Constructive metaheuristics for solving the Car Sequencing Problem under uncertain partial demand

Ignacio Moya[*,a], Manuel Chica[a,b], Joaquín Bautista[c]

[a] *Andalusian Research Institute DaSCI "Data Science and Computational Intelligence", University of Granada, 18071 Granada, Spain*
[b] *School of Electrical Engineering and Computing, The University of Newcastle, Callaghan, NSW 2308, Australia*
[c] *IOC-ETSEIB, Universidad Politécnica de Catalunya, 08028 Barcelona, Spain*

Abstract

The car sequencing problem is a well established problem that models the conflicts arising from scheduling cars into an assembly line. However, the existing approaches to this problem do not consider non-regular or out-of-catalog vehicles, which are commonly manufactured in assembly lines. In this paper, we propose a new problem definition that deals with non-regular vehicles. This novel model is called robust Car Sequencing Problem. We model this realistic optimization problem using scenarios defined by different production plans. The problem can be solved by measuring the impact of the plans' variability and by observing the violations of the problem constraints that appear when switching from one plan to another. In addition to our model formulation, we design and implement a set of constructive metaheuristics to tackle the traditional and the novel robust car sequencing problem. The selected metaheuristics are based on the greedy randomized adaptive search procedure, ant colony optimization, and variable neighborhood search. We have generated compatible instances from the main benchmark in the literature (CSPLib) and we have applied these metaheuristics for solving the new robust problem extension. We complement the experimental study by applying a post-hoc statistical analysis for detecting statistically relevant differences between the metaheuristics performance. Our results show that a memetic ant colony optimization with local search is the best method since it performs well for every problem instance regardless of the difficulty of the problem (i.e., constraints and instance size).

Keywords— Car sequencing problem, assembly lines, robust optimization, metaheuristics

[*]Corresponding author
Email addresses: imoya@ugr.es (Ignacio Moya), manuelchica@ugr.es (Manuel Chica), joaquin.bautista@upc.edu (Joaquín Bautista)

1. Introduction

The Car Sequencing Problem (CSP) was first introduced by Parello et al. [50], arising as a result of the car industry switching from mass standardization to mass customization. It is a feasibility problem that considers the issues that appear when sequencing cars with different options in a single assembly line. The CSP is modeled by transforming time, space or technical requirements into abstract binary options [39, 57]. Each of the vehicles assembled through the line needs an arbitrary set of these options. Then, a maximum load ratio is defined from the available options, representing the possible conflicts that appear in the line if many consecutive vehicles require the same option [61]. This ratio is represented using the parameters p_i/q_i, where p_i is the number of vehicles that may contain the option i in a sequence of length q_i without causing the line to stall. In a nutshell, it means that for an option with $p = 1$ and $q = 3$, only one vehicle in every subsequence of length three requires this option. Therefore, p_i/q_i represents the ratio constraints of the problem. Thus, the goal of the CSP is to find a full sequence of vehicles that does not overload any of the options for every subsequence.

However, this traditional approach does not consider non-regular or out-of-catalog vehicles, which are common in current assembly lines [3]. These special vehicles are distinguished from the regular or common ones by several features: for example, they are vehicles under demand and they are not considered by the estimations of the sales forecast department. They are typically oriented to public service organizations (e.g., ambulances, firefighter or police patrols, among others) and require uncommon components that imply additional operations and increase the time consumed in its processing. The components of special vehicles may be uncommon and depend on its type (for example ambulance, police patrol, or forest guards). Therefore, its inclusion in the daily production schedule will involve differences in component consumption, workload, or the tools required by the line. The number of special vehicles scheduled for daily production bounces between 10% and 20% of the total demand [3]. Moreover, partial production of each type of special vehicle may vary from one day to another, but the production of common vehicles remains regular and stable.

In this paper we propose a new model, based on the traditional CSP, that deals with non-regular vehicles and manages its variability. We refer to this novel approach as the robust CSP (r-CSP), since it is a new model that deals with the uncertainty created by the special vehicles and reduces its impact in the daily production scheduling by generating robust schedule solutions.

This is achieved by including different production plans for modeling those special vehicles based in factory production historical data.

After presenting our working hypothesis and a formal definition of our new model, we study its resolution similarly to other uncertainty problems in the field of Industrial Engineering. One example is the time and space assembly line balancing problem [4], where the impact of variability is measured by observing the conflicts that appear when switching from one production plan to another [14]. We propose to follow a similar approach for solving the r-CSP. This way, our strategy starts by solving the traditional feasibility problem using a selected production plan used as the reference. Then, the robustness of the resulting sequence is calculated by creating random permutations of the optimized sequence. Each permutation exchanges the special vehicles from the reference plan by the ones from the selected production plan, keeping the regular vehicles as they were. We compute the final robustness assessment by computing the additional violations of ratio constraints caused by the permutations on the sequence.

We solve the novel r-CSP taking as reference the current strategies applied to the CSP. In this regard, the traditional CSP is a heavily combinatorial problem that was proven to be NP-Complete [29] and it is frequently solved using metaheuristics. Metaheuristics are a family of approximate non-linear optimization methods that have proven powerful for solving hard and complex problems in science and engineering [58]. Different metaheuristics have been successfully applied to the CSP, including local search [33, 52], greedy randomized adaptive search procedures (GRASP) [6], genetic algorithms [62], variable neighbour-search (VNS) [53] and ant-colony optimization (ACO) [33, 34, 45]. From those, we have selected GRASP, VNS, and ACO for solving the r-CSP, since they all are constructive metaheuristics and usually the most effective for dealing with heavily constrained problems [13].

Our design includes add-hoc local search operators for dealing with the r-CSP sequences in combination with a greedy heuristic for guiding the constructive phase of the selected metaheuristics. Additionally, we include a memetic ACO (ACO-LS) that combines the global search strategy of ACO with local search refinement for the best solution of the cycle and the best solution of the metaheuristic run. We compare the performance of the selected metaheuristics when solving the traditional CSP and the novel r-CSP. Our experimentation considers the available instances from the main benchmark in the literature (CSPLib), which contains instances with different degrees

of complexity. Using the available instances from CSPLib, we have generated compatible ones for solving the r-CSP. Finally, we complement our study performing statistical post-hoc procedures for ensuring the statistical significance of our results.

Next Section 2 discusses existing bibliography regarding the CSP and explores related work in Industrial Engineering dealing with similar uncertainty issues. Then, the formalization of the model is presented in Section 3 and Section 4 describes the design of the metaheuristics. In Section 5, we present the setup and results of our experiments for both CSP and r-CSP. Finally, in Section 6, we discuss our conclusions and final remarks.

2. Background

2.1. The car sequencing problem

2.1.1. Traditional approaches

Since the initial definition by Parello et al. [50], several approaches have been proposed for solving the traditional CSP. These approaches are usually tested using CSPLib [30], the reference benchmark for the problem. These contributions can be mainly split between those treating it as a feasibility problem and those treating it as an optimization one [9]. The former group [10, 8, 19], treats the CSP as a constraint-satisfaction problem. We can note that one of the main drawbacks of these approaches is they are unable to handle infeasible instances, and this is the case of many instances from CSPLib. On the other hand, those approaches that tackle the CSP as an optimization problem try to minimize the number of constraint violations within a sequence, being able obtain a sequence with minimum constraint violation when there is no feasible sequence.

The latter approaches (i.e., those considering the CSP as an optimization problem) employ different strategies, such as constraint programing [7, 55], integer programing [34], branch & bound algorithms [24], or beam search [5, 32] to other non-exact methods such as metaheuristics [31, 33, 45, 62, 63]. The available results using CSPLib show both exact methods and metaheuristics achieve good results, but they also show that metaheuristics exhibit a more robust behavior when the size of the problem is high [32, 45, 62] (i.e., number of vehicles to be scheduled). Thus, our proposal is based on the use of metaheuristics, since they can handle infeasible problem instances having a high number of vehicles. This is an important issue when dealing with the r-CSP because the uncertainty of the problem can make infeasible the former feasible CSP instances.

From the different contributions that solve the CSP using metaheuristics, we can highlight

the methods based on local search, ACO and genetic algorithms. With respect to local search methods, the work of Puchta and Gottlieb [52] is specially relevant, as it introduced new local search operators that are commonly used in other contributions [6, 33, 45]. One of those new operators, the inversion operator (referred as *Lin2Opt*), became specially relevant and is established as an off-the-shelf operator. Another typical operator is the *Swap* operator, which also appears in the literature and sometimes only swapping adjacent vehicles [31]. However, these contributions focuses on exploring the landscape of the problem using a single local search operator. In contrast, our proposed local search procedure considers a sequential exploration of the neighbors using a regular Swap operator followed by an inversion operator. As it is shown in the following sections, this strategy allow us to achieve competitive results. In addition, local search methods usually include greedy heuristics for building the initial solutions previous to the local search optimization. We also found that the heuristic based on the dynamic sum of utilization rate, proposed by Gottlieb et al. [33], is the most employed heuristic in the literature [20, 45, 56, 62]. Thus, the constructive metaheuristics proposed for solving the r-CSP include this heuristic as well.

ACO is a metaheuristic commonly used by the CSP community [26, 33, 34, 45, 56, 59, 60]. From the available contributions, we acknowledged as most relevant for our manuscript the ones addressing the design of the pheromone trail matrix [26, 45]. Although ACO generally obtains good solutions, the design of the pheromone trail matrix is a critical decision that heavily influences the algorithm performance. In addition, the proposed ACO implementations can include local search methods for improving their results. To the best of our knowledge, the best results for the CSP using ACO are achieved by employing a custom three dimensional pheromone trail and local search refinement, developed by Morin et al [45]. This pheromone trail is designed for taking advantage of placing certain classes of vehicles at a specific distance from each other. Therefore, the pheromone trail considers a matrix of dimension $n \times n \times q_{max}$, where n is the number of regular CSP classes and q_{max} is the maximum q defined among the classes options. In this structure, trail value $\tau_{i,j,d}$ represents the appropriateness of placing vehicles of classes i and j at distance d. The work of Morin et al [45] also includes a local search refinement using the inversion operator. This refinement is applied to the solution found by the best ant of each ACO algorithm's cycle as well as to the global best solution.

In comparison with the previous metaheuristics, genetic algorithms are not commonly applied

to the CSP [31, 62, 63]. This could be related with the problems of traditional genetic operators to deal with the CSP requirements and the difficulty of defining specific operators for such a heavily constrained problem [62]. However, there are few exceptions that manage to handle the specific characteristics of this family of problems and obtain competitive results [63]. For example, Zinflou et al. [63] introduces a hybridization strategy for enhancing crossover strategies that integrates integer linear programming during the crossover process.

