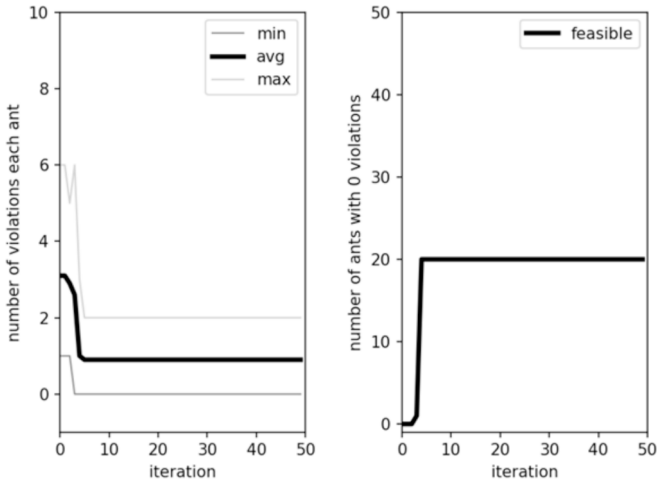

Fig. 17. A different run (#7) by the AS-GA at instance cgl_44 with no complete convergence of all ants.

Hamiltonian paths, making the problem harder to solve.

Many metaheuristics designed for the Traveling Salesman Problem (TSP) can be applied to solve this scheduling problem, like the Ant System (AS) algorithm. In the steel factory we are concerned with, the algorithm in use is the AS with Interval Reconstruction (AS-IR), an evolution designed for assuring feasibility in highly constrained scenarios –lately frequent due to an increasingly heterogeneous orderbook and tighter production rules. The campaign linking constraints impact the performance of the AS-IR, making it unreliable regarding feasibility.

Requiring a given end coil to the sequence does much harm to algorithms that, like the AS, are based in constructive heuristics. At first glance, the constructive heuristic can only keep aside the given end coil and add it after the sequence is finished. This easily results in getting a high cost or a constraint violation at the very end of the sequence. The AS relies on the learning mechanism (the positive feedback stored in the pheromone trails) to eventually fix this. But this ideal convergence does not always occur in the 30 challenging instances of this study. Serious problems of stagnation in (infeasible) local optima appear.

The motivation of this work has been to research how graph analysis techniques can assist the heuristics, helping them take the right decision at each step of the construction and look ahead to avoid compromising feasibility. Our research's ultimate purpose is to explore and broaden the knowledge bringing together graph theory and metaheuristics.

In this study we introduce a surrogate check for the Hamiltonian condition, *or feasibility surrogate check* (FSC). Checking if a graph is Hamiltonian is itself an NP-hard problem: this is the reason why we rely on a surrogate check. The method analyzes the graph of the remaining nodes to be visited and requires conditions on strong connectivity; a negative result rejects candidates that, if selected at that step, would spoil feasibility. We have embedded this FSC into the AS in the fashion of a non-Hamiltonian check, resulting in the AS-GA algorithm we propose in this work. The FSC manages to robustly steer the ants to the destination, enabling them to navigate more safely in challenging environments. The AS *stigmergy* mechanism brings convergence, making ever more ants achieve feasibility along the iterations. In the description of the AS-GA algorithm, we highlight the challenge posed by the computational cost the method introduces, and we detail the measures implemented to render it efficient enough for a short running time for the final user.

The results of the experimental analysis show that, though being computationally costly, applying the GA method amply pays off: running a test of 30 runs at each instance, the AS-GA assures a 99.67 % of feasible solutions, from the very first iterations. Only 3 runs fail, all at instance *cgl_44*, out of 900 total runs. The complexity of the graphs during the ants' construction is illustrated in the experimental analysis section and speaks about how easy it is at each building step to take the wrong decision towards feasibility, if no such analysis is performed. The high effectivity of the FSC makes the ants explore more in the feasible region of the solution space, which renders the AS-GA competitive in costs as well. The results show that the AS-GA clearly outperforms the base AS (56.89 % of success rate) and the AS-IR (79.67 %) in assuring feasible sequences, which is the first priority for the CGL. In some of the harder to solve instances, the latter algorithms fail to find any feasible solution at all. Additionally, we implement the hybrid algorithm AS-GA-IR, which in the comparison proves to inherit the goodness of both GA and IR methods, achieving 99.78 % success rate and reducing costs from

AS-GA.

The method we propose in this work is valid for any sequencing metaheuristic running constructive heuristics –as opposed to local search metaheuristics–, such as the Greedy Randomized Adaptive Search Procedures (GRASP). A key advantage of the GA method is that it does not rely heavily on the metaheuristic settings (the AS parameters in our case) for achieving feasibility. This allows to keep the best parameterization for standard instances and yet transparently be robust for hard to solve instances, without need to identify them as such. The main drawback of the method is that it is not scalable to very big instances. In them, few iterations can be run in the time budget of 180 s. Further research might explore how to render it yet more efficient –aside to the promising option of parallel computation–, to help reduce the running time. Another possible future line of research is to define a more sophisticated surrogate feasibility check adding more reliability without adding too much more computational cost.

CRediT authorship contribution statement

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

Data availability

Data will be made available on request.

Acknowledgements

This work was supported by the ERASMUS + Program via the project Academic System Resource Planning: A Fully-Automated Smart Campus (ASRP) [grant number UE-19-ASRP-598757].

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.cie.2024.110206.