2.1.2. Extensions to the traditional CSP model

One extension to the CSP that received plenty of attention by the CSP community is the one proposed by RENAULT for the ROADEF'2005 challenge [57]. This extension includes paint batching constraints and splits the previous capacity constraints into different categories depending on its priority. Because these changes on the original model include new requirements instead of modifying the core of the problem, the contributions solving the ROADEF version of the CSP should be seriously considered for improving the performance when solving other problem variations.

Several contributions addressing this version of the problem introduced interesting variations to previous strategies [15, 20, 28, 40, 51, 53, 64], but we highlight the work of Estellon et al. [20] and Ribeiro et al. [53] as specially relevant to our study. We can note that both of these contributions are competitive for solving the traditional CSP, as they respectively ranked first and second during the challenge. The work of Estellon et al. [20] is based on local search methods. It introduced useful notions for computing fast evaluations in local search operations and we will incorporate their guidelines in our implementations, as we will show in Section 4. Finally, Ribeiro et al. [53] proposed a metaheuristic combining VNS with iterated local search. Due to its good performance, we included some of their designing concepts into our custom VNS design.

Another model derived from the traditional CSP is the extended car sequencing problem [6]. This version extends the traditional requirements to establish a minimum number of operations into subsequences. This way, instead of focusing in capacity overload, the extended CSP considers the under assignment of certain options that may "under-load" certain stations in the assembly line. The authors proposed a GRASP [6, 48] for solving this problem extension and, therefore, it is a relevant contribution for designing the solving methods of the problem proposed in this manuscript. In [48], the authors presents how considering a small α_{gr} value improves the results of GRASP, since it increases the effect of the underlying greedy heuristic.

2.2. Dealing with uncertainty in similar problems

Being able to handle uncertainty is a usual issue for assembly lines and other industrial problems, since they are susceptible to different variability sources [38]. Some authors model uncertainty assuming intervals or distributions for the operations scheduled in the different stations [35, 36, 38]. In contrast, other contributions tackle the inclusion of uncertain demand as a robust optimization problem [14]. Generic robust optimization addresses the sensitivity of the solutions to certain perturbations [17]. Thus, one solution is considered as a robust one if it is not sensitive to perturbations of the decision variables of its neighborhood. This approach considers two main robustness measures [23]: (1) the expectation measure, which replaces the original objective function by another function that combines performance with expectation in the vicinity; and (2) the variance measure, where an additional criterion is added to the original function for measuring the deviation around the original criteria. In this sense, solving the CSP with uncertain partial demand is a robust optimization problem, where this problem variability can change the optimal solution [23].

Several contributions use scenarios for representing uncertainty in their demand [2, 14, 49]. Using this approach, these scenarios can be represented as production plans that model the variation of demand for the different vehicles. In Chica et al. [11], one production plan is first chosen as the reference and it is later employed to search for the optimal solution. The robustness of the resulting solution when external conditions change is then computed by using the rest of the production plans defined in the set. Robust solutions are those that include minimum or zero deviation across the defined production plans [11]. Therefore, the work presented in this manuscript considers a reference production plan to search for the optimal sequence and a set of alternate production plans for evaluating its robustness.

3. r-CSP model definition

This section states our proposed model for the r-CSP. First of all, we present the main parameters in Section 3.1. Then, the constraints of the model are described in Section 3.2. Section 3.3 explains the sequence construction and Section 3.4 defines the problem's objective function.

3.1. Main parameters

The main variables and parameters of the problem are described in Tables 1 and 2. These variables and parameters construct the base of the model and their meaning are described by the following items:

I_X	Set of classes of regular vehicles, $\|I_X\| = n$. Index $(i = 1, ..., n)$.
$I_{X'}$	Set of classes of non-regular vehicles, $\|I_{X'}\| = n'$. Index $(i = n + 1, ..., n + n')$.
I	Set of vehicle classes: $I = I_X \cup I_{X'}$ and $I_X \cap I_{X'} = \emptyset$, $\|I\| = n + n'$.
J	Set of options $(J : j = 1, ..., \|J\|)$.
E	Set of scenarios or production plans $(E : \varepsilon = 1, .., \|E\|)$.
$\vec{d}_\varepsilon D$	Demand vector for plan $\varepsilon \in E : \vec{d}_\varepsilon = (d_{1,\varepsilon}, ..., d_{\|I\|,\varepsilon})$ and total vehicle demand for a working day: $D = \sum_{\forall i} d_{i,\varepsilon}$, equal for every plan $\varepsilon \in E$.
$\vec{\lambda}_\varepsilon$	Mix production vector for plan $\varepsilon \in E : \vec{\lambda}_\varepsilon = (\lambda_{1,\varepsilon}, ..., \lambda_{\|I\|,\varepsilon}) : \vec{\lambda}_\varepsilon = \vec{d}_\varepsilon / D$.
p_j/q_j	Traditional CSP ratios that represents the maximum requirement allowed for the option $j \in J$ by the vehicles contained in any segment of the sequences $\pi_\varepsilon(D)(\forall \varepsilon \in E)$ with length q_j, that should be less than or equal to the value p_j.
$c_{j,t,\varepsilon}$	Weight for segment of consecutive cycles $[t - q_j + 1, t]$ in sequence $\pi_\varepsilon(D)$ linked to plan $\varepsilon \in E$, when the requirement of option $j \in J$ is greater than p_j in that segment. We assume that all the weights satisfy: $c_{j,t,\varepsilon} = 1(\forall j \in J, \forall t \in [q_j, D], \forall \varepsilon \in E)$.
$\pi_\varepsilon(D)$	Complete sequence of vehicles $\pi_\varepsilon(D) = (\pi_{1,\varepsilon}, ..., \pi_{D,\varepsilon})$ from plan $\varepsilon \in E$. We denote partial sequences of $\pi_\varepsilon(D$ as: $\pi_\varepsilon(t) = (\pi_{1,\varepsilon}, ..., \pi_{t,\varepsilon}) \subseteq \pi_\varepsilon(D), \forall t \in [1, D]$. We will also employ symbols $\pi_\varepsilon(t)$ and $\pi_\varepsilon(D)$ as parameters.
$X_{i,t}$	Number of regular vehicles of class $i \in I_X$ contained in all the partial sequence $\pi_\varepsilon(t) \subseteq \pi_\varepsilon(D)$ of plan $\varepsilon \in E$. Its calculation is computed as: $X_{i,t} = \sum_{\tau=1}^{t} x_{i,\tau}, \forall i \in I_X, \forall t \in [i, D]$.
$X'_{i,t,\varepsilon}$	Number of special vehicles of type $i \in I_{X'}$ contained in partial sequence $\pi_\varepsilon(t) \subseteq \pi_\varepsilon(D)$ of plan $\varepsilon \in E$. It is computed like: $X'_{i,t,\varepsilon} = \sum_{\tau=1}^{t} x_{i,t,\varepsilon}, \forall i \in I_{X'}, \forall t \in [1, D], \forall \varepsilon \in E$.
$Y_{j,t,\varepsilon}$	Number of times that option $j \in J$ is required by regular and special vehicles contained in partial sequence $\pi_\varepsilon(t) \subseteq \pi_\varepsilon(D)$ of plan $\varepsilon \in E$. It is computed like: $Y_{j,t,\varepsilon} = \sum_{i \in I_X} n_{j,i} X_{i,t} + \sum_{i \in I_{X'}} n_{j,i} X'_{i,t,\varepsilon}, j \in J, \forall t \in [1, D], \forall \varepsilon \in E$. We also convene that: $Y_{j,0,\varepsilon} = 0, \forall j \in J, \forall \varepsilon \in E$.

Table 1: Description of the parameters and variables for modeling the r-CSP.

$n_{j,i} =$	$\begin{cases} 1 & \text{if the option } j \in J \text{ appears in the vehicle type } i \in I, \\ 0 & \text{otherwise.} \end{cases}$
$x_{i,t} =$	$\begin{cases} 1 & \text{if a regular vehicle } i \in I_X \text{ is assigned to position } t \ (t = 1, ..., D) \\ & \pi_\varepsilon(D) \text{ of plans } \varepsilon \in E, \\ 0 & \text{otherwise.} \end{cases}$
$x'_{i,t,\varepsilon} =$	$\begin{cases} 1 & \text{if a fleet vehicle } i \in I_{X'} \text{ is assigned to position } t \ (t = 1, ..., D) \\ & \text{of sequence } \pi_\varepsilon(D) \text{ of plan } \varepsilon \in E, \\ 0 & \text{otherwise.} \end{cases}$
$z_{j,t,\varepsilon} =$	$\begin{cases} 1 & \text{if the requirement of option } j \text{ is greater than the } p_j \text{ value in the} \\ & \text{segment } [t - q_j + 1, t] \text{ of the sequence } \pi_\varepsilon(D), \\ 0 & \text{otherwise.} \end{cases}$

Table 2: Description of the binary parameter $n_{j,i}$ and binary decision variables of the r-CSP model.

1. There are two separate vehicle families: the common or regular ones, represented as the set of classes I_X; and the special vehicles, represented by $I_{X'}$.
2. The total number of regular vehicles, D_X, should be accomplished during a working day. This value is the same for every production plan $\varepsilon \in E$.
3. The total number of non-regular vehicles, $D_{X'}$, should be accomplished during a working day. This value is the same for every production plan $\varepsilon \in E$.
4. The total number of vehicles, $D \equiv D_X + D_{X'}$, should be accomplished during a working day. This value is the same for every production plan $\varepsilon \in E$.
5. The demand for a regular vehicle $d_i, \forall i \in I_X$, should be the same for every production plan $\varepsilon \in E$. It means that $d_{i,\varepsilon} = d_i, \forall \varepsilon \in E$.
6. The demand for a non-regular vehicle $d_{i'}$, where $i' \in I_{X'}$, can change between two different plans $\{\varepsilon, \varepsilon'\} \subseteq E$.
7. Changes in the assembly line (e.g., robots, tools, or working staff) should be as minimum as possible. Then, production sequences $\pi_\varepsilon(D)$ and $\pi_{\varepsilon'}(D)$ should be as similar as possible for every two plans $\{\varepsilon, \varepsilon'\} \subseteq E$. This is: $\pi_\varepsilon(D) \approx \pi_{\varepsilon'}(D), \forall \{\varepsilon, \varepsilon'\} \subseteq E$.
8. Every regular vehicle will keep the same position in all the sequences $\pi_\varepsilon(D), \forall \varepsilon \in E$.

3.2. Constraints

This section presents the formal definition of the r-CSP using integer linear programing constraints. We define a set of constraints that any valid sequence should meet. The restriction shown

in Constraint 1 imposes that only one vehicle (regular or special) can be placed in the assembly line for every production cycle $t \in [1, D]$ and every production plan $\varepsilon \in E$:

$$\sum_{i \in I_X} x_{i,t} + \sum_{i \in I_{X'}} {x'}_{i,t,\varepsilon} = 1, \forall t \in [1, D], \forall \varepsilon \in E. \tag{1}$$

Constraints 2 and 3 force the satisfaction of the demand of regular (I_X) and special ($I_{X'}$) vehicles for every production plan:

$$\sum_{t \in [1,D]} x_{i,t} = d_i, \forall i \in I_X. \tag{2}$$

$$\sum_{t \in [1,D]} {x'}_{i,t,\varepsilon} = d_{i,\varepsilon}, \forall i \in I_{X'}, \forall \varepsilon \in E. \tag{3}$$

Constraint 4 counts the number of regular vehicles $i \in I_X$ placed in the assembly line until the production cycle $t \in [1, D]$ in any production plan $\varepsilon \in E$:

$$X_{i,t} - \sum_{\tau \in [1,t]} x_{i,\tau} = 0, \forall i \in I_X, \forall t \in [1, D]. \tag{4}$$

Constraint 5 is analogous to constraint (4) but referring to non-regular vehicles $I_{X'}$ and considering each production plan $\varepsilon \in E$:

$${X'}_{i,t,\varepsilon} - \sum_{\tau \in [1,t]} {x'}_{i,\tau,\varepsilon} = 0, \forall i \in I_{X\prime}, \forall t \in [1, D], \forall \varepsilon \in E. \tag{5}$$

Constraint 6 counts the number of times that the option $j \in J$ is required by the vehicles consecutively placed in the line until any production cycle $t \in [1, D]$ in any production plan $\varepsilon \in E$:

$$Y_{j,t,\varepsilon} - \sum_{i \in I_X} n_{j,i} X_{i,t} - \sum_{i \in I_{X'}} n_{j,i} {X'}_{i,t,\varepsilon} = 0, \forall j \in J, \forall t \in [1, D], \forall \varepsilon \in E. \tag{6}$$

Constraint 7 checks the requirements of every option $j \in J$ for every segment with length $q_j (\forall j \in J)$ from sequence $\pi_\varepsilon(D) (\forall \varepsilon \in E)$:

$$Y_{j,t,\varepsilon} - Y_{j,t-q_j,\varepsilon} \leq p_j + D \cdot z_{j,t,\varepsilon}, \forall j \in J, \forall t \in [q_j, D], \forall \varepsilon \in E. \quad (7)$$

Constraints 8, 9, and 10 respectively define the binary variables $x_{i,t}$, ${x'}_{i,t,\varepsilon}$, and $z_{j,t,\varepsilon}$:

$$x_{i,t} \in \{0,1\}, \forall i \in I_X, \forall t \in [1, D]. \quad (8)$$

$${x'}_{i,t,\varepsilon} \in \{0,1\}, \forall i \in I_{X'}, \forall t \in [1, D], \forall \varepsilon \in E. \quad (9)$$

$$z_{j,t,\varepsilon} \in \{0,1\}, \forall j \in J, \forall t \in [q_j, D], \forall \varepsilon \in E. \quad (10)$$

Finally, Constraint 11 sets the variable $Y_{j,0,\varepsilon} (\forall j \in J, \forall \varepsilon \in E)$ to 0 for the not existing production cycle $t = 0$:

$$Y_{j,0,\varepsilon} = 0, \forall j \in J, \forall \varepsilon \in E. \quad (11)$$

3.3. Sequence construction

Using this r-CSP model we can build multi-sequences $\vec{\pi}(E, D)$ composed by the sequences $\pi_\varepsilon(D) = (\pi_{1,\varepsilon}, ..., \pi_{D,\varepsilon})$ of each production plan $\varepsilon \in E$. This construction is shown in Equation 12. The regular vehicles $i \in I$ are linked to the elements $\pi_{t,\varepsilon} (\forall t \in [1, D], \forall \varepsilon \in E)$ of the multi-sequence $\vec{\pi}(E, D)$ by the binary variables $x_{i,t} (\forall i \in I_X, \forall t \in [1, D])$ and $x\prime_{i,t,\varepsilon} (\forall i \in I_{X\prime}, \forall t \in [1, D], \forall \varepsilon \in E)$, defined in Equations 13 and 14. We must remark that every regular vehicle ($i \in I_X$) keeps its position in the sequence for every production plan, but all the positions occupied by special vehicles depend on the production plan $\varepsilon \in E$. As a consequence, sequences $\pi_\varepsilon(D)$ have a common component (composed by regular vehicles) and an exclusive component (composed by special vehicles).

$$\vec{\pi}(E,D) = \left\{ \begin{array}{c} \pi_1(D) \\ \pi_2(D) \\ ... \\ \pi_\varepsilon(D) \\ ... \\ \pi_{|E|}(D) \end{array} \right\} = \left\{ \begin{array}{c} (\pi_{1,1},...,\pi_{t,1},...,\pi_{D,1}) \\ (\pi_{1,2},...,\pi_{t,2},...,\pi_{D,2}) \\ ... \\ (\pi_{1,\varepsilon},...,\pi_{t,\varepsilon},...,\pi_{D,\varepsilon}) \\ ... \\ (\pi_{1,|E|},...,\pi_{t,|E|},...,\pi_{D,|E|}) \end{array} \right\} \tag{12}$$

$$x_{i,t} = 1 \Rightarrow \pi_{t,\varepsilon} = i, \forall i \in I_X, \forall t \in [1,D], \forall \varepsilon \in E. \tag{13}$$

$$x\prime_{i,t,\varepsilon} = 1 \Rightarrow \pi_{t,\varepsilon} = i, \forall i \in I_{X\prime}, \forall t \in [1,D], \forall \varepsilon \in E. \tag{14}$$

3.4. Objective function

Using the above formulation, we introduce the objective function for our proposed r-CSP model which minimizes the number of violations of restrictions defined for options $j \in J$. To do this we use their maximum number of vehicles requiring the option $p_j(\forall j \in J)$ for each plan $\varepsilon \in E$ and for each interval of consecutive production cycles of length $q_j(\forall j \in J)$, represented by the binary variable $z_{j,t,\varepsilon}$. Equation 15 shows the objective function. This formulation treats the r-CSP as a maximum satisfiability problem (MAX-SAT), which is connected with the original formulation of Parello et al. [50].

$$min\ Z = \sum_{j \in J} \sum_{t \in [q_j, D]} \sum_{\varepsilon \in E} z_{j,t,\varepsilon} \Leftrightarrow max\ Z\prime = \sum_{j \in J} \sum_{t \in [q_j, D]} \sum_{\varepsilon \in E} (1 - z_{j,t,\varepsilon}). \tag{15}$$

4. Metaheuristics applied to the r-CSP and CSP

As previously introduced, several metaheuristics have been successful when solving the CSP [33, 45, 51, 52, 53]. Many of the metaheuristics applied to the CSP belong to the family of the constructive metaheuristics [58], which perform well with heavy constrained problems [1, 13, 12, 37]. Constructive metaheuristics build solutions from scratch, adding elements iteratively until a full solution is built. Because of its good performance, we chose to solve the r-CSP with constructive metaheuristics as well. Our design for the selected metaheuristics use a greedy heuristic during its constructive phase. This heuristic is based on the dynamic sum of utilization rate [33], which was

used in other approaches to the CSP [20, 45, 56, 62]. Using this heuristic, the available elements are ordered by favoring the most overloaded elements, so the vehicles requiring more restricted options would be selected first.

We will describe in Section 4.1 the robustness assessment for the r-CSP sequences and the fitness function employed by the algorithms. Then, Section 4.2 introduces the local search procedure used by the metaheuristics. Sections 4.3 and 4.4 present our GRASP and ACO design for the CSP and the r-CSP. Section 4.5 describes the VNS metaheuristic. Finally, Section 4.6 presents the design of the memetic algorithm (ACO-LS) used in our experiments.

4.1. Robustness assessment and fitness function

The selected metaheuristics use the same objective function for their fitness computations. In the case of the traditional CSP, the fitness values represent the number of violations of ratio constraints found in a sequence. This is a special case for the objective function defined in Equation 15 but only considering the reference production plan (i.e., $|E| = 1$). We use the latter as the base function and we name it $f_b(\pi_0(D))$.

Computing the fitness value for $|E| > 1$ implies to generate and check every combination of plans and fleet vehicles ($\forall \varepsilon \in E$), by calculating all the violations of ratio constraints. In order to check the permuted sequences, we choose an approximate solution instead of generating every possible sequence and checking its ratio constraints, since it would imply very long execution times. We simulate this value following an approach similar to the so called *simheuristics* [41]. This way, we approximate this value by randomly generating S possible sequences for each production plan using the original multi-sequence $\vec{\pi}(E, D)$ and by computing the average of the minimum violations obtained for each production plan $\varepsilon \in E$.

$$f_r(\vec{\pi}(E, D)) = \frac{\sum_{\varepsilon \in E} min \sum_{j \in J} \sum_{t \in [q_j, D]} z_{j,t,\varepsilon}}{|E|}. \tag{16}$$

Function $f_r(\vec{\pi}(E, D))$, defined by Equation 16, is an approximation for computing the values of the objective function of the r-CSP (shown in Equation 15 of Section 3). The final fitness function for solving the r-CSP combines the basic number of violations (using only the reference production plan) with the robustness metric using a parameter $\theta \in [0, 1]$. The θ parameter sets the importance of the reference plan with respect to the rest of production plans. The final objective function defined for the r-CSP, $f(\vec{\pi}(E, D))$, is defined in Equation 17.

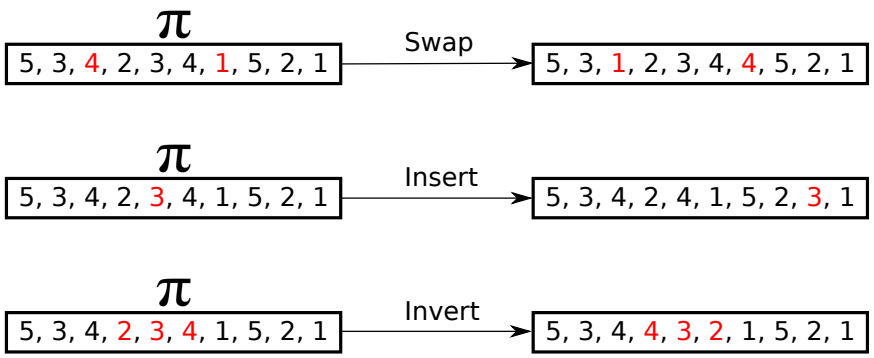

Figure 1: Scheme illustrating the application of the defined operators to a baseline sequence π.

$$f(\vec{\pi}(E,D)) = \theta \cdot f_b(\pi_0(D)) + (1-\theta) \cdot f_r(\vec{\pi}(E,D)). \tag{17}$$

4.2. Local search

Our metaheuristics design considers a local search procedure, which is used by every metaheuristic using the same setup. This procedure is an important component of GRASP and VNS and was also used in conjunction to the ACO algorithm to create a memetic algorithm for solving the problem (ACO-LS). Local search procedures [54] iteratively refine the solutions by altering its values in order to maximize their quality (fitness function). The local search operators modify a given solution using neighborhood relations defined on the search space. We selected our operators based on previous work [33, 52] and the results of our experiments. These operators are three: *Swap*, *Insert*, and *Invert*.