References

Alba, E., & Chicano, F. (2007). ACOhg: Dealing with huge graphs. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation* (pp. 10–17). https://doi.org/10.1145/1276958.1276961

Álvarez-Gil, N., Álvarez García, S., Rosillo, R., & de la Fuente, D. (2022a). Sequencing jobs with asymmetric costs and transition constraints in a finishing line: A real case study. *Computers & Industrial Engineering, 165*, Article 107908. https://doi.org/10.1016/j.cie.2021.107908

Álvarez-Gil, N., Álvarez García, S., Rosillo, R., & de la Fuente, D. (2022b). Problem instances dataset of a real-world sequencing problem with transition constraints and asymmetric costs. *Data in Brief, 41*, Article 107844. https://doi.org/10.1016/j.dib.2022.107844

Applegate, D. L., Bixby, R. E., Chvatal, V., & Cook, W. J. (2006). *The traveling salesman problem: A computational study*. Princeton University Press.

Baker, K. R., & Trietsch, D. (2009). *Principles of sequencing and scheduling*. John Wiley.

Bang-Jensen, J., & Gutin, G. Z. (2009). Digraphs. *Springer*. https://doi.org/10.1007/978-1-84800-998-1

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 26*(1), 29–41. https://doi.org/10.1109/3477.484436

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.

Faudree, R. (2003). Graph Theory. In R. A. Meyers (Ed.), *Encyclopedia of Physical Science and Technology* (Third Edition, pp. 15–31). Academic Press. https://doi.org/10.1016/B0-12-227410-5/00296-9.

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

Fernandez, S., Alvarez, S., Díaz, D., Iglesias, M., & Ena, B. (2014). Scheduling a Galvanizing Line by Ant Colony Optimization. In M. Dorigo, M. Birattari, S. Garnier, H. Hamann, M. Montes de Oca, C. Solnon, & T. Stützle (Eds.), *Swarm Intelligence* (pp. 146–157). Springer International Publishing. https://doi.org/10.1007/978-3-319-09952-1_13.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability, (Vol. 174)*.. freeman San Francisco. https://bohr.wlu.ca/hfan/cp412/references/ChapterOne.pdf.

Glover, F. (1989). Tabu search—Part I. *ORSA Journal on Computing, 1*(3), 190–206. https://doi.org/10.1287/ijoc.1.3.190

Graves, S. C. (1981). A review of production scheduling. *Operations Research, 29*(4), Article 4.

Harjunkoski, I., & Grossmann, I. (2001). A decomposition approach for the scheduling of steel plant production. *Computers & Chemical Engineering, 25*, 1647–1660. https://doi.org/10.1016/S0098-1354(01)00729-3

Hejazi, R., & Saghafian, S. (2005). Flowshop-scheduling problems with makespan criterion: A review. *International Journal of Production Research, 43*(14), 2895–2929. https://doi.org/10.1080/00207540500056417

Huang, C., Li, Y., & Yao, X. (2020). A survey of automatic parameter tuning methods for metaheuristics. *IEEE Transactions on Evolutionary Computation, 24*(2), 201–216. https://doi.org/10.1109/TEVC.2019.2921598

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*(13), Article 13. https://doi.org/10.3390/app9132692

Karp, R. M. (1972). Reducibility among Combinatorial Problems. In R. E. Miller, J. W. Thatcher, & J. D. Bohlinger (Eds.), Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department (pp. 85–103). Springer US. https://doi.org/10.1007/978-1-4684-2001-2_9.

Kapanoglu, M., & Koc, I. O. (2006). A Multi-population Parallel Genetic Algorithm for Highly Constrained Continuous Galvanizing Line Scheduling. In F. Almeida, M. J. Blesa Aguilera, C. Blum, J. M. Moreno Vega, M. Pérez Pérez, A. Roli, & M. Sampels (Eds.), *Hybrid Metaheuristics* (pp. 28–41). Springer. https://doi.org/10.1007/11890584_3.

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., & Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives, 3*, 43–58. https://doi.org/10.1016/j.orp.2016.09.002

Maravelias, C. T., & Sung, C. (2009). Integration of production planning and scheduling: Overview, challenges and opportunities. *Computers & Chemical Engineering, 33*(12), Article 12.

Miller, C. E., Tucker, A. W., & Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM, 7*(4), 326–329. https://doi.org/10.1145/321043.321046

Okano, H., Davenport, A., Trumbo, M., Reddy, C., Yoda, K., & Amano, M. (2004). Finishing line scheduling in the steel industry. *IBM Journal of Research and Development, 48*, 811–830. https://doi.org/10.1147/rd.485.0811

Sleegers, J., & van den Berg, D. (2022a). Backtracking (the) Algorithms on the Hamiltonian Cycle Problem.

Sleegers, J., & van den Berg, D. (2022). The hardest hamiltonian cycle problem instances: The plateau of yes and the cliff of no. *SN Computer Science, 3*(5), 372. https://doi.org/10.1007/s42979-022-01256-0

Stützle, T., & Dorigo, M. (2004). Ant Colony Optimization.

Tang, L., & Gao, C. (2009). A modelling and tabu search heuristic for a continuous galvanizing line scheduling problem. *ISIJ International, 49*(3), 375–384.

Totten, G. E., Funatani, K., & Xie, L. (2004). *Handbook of Metallurgical Process Design*. CRC Press.

Valls Verdejo, V., Alarcó, M. A. P., & Sorlí, M. P. L. (2009). Scheduling in a continuous galvanizing line. *Computers & Operations Research, 36*(1), Article 1. https://doi.org/10.1016/j.cor.2007.09.006

Vandegriend, B., & Culberson, J. (1998). The Gn, mphase transition is not hard for the Hamiltonian cycle problem. *Journal of Artificial Intelligence Research, 9*(1), 219–245.

Wang, J., Fan, X., Zhang, C., & Wan, S. (2014). A Graph-based Ant Colony Optimization Approach for Integrated Process Planning and Scheduling. *Chinese Journal of Chemical Engineering, 22*(7), Article 7. https://doi.org/10.1016/j.cjche.2014.05.011.