Figure 1 shows how the defined operators modify a baseline sequence π. Swap operator exchanges two positions in the sequence that are selected randomly. We can see in Figure 1 how the Swap operator exchanges positions 3 and 7. Insert operator moves one car from its current position to another, with both positions selected at random. In Figure 1, the Insert operator moves the car from position 5 to position 9. Finally, the Invert operator inverts a subsequence between two positions selected at random. Figure 1 shows how the Invert operator inverts the subsequence between positions 4 and 6.

Our local search procedure follows a stochastic hill climbing strategy [54]. At each step, a random neighbor of the current solution is generated and evaluated. The move is accepted if the new current solution improves the previous solution. Otherwise, the move is reverted. Using the latter operators, we define a strategy where the local search explores the solution space using the

Swap operator until the search stops. After this process finishes, the Invert local search operator is applied sequentially on the resulting sequence, only in the case a feasible solution was not found in the first attempt. Finally, the *Insert* operator is only used by one of the implemented metaheuristics (VNS, described in Section 4.5).

4.3. GRASP

The greedy randomized adaptive search procedure (GRASP) [21, 22] is a multi-start constructive metaheuristic that finds solutions combining randomness and a greedy heuristic. This process is carried out through a given number of iterations, defined by the user. Each iteration is developed in two steps; a semi-greedy constructive phase, followed by an improvement phase based on neighborhood search. This metaheuristic uses heuristic with a diversity mechanism to explore different solutions. Specifically, the diversity of the algorithm is achieved due to a random component that intervenes in the constructive phase: new elements are added to the partial solutions by randomly selecting them from a restricted candidate list. This candidate list is filled and ordered by defining a parameter α that acts as a regulator of the randomness and a heuristic function.

Our specific design for the CSP and r-CSP considers three steps. First, a car class ($i \in I$) is included in the list if it satisfies one of the following conditions: its inclusion would not increase the actual number of violations of the current subsequence or the increment would be the minimum possible from every available car class. Then, if the candidate list contains more than one element (typically in the first steps it has multiple elements), those candidates are ordered using a heuristic value. Finally, the candidate list is truncated to a smaller size depending on the value of the $\alpha_{gr} \in (0, 1]$ parameter, which regulates the size of the candidate list, leaving only the elements with better heuristic values. For example, $\alpha_{gr} = 0.6$ will define a list size to 60% of the number of different filtered elements. After the constructive phase, an improvement phase is applied for refining the quality of the solution found. This improvement phase makes use of the local search procedure, detailed in section 4.2.

4.4. ACO

Ant colony optimization (ACO) [18] is a bioinspired metaheuristic developed for solving heavy constrained combinatorial problems. ACO mimics the behavior of ants, which communicate through the use of a pheromone trail accessed by every ant in the colony. The ants can use this trail as a memory device that guides them while building solutions. The trail itself is modified

depending on the quality of the final solution developed by the ants. At each step in the ant's path, the artificial ant chooses the next element to be added to the sequence by compromising between the memory stored in the pheromones trail and a greedy heuristic. The greedy heuristic is chosen with probability q_0. Otherwise, the information in the pheromones trail is used and the next vehicle is chosen using the parameters α_{aco}, β, δ, which regulates the weight of the pheromone trail, the vehicle adding fewer new conflicts, and the heuristic value, respectively. The ACO metaheuristic runs a given number of ants (i.e., solutions to the problem) for a defined number of cycles. For each cycle, the best ant of the cycle (the solution with highest quality) modifies the pheromone trail, influencing the behavior of the following ants.

Our ACO design follows the description of Morin et al. [45], which is the ACO version obtaining the best results in the traditional CSP. This ACO approach [26, 45] includes a novel three dimensional pheromone matrix of size $n \times n \times q_{max}$. This way, $\tau_{i,j,d}$ represents the appropriateness of placing vehicles of class i and j separated by d positions. The pheromone trail is initiated to τ_0 and the parameters ρ_l and ρ_g manage its update process. On the one hand, ρ_l manages the update of the pheromone trail after an ant chooses a vehicle, slightly reducing its pheromones trail values to help the diversification of the algorithm. On the other hand, ρ_g manages the persistence of the pheromone trail during the evaporation process and its value is updated taking into account the best ant of the cycle.

4.5. VNS

The variable neighborhood search (VNS) [44] is a metaheuristic that introduces perturbations by a systematic change of neighborhood. Perturbations take the current solution and iteratively apply k neighbor changes. After each perturbation, the search continues using a local search procedure. This process is repeated through different iterations until $k = k_{max}$ or until the current solution meets a given acceptance criteria (in our problem, finding a sequence with better solution fitness). If the current sequence is accepted by the acceptance criteria the value of k is restarted to 1. The search continues until the algorithm meets a global stopping criteria (i.e., when a feasible solution is found or when the algorithm reaches the maximum number of evaluations).

The algorithm generates its initial solution using a greedy strategy, which is based on always making the decision with the best heuristic value at each step of the solution's construction. Then, the systematic change of neighborhood is achieved by applying k times the *Insert* operator. For

example, when $k = 3$, three consecutive insertions are applied. We can note that each application of the *Insert* operator changes the neighborhood, and thus, changes the landscape [31]. Since the refining local search procedure is based in *Swap* and *Inversion* operators, it would require multiple applications to return to the previous visited local optima. The latter fact makes the introduction of additional control mechanisms unnecessary [43]. This operator was depicted in detail in Section 4.2.

4.6. Memetic algorithms

Memetic algorithms [46, 47] complement a global search method by including a local search procedure that improves the quality of the solutions found during the global search of the algorithm. Typically, memetic algorithms apply this local search refinement to every solution found during the algorithm's run, but this can be time-consuming and it was proven that this design decision does not always lead to the best performing memetic algorithm [42]. We have included in our experiments a memetic ACO (ACO-LS) that modifies the basic ACO design and adds a local search refinement for both the best solution of the cycle and the best solution found. The maximum number of local search steps devoted to improve the best ant of each cycle is managed by parameter $steps_l$. Parameter $steps_g$ manages the maximum number of local search steps for the best solution found. In contrast with [45], we selected different neighbor operators based on our experimentation (i.e., a combination of *Swap* and *Invert* operators). We present the pseudo-code of the proposed memetic algorithm in Algorithm 1.

5. Experimentation

5.1. Experimental setup

We have tested the performance of our selected metaheuristics using the instances from CSPLib [30], since it is a well establish benchmark and it is generally used by the CSP community [33, 45, 62]. This benchmark has instances with different degrees of complexity that can be split in three categories: the feasible ones, the classic ones, and the advanced ones. The feasible ones (also referred as SET1 in our experiments) are compounded by 70 instances containing 200 vehicles with 5 options and between 17 and 30 vehicle classes. These instances are the most basic ones because they are designed to be feasible. Instances belonging to SET1 are divided in subsets of 10 instances that share the same utilization rate for their vehicle options. Each subset has a different utilization rate that increases from 60% to 90%. The classic instances are gathered in the SET2 [30]. This set

Algorithm 1: Pseudo-code for the proposed memetic algorithm (ACO-LS).

```
1  begin
2      Set every position of the pheromone trail to τ0;
3      cycle = 0;
4      while cycle < maxcycles and no feasible sequence found do
5          for each ant do
6              Initialize first car at random;
7              for position = 2 to D do
8                  if new random value≤ q0 then
9                      Add a car from the class with the best heuristic value;
10                 else
11                     Add a car using the pheromone trail;
12                 Update trail with selected class;
13         Apply local search to best solution of the cycle;
14         Update pheromone trail using the best solution of the cycle;
15         Update best solution so far;
16         cycle = cycle + 1;
17     if best solution is not feasible then
18         Apply local search to best solution;
```

contains 9 instances with 100 vehicles with 5 options and between 18 and 24 classes. SET2 is also harder than SET1 because only 4 of its instances are feasible. Finally, SET3 contains the advanced instances. These instances were first proposed in [34] and contains the hardest sequences. These instances are 30 sequences, split in subsets of 200, 300, and 400 vehicles that have similar options and classes as the ones presented in the SET2.

To the best of our knowledge, there is not any benchmark for sequencing problems that consider special vehicles or alternate production plans. Therefore, we propose to generate those instances starting from the instances in CSPLib. In order to improve the compatibility of the new instances with the traditional ones, we have added additional information to the original instances as required by the r-CSP: the number of classes of special vehicles ($|I_{X'}|$), the number of alternate production plans ($|E|$), and the demand of the new plans ($\vec{d}_e D$). This way, for every traditional CSP instance, 20% of its classes are considered as non-regular ones. The original demand of the classes switching to non-regular is used as the demand of the special vehicles in the reference production plan. Specifically, the new defined non-regular classes will be the last classes defined in a traditional CSP instance. For example, if a sequence with 10 classes is translated to consider special vehicles,

GRASP		**ACO**				**ACO-LS**		**VNS**	
Name	Value	Name	Value	Name	Value	Name	Value	Name	Value
$iterations$	400	$ants$	15	δ	3	$steps_l$	2,000	k_{max}	400
$steps$	10,000	$cycles$	266,666	q_0	0.9	$steps_g$	2,000,000	$steps$	10,000
α_{gr}	0.15	α_{aco}	4	τ_0	0.005	$cycles$	992		
		β	6	ρ_l	0.99				
				ρ_g	0.99				

Table 3: Name and value for every parameter of the selected metaheuristics.

classes from 1 to 8 will remain unaltered and classes 9 and 10 will be considered as special vehicles. As a consequence, the total amount of vehicles belonging to the special vehicles demand depends on the original demand of classes 9 and 10. Finally, 10 completely random production plans are added for those non-regular classes.

The parameter configuration of the metaheuristics is shown in Table 3. The parameters of GRASP and VNS were selected in a preliminary experimentation. ACO parameters were taken from the original contribution of [45]. In the case of ACO-LS, only the number of cycles and the parameters for the local search are specified because ACO-LS shares the rest of parameters with ACO. However, we reduce the number of cycles for ACO-LS because of the number of evaluations consumed by the local search procedure. During our experimentation, every metaheuristic runs until either finding a feasible solution or reaching a maximum number of 4,000,000 evaluations of full-length sequences. Every metaheuristic is run 30 times using different seeds. Parameters θ and S for the fitness function are set to $\theta = 0.6$ and $S = 30$. All the metaheuristics were implemented in Java, and we used the JAMES framework [16] for implementing the local search procedures and the VNS metaheuristic. In addition, we have published our source code and the generated r-CSP benchmark of instances [1].

5.2. Algorithmic comparison for the traditional CSP

In this section we compare the performance of the metaheuristics considered in our study solving traditional CSP instances from CSPLib. These results are provided as number of violations of load constraints, which correspond to the fitness values. Table 4 shows the fitness results of the algorithms when applied to SET1. Because the instances of this set are easy to solve, we show a summary of the averaged fitness for every algorithm execution and instance. These results

[1] https://bitbucket.org/imoya/jcsp/

Instances	**GRASP**	**ACO**	**ACO-LS**	**VNS**
60-*	0	0	0	0
65-*	0	0	0	0
70-*	0	0	0	0
75-*	0.01	0	0	0
80-*	0	0	0	0
85-*	0	0	0	0
90-*	0.87	0	0	0

Table 4: Average fitness values for every metaheuristic solving the traditional CSP using the instances contained in SET1.

show that the selected algorithms are capable of solving the easy instances in almost every run. Nevertheless, GRASP reduces its effectiveness when solving harder instances (the ones with a higher utilization rate).

Table 5 has the results when solving the classic and advances instances, gathered in SET2 and SET3. These values represent the average fitness and the best solution found among the different runs of the metaheuristics for the different instances. We include the best solution found in the literature for each instance in order to compare our results with the state of the art [32, 45]. In addition, the ranking of the metaheuristics resulting from these values is shown in Table 6. The ranking of the metaheuristics is computed using the average result of each metaheuristic for each instance. Then, the mean rank is computed for each set and subset of SET2 and SET3.

The results for SET2 and SET3 suggest that ACO-LS outperforms the other metaheuristics, since most of the highlighted values of Table 5 belong to ACO-LS. This is corroborated by the average ranking values shown in Table 6, where ACO-LS achieves values close to 1. ACO is the second best metaheuristics in these instances, achieving rank values close to 2 and even 1.6 for the the 400-* subset. VNS shows good performance, achieving the best value for several instances and ranking close to ACO. However, its performance decreases for the largest instances (subsets 300-* and 400-*). These results also confirm that GRASP's performance decreases when increasing the difficulty of the instances, since it is ranked as the last one in almost every instance.

With respect to the state of the art, ACO-LS has the best result for almost every instance, with ACO reaching many of them. VNS also reaches some of the best solutions, specially from SET2 and subset 200-*. The results shown in Table 5 also show an improvement of the best values known for some instances. Specifically, during our experiments we have found new overall best solutions for *300_05* and *300_10* instances.

	Instances	Best found	GRASP		ACO		ACO-LS		VNS	
			Best	Avg (σ)	Best	Avg (σ)	Best	Avg (σ)	Best	Avg (σ)
SET 2	p10_93	3	9	10.9 (0.8)	4	4.93 (0.64)	**3**	**4.03 (0.72)**	4	4.77 (0.73)
	p16_81	0	6	6.57 (0.5)	**0**	0.9 (0.66)	**0**	**0.37 (0.49)**	**0**	1.73 (0.69)
	p19_71	2	4	5.37 (0.72)	**2**	2.27 (0.45)	**2**	**2 (0)**	**2**	2.03 (0.18)
	p21_90	2	3	3.87 (0.63)	**2**	2.27 (0.45)	**2**	**2 (0)**	**2**	2.1 (0.31)
	p26_82	0	3	4.13 (0.51)	**0**	0.03 (0.18)	**0**	**0 (0)**	**0**	0.93 (0.52)
	p36_92	2	4	4.7 (0.53)	**2**	2.2 (0.41)	**2**	**2 (0)**	**2**	2.07 (0.25)
	p41_66	0	**0**	0.7 (0.47)	**0**	**0 (0)**	**0**	**0 (0)**	**0**	0.23 (0.43)
	p4_72	0	4	5.5 (0.68)	**0**	0.03 (0.18)	**0**	**0 (0)**	**0**	0.87 (0.57)
	p6_76	6	**6**	**6 (0)**	**6**	**6 (0)**	**6**	**6 (0)**	**6**	**6 (0)**
SET 3	200_01	0	6	7.9 (0.84)	**0**	1.5 (0.68)	**0**	**0.87 (0.57)**	**0**	1.6 (1.04)
	200_02	2	8	10.27 (0.94)	**2**	3.2 (0.61)	**2**	2.77 (0.43)	**2**	**2.53 (0.57)**
	200_03	3	17	19.37 (1.07)	7	7.4 (0.62)	**5**	**6.87 (0.51)**	10	12.63 (1.13)
	200_04	7	16	17.67 (0.76)	8	8.1 (0.31)	**7**	**7.27 (0.45)**	8	10.23 (0.94)
	200_05	6	8	9.07 (0.58)	7	7.47 (0.51)	**6**	6.53 (0.51)	**6**	**6.5 (0.51)**
	200_06	6	9	10.4 (0.67)	**6**	6.07 (0.25)	**6**	**6 (0)**	**6**	6.17 (0.38)
	200_07	0	6	8 (1.08)	**0**	0.17 (0.38)	**0**	**0 (0)**	**0**	**0 (0)**
	200_08	8	10	11.97 (0.61)	**8**	**8 (0)**	**8**	**8 (0)**	**8**	8.43 (0.5)
	200_09	10	12	14.43 (0.82)	11	11.2 (0.41)	**10**	**10.1 (0.31)**	**10**	**10.07 (0.25)**
	200_10	19	21	22.1 (0.61)	20	20.67 (0.55)	**19**	**19.73 (0.64)**	**19**	20.07 (0.52)
	300_01	0	14	17.27 (1.28)	2	3.17 (0.46)	**0**	**2.3 (0.7)**	1	3 (1.31)
	300_02	12	28	31.67 (1.24)	**12**	12.87 (0.35)	**12**	**12.13 (0.35)**	14	16.8 (1.21)
	300_03	13	21	23.77 (1.17)	**13**	13.63 (0.49)	**13**	**13 (0)**	20	21.87 (1.11)
	300_04	7	22	24.17 (1.09)	**7**	8.47 (0.57)	**7**	**7.43 (0.5)**	11	13.53 (1.04)
	300_05	27	40	44.5 (1.55)	27	29.47 (1.25)	**25**	**27.03 (1.25)**	39	42.37 (1.73)
	300_06	2	17	19.7 (1.39)	**4**	4.27 (0.45)	**4**	**4.13 (0.35)**	11	15.07 (1.41)
	300_07	0	19	22.6 (1.38)	**0**	0.23 (0.43)	**0**	**0.07 (0.25)**	2	6.4 (2.42)
	300_08	8	19	22.1 (1.35)	**8**	8.57 (0.5)	**8**	**8 (0)**	**8**	10.23 (0.97)
	300_09	7	21	23.37 (1.35)	**7**	8.23 (0.5)	**7**	**7.57 (0.5)**	13	16.07 (1.48)
	300_10	20	29	32.83 (1.23)	19	20.57 (0.68)	**16**	**18.23 (1.17)**	24	25.87 (0.97)
	400_01	1	14	18.77 (1.59)	**1**	1.27 (0.45)	**1**	**1.1 (0.31)**	5	10.6 (1.67)
	400_02	15	38	42.5 (1.7)	16	17.33 (0.88)	**15**	**16.2 (0.66)**	31	34.8 (1.73)
	400_03	6	24	25.17 (0.87)	**6**	**6 (0)**	**6**	**6 (0)**	13	15.73 (1.31)
	400_04	19	35	37.27 (1.01)	20	20.03 (0.18)	**19**	**19.1 (0.31)**	27	29.7 (0.95)
	400_05	0	14	17.53 (1.43)	**0**	**0 (0)**	**0**	**0 (0)**	**0**	3.47 (1.78)
	400_06	0	14	19.2 (1.63)	**0**	**0 (0)**	**0**	**0 (0)**	**0**	2.23 (1.25)
	400_07	4	23	25.63 (1.27)	**4**	5.17 (0.53)	**4**	**4.53 (0.57)**	12	16.83 (2.36)
	400_08	4	20	22.6 (1.43)	**4**	5.53 (0.57)	**4**	**5.03 (0.61)**	14	17.53 (1.93)
	400_09	5	35	39.1 (1.52)	11	13.2 (1)	**7**	**9.93 (1.48)**	16	22.43 (2.25)
	400_10	0	13	16.03 (1.61)	**0**	**0 (0)**	**0**	**0 (0)**	**0**	2.03 (1.1)

Table 5: Minimum and average fitness values for every metaheuristic solving the traditional CSP using the instances contained in SET2 and SET3. The overall best values known for every instance are also included for comparison purposes.

		GRASP	ACO	ACO-LS	VNS
SET 2	Mean	3.67	2.22	1	2.33
SET 3	Mean	4	2.03	1.1	2.67
	200-*	4	2.4	1.3	2.1
	300-*	4	2.1	1	2.9
	400-*	4	1.6	1	3

Table 6: Mean rank for every metaheuristic solving the traditional CSP using the instances from SET2 and SET3. In the case of SET3, we also include the mean rank for the different subsets.

5.3. Algorithmic comparison for the r-CSP

This section discusses the performance of the metaheuristics solving the r-CSP. Tables 7 and 8 show the results using the instances contained in SET1. Table 7 shows the subsets from 60-* to 75-* and Table 8 shows subsets from 80-* to 90-*. In addition, Table 9 contains the mean ranking of the metaheuristics for SET1 and for every subset. The results for these instances show good performance of both GRASP and ACO-LS, which achieve the best results in most of the instances. On the one hand, GRASP achieves 60% of minimum values and 70% of the best average values. On the other hand, ACO-LS achieves 40% of minimum values and 30% of the best average values.

We can see that the good performance of GRASP in this set is related to the low utilization rate of its instances. This increases the number of feasible sequences for the reference plan and favors metaheuristics with slightly wider diversity. In this regard, the greedy heuristic included for the constructive phase could help both the ACO and VNS to reach a local optimum where the reference plan is feasible. This can also explain why GRASP slowly reduces its performance in favor of ACO and ACO-LS for the subsets contained in Table 8, where the utilization rate is higher and it is harder to find a feasible sequence for the reference plan. For the instances shown in Table 8, ACO-LS achieves 73.33% of minimum values and 66.67% of best average values, while GRASP only achieves 16.67% and 30% respectively.

The performance drop of GRASP when solving the harder instances of SET1 can also be observed in the ranking results of Table 9. These values show how GRASP's rank steadily increases from subset 75-* (mean ranking 1.4) to subset 90-* (mean ranking 3). In contrast, the mean ranking of ACO and ACO-LS for these subsets is reduced until reaching 2.1 and 1.1, respectively. The VNS performance is also reduced for the harder instances, but its ranking position just makes from 3.4 to 3.8.

The results for the instances in SET2 and SET3 are shown in Table 10. These results suggest

Instances	**GRASP**		**ACO**		**ACO-LS**		**VNS**	
	Best	Avg (σ)	Best	Avg (σ)	Best	Avg (σ)	Best	Avg (σ)
60-01	**3.08**	3.63 (0.22)	3.8	5 (0.52)	3.2	**3.51 (0.14)**	4.64	5.97 (0.52)
60-02	59.48	**60.5 (0.4)**	**59.08**	60.8 (0.89)	**59.08**	60.7 (0.74)	62.92	63.73 (0.47)
60-03	**4.32**	**4.66 (0.17)**	6.96	8.3 (0.53)	5.32	5.73 (0.24)	6	6.82 (0.44)
60-04	30.56	31.23 (0.35)	**28.68**	30.4 (0.87)	**28.68**	**30.35 (0.8)**	31.28	32.87 (0.85)
60-05	**1.48**	**2.4 (0.31)**	3.16	4.7 (0.94)	2.8	3.68 (0.32)	3.44	4.62 (0.71)
60-06	**2.24**	**3.01 (0.21)**	3.56	5.74 (1.32)	3.52	4.69 (0.65)	2.68	3.56 (0.4)
60-07	1.72	2.18 (0.24)	2.88	5.67 (0.7)	**0.6**	**1.14** (0.19)	2.92	3.74 (0.37)
60-08	**3.52**	**4.22 (0.23)**	8.56	9.92 (0.58)	5.04	6.13 (0.28)	6.56	7.71 (0.53)
60-09	**2.96**	**3.44 (0.2)**	3.04	4.17 (0.56)	**2.96**	3.61 (0.31)	3.88	5.03 (0.5)
60-10	**2.92**	**3.71 (0.39)**	5.84	9 (1.8)	3.16	3.92 (0.32)	9.24	10.81 (0.58)
65-01	6.28	7.04 (0.31)	5.76	7.2 (0.83)	**5.72**	**6.79 (0.51)**	7.24	7.82 (0.34)
65-02	59.96	**60.72 (0.38)**	**59.52**	61.31 (0.73)	**59.52**	61.31 (0.73)	62.68	62.68 (0)
65-03	**5.6**	**6.24 (0.25)**	8.6	10.86 (1.24)	6.08	6.82 (0.31)	9.68	11.02 (0.5)
65-04	26.6	27.57 (0.44)	**24.72**	26.5 (0.81)	**24.72**	**26.47 (0.8)**	27.32	29.1 (0.75)
65-05	**4.96**	**5.53 (0.28)**	7.12	9.93 (1.38)	6.2	6.77 (0.3)	5.52	7.42 (0.76)
65-06	**5.2**	**5.99 (0.24)**	6.16	8.85 (1.58)	6.16	8.1 (0.98)	8.76	10.13 (0.6)
65-07	5.08	5.68 (0.3)	6.04	8.53 (1.07)	**3.8**	**4.88 (0.42)**	7.6	9.05 (0.52)
65-08	**5.24**	**5.99 (0.24)**	9	11.61 (0.87)	7.76	8.45 (0.31)	8.04	9.07 (0.57)
65-09	**2.32**	**2.75 (0.21)**	3.68	4.93 (0.84)	2.84	3.36 (0.29)	3.6	4.59 (0.52)
65-10	**4**	**4.99 (0.39)**	7.32	9.45 (1.46)	4.72	5.63 (0.35)	7.16	9.43 (1.07)
70-01	7.24	8.56 (0.38)	**6.68**	8.39 (0.66)	**6.68**	**7.97 (0.48)**	11.2	11.91 (0.4)
70-02	58.56	60.03 (0.52)	**57.12**	**58.45 (0.77)**	**57.12**	**58.45 (0.77)**	61.04	62.07 (0.43)
70-03	10.88	**11.53 (0.25)**	11.48	15.39 (1.44)	**10.4**	11.75 (0.38)	12.76	13.6 (0.57)
70-04	10.48	11.33 (0.43)	**8.28**	9.74 (0.82)	**8.28**	**9.63 (0.79)**	11.04	11.94 (0.53)
70-05	**12.28**	**12.81 (0.28)**	14.2	17.68 (1.51)	13.36	14.16 (0.4)	14.88	15.96 (0.6)
70-06	**3.28**	**3.94 (0.29)**	4.32	6.13 (1.02)	3.72	5.2 (0.7)	6.56	7.38 (0.34)
70-07	8.92	9.86 (0.46)	10.08	12.02 (1.09)	**8.16**	**9.1 (0.45)**	12.12	13.88 (0.86)
70-08	**8.32**	**8.99 (0.35)**	11.12	14.81 (1.15)	10.48	11.76 (0.46)	9.44	10.46 (0.53)
70-09	**6.48**	**7.52 (0.4)**	7.24	10.52 (1.63)	7.68	9.11 (0.55)	7.36	10.01 (1.3)
70-10	**7.84**	**8.82 (0.47)**	10.6	13.38 (1.26)	8.6	9.25 (0.34)	11.16	13.88 (1.25)
75-01	10.8	11.28 (0.22)	**8.4**	10.04 (0.69)	**8.4**	**9.87 (0.71)**	11.84	13.22 (0.44)
75-02	**106.32**	**108.29 (0.81)**	107.48	109.85 (1.34)	107.48	109.79 (1.2)	109.92	113.16 (1.35)
75-03	**15.88**	**16.42 (0.24)**	17.64	19.61 (1.21)	17.2	18.02 (0.38)	17.4	18.74 (0.69)
75-04	17.64	19.02 (0.49)	**13.88**	15.53 (0.9)	**13.88**	**15.3 (0.72)**	18.6	20.34 (0.98)
75-05	**8.92**	**9.83 (0.29)**	9.52	13.93 (1.47)	9.52	12.48 (0.87)	10.8	11.92 (0.59)
75-06	**9.52**	**10.08 (0.34)**	10.36	13.61 (1.67)	10.36	13.15 (1.28)	12.4	13.84 (0.77)
75-07	**14.04**	**14.86 (0.38)**	15.16	17.31 (0.92)	14.32	15.87 (0.69)	14.96	16.77 (0.9)
75-08	**17.96**	**19.12 (0.37)**	25.12	27.81 (0.98)	23.04	24.53 (0.51)	20.52	21.65 (0.8)
75-09	6.64	**7.95 (0.45)**	7.44	10.44 (1.72)	**5.28**	9.56 (1.53)	9.12	10.52 (0.73)
75-10	11.72	**12.97 (0.46)**	**11.08**	16.37 (2.1)	11.48	14.22 (1.09)	16.52	18.39 (0.78)

Table 7: Minimum and average fitness values for every metaheuristic solving the r-CSP using the instances from SET1 (subsets from 60-* to 75-*).

Instances	GRASP		ACO		ACO-LS		VNS	
	Best	Avg (σ)	Best	Avg (σ)	Best	Avg (σ)	Best	Avg (σ)
80-01	6.4	7.11 (0.31)	**3.92**	6.39 (0.77)	4.92	**6.18 (0.57)**	7.8	8.41 (0.29)
80-02	**110.92**	**112.89 (0.89)**	112.56	115.11 (1.61)	112.56	114.95 (1.39)	113.12	115.61 (1.31)
80-03	13.24	13.82 (0.22)	**11.68**	12.62 (0.61)	**11.68**	**12.58 (0.63)**	15.64	16.67 (0.45)
80-04	34.72	35.68 (0.49)	**32.2**	33.77 (0.77)	32.36	**33.71 (0.74)**	34.88	36.65 (0.69)
80-05	19.16	**20.09 (0.46)**	**19.04**	21.03 (1.12)	**19.04**	20.86 (0.97)	23.28	24.79 (0.75)
80-06	**9.44**	**10.4 (0.34)**	12.8	15.5 (1.2)	12.64	15.14 (1.35)	12.76	14.29 (0.71)
80-07	15.4	**16.36 (0.34)**	14.96	17.02 (1.12)	**13.76**	16.96 (1.22)	19.2	20.86 (0.66)
80-08	**13.6**	**14.68 (0.38)**	17.2	19.71 (1.2)	17.36	19.55 (1.15)	14.48	15.82 (0.8)
80-09	86.4	87.62 (0.65)	82.56	86.45 (1.65)	**80.04**	**86.23 (1.96)**	89.4	92.24 (1.03)
80-10	**12.28**	**13.79 (0.4)**	13.56	14.75 (0.61)	**12.28**	14.39 (0.87)	15.64	16.53 (0.55)
85-01	6.2	7.02 (0.35)	**5**	6.35 (0.75)	5.04	**6.06 (0.49)**	7.76	8.63 (0.38)
85-02	29.76	**30.53 (0.38)**	29.4	32.6 (1.21)	**29**	30.56 (0.95)	32.56	34.16 (0.89)
85-03	36.76	37.64 (0.46)	41.2	43.05 (1.08)	**36.2**	**37.01 (0.51)**	38.4	41.2 (1.18)
85-04	23.48	**24.02 (0.36)**	22.8	24.77 (1.12)	**21.12**	24.29 (1.47)	26.12	27.24 (0.85)
85-05	**50.2**	**51.2 (0.57)**	50.92	52.11 (0.73)	50.92	52.06 (0.72)	52.08	53.26 (0.7)
85-06	61.84	62.65 (0.4)	61.64	63.66 (1.18)	**60.96**	**62.62 (1.05)**	61.16	62.94 (0.93)
85-07	11.6	12.15 (0.25)	9	10.86 (0.9)	**9**	**10.74 (0.83)**	13.8	14.51 (0.42)
85-08	19.72	20.56 (0.41)	17.08	19.19 (1.12)	**16.36**	**18.88 (1.29)**	21	22 (0.52)
85-09	55.04	56.34 (0.56)	53.36	56.08 (1.34)	**52.88**	**55.68 (1.22)**	55	56.93 (0.97)
85-10	22.2	23.73 (0.58)	**16.6**	19.12 (1.66)	**16.6**	**19.04 (1.57)**	25.72	27.85 (1.14)
90-01	12.76	14.27 (0.53)	11.2	12.66 (0.87)	**10.24**	**12.26 (0.86)**	13.6	14.35 (0.54)
90-02	26.28	27.37 (0.59)	23.64	27.37 (2.13)	**23.32**	**27.33 (2.11)**	26.44	30.13 (1.45)
90-03	18.52	21.91 (0.92)	17.24	**21.46 (2.46)**	**15.16**	21.63 (2.77)	21.36	23.05 (1.51)
90-04	13.4	14.35 (0.35)	10.2	11.16 (0.54)	**9.92**	**10.57 (0.35)**	14.28	15.35 (0.54)
90-05	42.4	44.85 (0.84)	**37.76**	40.61 (1.17)	37.88	**40.22 (1.32)**	40.52	42.15 (0.83)
90-06	61.8	63.43 (0.62)	63.08	65.32 (1.2)	**61.24**	**63.38 (1.01)**	66.56	69.52 (1.63)
90-07	26.48	28.57 (0.61)	24.6	26.15 (1.11)	**24.16**	**26.05 (0.96)**	25.92	27.22 (0.6)
90-08	65.6	67.01 (0.63)	66.28	67.6 (0.68)	**63.6**	**64.63 (0.37)**	67.2	68.8 (0.84)
90-09	44.2	45.46 (0.44)	42.28	43.74 (0.7)	**41.44**	**42.13 (0.48)**	45.48	46.8 (0.69)
90-10	38.04	38.97 (0.49)	**34.84**	36.5 (0.87)	**34.84**	**36.48 (0.85)**	39.08	40.22 (0.73)

Table 8: Minimum and average fitness values for every metaheuristic solving the r-CSP using the instances from SET1 (subsets from 80-* to 90-*).

		GRASP	ACO	ACO-LS	VNS
SET 1	Mean	1.84	2.91	1.63	3.59
	60-*	1.4	3.4	1.8	3.4
	65-*	1.4	3.2	1.7	3.6
	70-*	1.7	3	1.7	3.5
	75-*	1.4	3.2	2	3.4
	80-*	1.8	2.8	1.8	3.6
	85-*	2.2	2.7	1.3	3.8
	90-*	3	2.1	1.1	3.8

Table 9: Mean rank for every metaheuristic solving the r-CSP using the instances of SET1. The mean rank for the different subsets is also included.

that ACO-LS outperforms the other metaheuristics, specially in the subsets 300-* and 400-*, achieving most of the best values for the hardest instances. These results highlight the relevance of finding a feasible solution for the reference plan, since most of those instances are either not feasible or their feasibility is unknown. This also explains why VNS and ACO increase their performance when solving the instances of SET2 and SET3 with respect to solving instances of SET1. It also explains the performance drop of GRASP. Table 11 shows the mean rank of the metaheuristics solving the instances of SET2 and SET3. These values support our previous analysis and suggest that ACO-LS outperforms the other algorithms. The only exception would be the subset 200-*, where VNS achieves a lower value. However, the performance of VNS decreases for the hardest instances of subsets 300-* and 400-*, where its mean ranking falls to 2.9 and 3, respectively.

In addition to our previous analysis, we have applied different post-hoc procedures for ensuring the statistical significance of the mean rank values for SET1, SET2, and SET3. Specifically, we have considered Friedman's nonparametric test [25] and Wilcoxon ranksum test [27][2] (null hypothesis $R_i = R_j$, alternate hypothesis $R_i < R_j$, where R_i and R_j represent the average ranking of two different algorithms). We first test the null hypothesis using Friedman's non parametric test (which claims that there is no significant difference between the performance of the algorithms). The resulting $p-$values, shown in Table 12, are lower than the level of significance ($\alpha = 0.05$) for every set and the null hypothesis is therefore, rejected. Additionally to the $p-$values, Friedman's test results in $\chi^2_F = 106.52$ for SET1, $\chi^2_F = 22.6$ for SET2, and $\chi^2_F = 62.76$ for SET3.

We continue the analysis by using the Wilcoxon's ranksum test to every pair of metaheuristics by

[2]We ran these procedures using R. Friedman's test is computed using *friedman.test* from the standard package PMCMR. Wilcoxon's test is performed by *wilcox.test*, which is an R built-in function.

	Instances	GRASP		ACO		ACO-LS		VNS	
		Best	Avg (σ)	Best	Avg (σ)	Best	Avg (σ)	Best	Avg (σ)
SET 2	p10_93	15.88	17.53 (0.56)	10.6	11.77 (0.64)	**9.92**	**11.02 (0.58)**	10.72	11.87 (0.66)
	p16_81	10.28	11.07 (0.47)	5.24	5.89 (0.46)	**4.8**	**5.54 (0.35)**	5.52	6.65 (0.56)
	p19_71	9.52	10.94 (0.59)	7.36	7.98 (0.47)	**7**	**7.4 (0.27)**	7.08	7.59 (0.32)
	p21_90	4.28	5.75 (0.64)	3.44	4.07 (0.52)	3.36	3.91 (0.31)	**3.2**	**3.88 (0.35)**
	p26_82	12.8	13.53 (0.34)	9.12	10.33 (0.49)	**9.08**	**10.19 (0.47)**	10.16	10.98 (0.53)
	p36_92	12.68	13.36 (0.38)	10.68	11.17 (0.3)	**9.8**	**10.71 (0.27)**	10.32	10.93 (0.38)
	p41_66	6.4	7.29 (0.41)	**5.64**	7.18 (0.72)	5.84	**6.98 (0.75)**	6.08	7.03 (0.6)
	p4_72	10.08	11.17 (0.56)	6.2	6.82 (0.37)	**5.96**	**6.58 (0.37)**	6.2	7.35 (0.53)
	p6_76	7.28	7.83 (0.3)	6.84	7.31 (0.23)	**6.8**	**7.3 (0.24)**	7.16	7.58 (0.24)
SET 3	200_01	53.4	54.83 (0.75)	48.04	49.46 (0.58)	48.28	49.09 (0.45)	**46.76**	**48.84 (0.95)**
	200_02	20.68	22.87 (0.83)	16.44	17.32 (0.57)	15.56	16.85 (0.55)	**14.68**	**15.73 (0.6)**
	200_03	43.32	46.02 (0.99)	35.2	36.28 (0.66)	**34.76**	**35.79 (0.53)**	38.4	39.86 (0.83)
	200_04	25.64	27.65 (0.92)	19.32	19.83 (0.31)	**18.32**	**19.1 (0.4)**	19.72	21.25 (0.75)
	200_05	23.44	24.75 (0.62)	23.36	24.26 (0.47)	22.28	23.55 (0.59)	**20.92**	**22.07 (0.52)**
	200_06	24.52	25.7 (0.53)	22.24	23.23 (0.45)	**20.52**	22.53 (0.61)	20.96	**22.09 (0.54)**
	200_07	13.76	15.57 (0.92)	8.2	8.69 (0.27)	8.04	8.41 (0.22)	**7.84**	**8.23 (0.21)**
	200_08	30.28	32.29 (0.69)	26.2	26.86 (0.41)	**25.64**	**26.61 (0.38)**	27.24	29.33 (0.71)
	200_09	20.44	22.35 (0.77)	20.44	21.01 (0.39)	19.32	19.84 (0.32)	**17.72**	**18.32 (0.29)**
	200_10	34.76	37.17 (0.78)	36.24	37.91 (0.85)	35.08	36.81 (0.9)	**34.32**	**35.44 (0.52)**
	300_01	71.36	73.56 (1.22)	60.4	62.45 (0.78)	60.28	61.83 (0.76)	**57.88**	**60.97 (1.4)**
	300_02	44.16	47.69 (1.31)	28.8	29.71 (0.37)	**28.64**	**29.15 (0.28)**	30	32.87 (1.19)
	300_03	49.76	52.42 (1.09)	43.16	44.96 (0.7)	**41.96**	**44.28 (0.84)**	52.76	54.17 (0.82)
	300_04	55.08	57.41 (1.25)	41.84	43.42 (0.61)	**41.56**	**42.73 (0.56)**	44.52	47.26 (1.15)
	300_05	71.52	74.93 (1.43)	60.24	63.12 (1.33)	**59.44**	**62.01 (1.18)**	70.4	74.24 (1.84)
	300_06	44.16	47.46 (1.21)	**33.8**	**34.98 (0.63)**	33.92	35.17 (0.7)	41.12	44.09 (1.21)
	300_07	62	64.68 (1.03)	45.4	46.62 (0.7)	**44.84**	**46.35 (0.78)**	45.52	51.22 (2.47)
	300_08	38.4	41.16 (1.24)	26.72	27.43 (0.37)	**26.04**	**26.88 (0.42)**	27.28	29.87 (0.97)
	300_09	46.56	48.17 (0.95)	35.44	37.22 (0.68)	**35.12**	**36.53 (0.68)**	41.6	43.82 (1.2)
	300_10	113.16	116.2 (1.5)	104.64	107.21 (1.4)	**104.12**	**105.88 (0.92)**	106.48	110.25 (1.29)
	400_01	84.84	89.38 (1.5)	69.12	70.09 (0.5)	**68.96**	**70.05 (0.46)**	79.12	82.13 (1.36)
	400_02	70.64	74.18 (1.64)	50.84	52.43 (0.74)	**50.2**	**51.78 (0.67)**	65.2	67.85 (1.5)
	400_03	56.12	57.66 (0.81)	39.52	40.4 (0.26)	**39.4**	**40.19 (0.36)**	47.12	49.85 (1.34)
	400_04	106.04	108.47 (1.26)	86.16	91.23 (1.64)	**83.16**	**89.67 (2.04)**	95.44	97.38 (0.85)
	400_05	61.36	65.55 (1.74)	**46.44**	49.02 (1.2)	46.52	**48.83 (1)**	47.76	52.32 (1.75)
	400_06	87.16	89.49 (1.07)	**66.84**	**70.04 (1.72)**	67.88	70.1 (1.35)	72.32	75.24 (1.71)
	400_07	83.72	88.4 (1.43)	65.72	67.44 (0.78)	**65**	**66.76 (0.77)**	73.84	78.2 (2.23)
	400_08	45.8	48.24 (1.09)	**31.04**	32.29 (0.52)	31.4	**32.04 (0.36)**	40.6	43.71 (1.63)
	400_09	145.52	150.82 (1.66)	130.72	132.44 (0.84)	**129.56**	**131.64 (0.89)**	131.64	137.13 (2.08)
	400_10	126.72	132.31 (2.19)	118.88	121.46 (1.75)	**116.2**	**120.12 (1.67)**	120.28	124.2 (1.63)

Table 10: Minimum and average fitness values for every metaheuristic solving the r-CSP using the instances of SET2 and SET3.

		GRASP	ACO	ACO-LS	VNS
SET 2	Mean	4	2.44	1.11	2.44
SET 3	Mean	3.93	2.23	1.33	2.5
	200-*	3.9	2.8	1.7	1.6
	300-*	3.9	2	1.2	2.9
	400-*	4	1.9	1.1	3

Table 11: Mean rank for every metaheuristic solving the r-CSP using the instances from SET2 and SET3. In the case of SET3, we also include the mean rank for the different subsets.

		SET1	SET2	SET3
Friedman test		$2.2 \cdot 10^{-16}$	$4.893 \cdot 10^{-5}$	$1.511 \cdot 10^{-13}$
Wilcoxon ranksum test	GRASP vs. ACO / ACO vs. GRASP	$1.184 \cdot 10^{-9}$ / 1	1 / $6.103 \cdot 10^{-5}$	1 / $3.643 \cdot 10^{-12}$
	GRASP vs. ACO-LS / ACO-LS vs. GRASP	0.7723 / 0.2291	1 / $3.516 \cdot 10^{-5}$	1 / $3.509 \cdot 10^{-13}$
	GRASP vs. VNS / VNS vs. GRASP	$2.2 \cdot 10^{-16}$ / 1	1 / $6.461 \cdot 10^{-5}$	1 / $1.045 \cdot 10^{-11}$
	ACO vs. ACO-LS / ACO-LS vs. ACO	1 / $4.799 \cdot 10^{-15}$	0.9999 / $1.879 \cdot 10^{-4}$	1 / $2.33 \cdot 10^{-7}$
	ACO vs. VNS / VNS vs. ACO	10^{-6} / 1	0.4407 / 0.5981	0.0252 / 0.9757
	ACO-LS vs. VNS / VNS vs. ACO-LS	$2.2 \cdot 10^{-16}$ / 1	$6.463 \cdot 10^{-4}$ / 0.9995	$1.747 \cdot 10^{-6}$ / 1

Table 12: Resulting $p-$values of Friedman's test and Wilcoxon ranksum test for every pair of metaheuristics for SET1, SET2, and SET3.

considering a level of significance $\alpha = 0.05$. The resulting $p-$values confirm our previous analysis. In the case of SET1, GRASP and ACO-LS perform statistically better than ACO and VNS but the comparison between them is inconclusive. In the case of SET2, where GRASP drastically reduces its performance, ACO-LS is the only metaheuristic performing significantly better than the rest of the algorithms. ACO and VNS increase their performance and outrank GRASP but the comparison between them is non significant. Finally, in the case of SET3, both ACO and ACO-LS outrank the other metaheuristics and ACO-LS stands out as the statistically best method.

6. Concluding remarks

In this paper we have introduced a new model, r-CSP, that extends the traditional CSP when considering non-regular vehicles. This new model considers the concept of uncertain partial de-

mand for special vehicles, which is modeled based on different scenarios that are built using the defined production plans. We have presented both a formal definition of the problem and the objective function for solving the r-CSP. In addition, we have implemented a set of constructive metaheuristics for solving the problem: GRASP, VNS, ACO, and a memetic ACO (ACO-LS).

Regarding the resolution of the problem, we have generated compatible instances using the existing ones from the well known benchmark CSPLib. Using these generated instances, we applied the implemented metaheuristics to both the traditional CSP and the r-CSP. The results proved that our selected methods behave successfully compared to the current state of the art and they even improved the best known solution for two instances. Using the newly generated instances, we applied and compared the constructive metaheuristics for solving the r-CSP. After analyzing the results on these instances and applying Friedman's test and Wilcoxon ranksum test, we concluded that the memetic metaheuristic ACO-LS is the best solving method for every instance set regardless of the difficulty or the size of its instances. In contrast, GRASP obtained good results with the instances considering a low utilization rate, but its performance decayed for the harder instances. VNS and ACO obtained similar results solving instances with 200 or lower vehicles, but ACO outperformed VNS for the bigger instances.

Future work will be focused on defining additional robustness metrics and methods that allow the assessment of the quality of the found solutions beyond the basic constraint satisfiability approach. Additionally, we aim to propose and study multiobjective extensions for the CSP and r-CSP.

Acknowledgments

This work is supported by Spanish Ministerio de Economía y Competitividad under EXASOCO (ref. PGC2018-101216-B-I00) and OPTHEUS (ref. PGC2018-095080-B-I00) projects, including European Regional Development Funds (ERDF). M. Chica is also supported through the Ramón y Cajal program (RYC-2016-19800).

References

[1] Ahmadizar, F., 2012. A new ant colony algorithm for makespan minimization in permutation flow shops. Computers & Industrial Engineering 63, 355–361.

[2] Battaïa, O., Dolgui, A., 2013. A taxonomy of line balancing problems and their solutionapproaches. International Journal of Production Economics 142, 259–277. Anticipation of risks impacts and industrial performance evaluation in distributed organizations life cycles.

[3] Bautista, J., 2019. Robust car sequencing problem: Description, models and metrics. Dirección y Organización 68, 105–116.
[4] Bautista, J., Pereira, J., 2007. Ant algorithms for a time and space constrained assembly line balancing problem. European Journal of Operational Research 177, 2016–2032.
[5] Bautista, J., Pereira, J., Adenso-Díaz, B., 2008a. A beam search approach for the optimization version of the car sequencing problem. Annals of Operations Research 159, 233–244.
[6] Bautista, J., Pereira, J., Adenso-Díaz, B., 2008b. A GRASP approach for the extended car sequencing problem. Journal of Scheduling 11, 3–16.
[7] Bergen, M.E., van Beek, P., Carchrae, T., 2001. Constraint-based vehicle assembly line sequencing, in: Stroulia, E., Matwin, S. (Eds.), Advances in Artificial Intelligence, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 88–99.
[8] Boysen, N., Fliedner, M., 2007. Comments on "solving real car sequencing problems with ant colony optimization". European Journal of Operational Research 182, 466–468.
[9] Boysen, N., Fliedner, M., Scholl, A., 2009. Sequencing mixed-model assembly lines: Survey, classification and model critique. European Journal of Operational Research 192, 349–373.
[10] Butaru, M., Habbas, Z., 2005. The car-sequencing problem as n-ary csp – sequential and parallel solving, in: Zhang, S., Jarvis, R. (Eds.), AI 2005: Advances in Artificial Intelligence, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 875–878.
[11] Chica, M., Bautista, J., Cordón, Ó., Damas, S., 2016. A multiobjective model and evolutionary algorithms for robust time and space assembly line balancing under uncertain demand. Omega 58, 55–68.
[12] Chica, M., Cordón, O., Damas, S., 2011. An advanced multi-objective genetic algorithm design for the time and space assembly line balancing problem. Computers & Industrial Engineering 61, 103–117.
[13] Chica, M., Cordón, O., Damas, S., Bautista, J., 2010. Multiobjective, constructive heuristics for the 1/3 variant of the time and space assembly line balancing problem: ACO and random greedy search. Information Sciences 180, 3465–3487.
[14] Chica, M., Cordón, Ó., Damas, S., Bautista, J., 2013. A robustness information and visualization model for time and space assembly line balancing under uncertain demand. International Journal of Production Economics 145, 761–772.
[15] Cordeau, J.F., Laporte, G., Pasin, F., 2008. Iterated tabu search for the car sequencing problem. European Journal of Operational Research 191, 945–956.
[16] De Beukelaer, H., Davenport, G.F., De Meyer, G., Fack, V., 2017. JAMES: An object-oriented Java framework for discrete optimization using local search metaheuristics. Software: Practice and Experience 47, 921–938.
[17] Deb, K., Gupta, H., 2006. Introducing robustness in multi-objective optimization. Evolutionary computation 14, 463–494.
[18] Dorigo, M., Stützle, T., 2010. Ant colony optimization: overview and recent advances. Handbook of metaheuristics .
[19] Drexl, A., Kimms, A., Matthießen, L., 2006. Algorithms for the car sequencing and the level scheduling problem. Journal of Scheduling 9, 153–176.
[20] Estellon, B., Gardi, F., Nouioua, K., 2008. Two local search approaches for solving real-life car sequencing problems. European Journal of Operational Research 191, 928–944.
[21] Feo, T.A., Resende, M.G., 1989. A probabilistic heuristic for a computationally difficult set covering problem. Operations research letters 8, 67–71.
[22] Feo, T.A., Resende, M.G., 1995. Greedy randomized adaptive search procedures. Journal of global optimization 6, 109–133.
[23] Ferreira, J., Fonseca, C., Covas, J., Gaspar-Cunha, A., 2008. Evolutionary multi-objective robust optimization, in: Advances in evolutionary algorithms. InTech, pp. 261–278.
[24] Fliedner, M., Boysen, N., 2008. Solving the car sequencing problem via branch & bound. European Journal of Operational Research 191, 1023–1042.
[25] Friedman, M., 1940. A comparison of alternative tests of significance for the problem of m rankings. The Annals of Mathematical Statistics 11, 86–92.
[26] Gagné, C., Gravel, M., Morin, S., Price, W., 2008. Impact of the pheromone trail on the performance of ACO algorithms for solving the car-sequencing problem. Journal of the Operational Research Society 59, 1077–1090.
[27] García, S., Molina, D., Lozano, M., Herrera, F., 2008. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 special session on real parameter optimization. Journal of Heuristics 15, 617–644.
[28] Gavranović, H., 2008. Local search and suffix tree for car-sequencing problem with colors. European Journal of

Operational Research 191, 972–980.
[29] Gent, I.P., 1998. Two results on car-sequencing problems. Report APES-02-98.
[30] Gent, I.P., Walsh, T., 1999. CSPLib: a benchmark library for constraints, in: Principles and Practice of Constraint Programming–CP'99, Springer. pp. 480–481.
[31] Golle, U., 2011. Fitness landscape analysis and design of metaheuristics for car sequencing. On the Car Sequencing Problem: Analysis and Solution Methods , 100.
[32] Golle, U., Rothlauf, F., Boysen, N., 2015. Iterative beam search for car sequencing. Annals of Operations Research 226, 239–254.
[33] Gottlieb, J., Puchta, M., Solnon, C., 2003. A study of greedy, Local Search, and Ant Colony Optimization approaches for Car Sequencing Problems, in: Applications of Evolutionary Computing, Springer, Berlin, Heidelberg. pp. 246–257.
[34] Gravel, M., Gagne, C., Price, W.L., 2005. Review and comparison of three methods for the solution of the car sequencing problem. Journal of the Operational Research Society 56, 1287–1295.
[35] Gurevsky, E., Battaïa, O., Dolgui, A., 2012. Balancing of simple assembly lines under variations of task processing times. Annals of Operations Research 201, 265–286.
[36] Gurevsky, E., Battaïa, O., Dolgui, A., 2013. Stability measure for a generalized assembly line balancing problem. Discrete Applied Mathematics 161, 377–394.
[37] Hanafi, R., Kozan, E., 2014. A hybrid constructive heuristic and simulated annealing for railway crew scheduling. Computers & Industrial Engineering 70, 11 – 19.
[38] Hazır, Ö., Dolgui, A., 2013. Assembly line balancing under uncertainty: Robust optimization models and exact solution method. Computers & Industrial Engineering 65, 261–267.
[39] Hindi, K.S., Ploszajski, G., 1994. Formulation and solution of a selection and sequencing problem in car manufacture. Computers & Industrial Engineering 26, 203–211.
[40] Joly, A., Frein, Y., 2008. Heuristics for an industrial car sequencing problem considering paint and assembly shop objectives. Computers & Industrial Engineering 55, 295–310.
[41] Juan, A.A., Faulin, J., Grasman, S.E., Rabe, M., Figueira, G., 2015. A review of simheuristics: Extending metaheuristics to deal with stochastic combinatorial optimization problems. Operations Research Perspectives 2, 62–72.
[42] Krasnogor, N., Smith, J., 2000. A memetic algorithm with self-adaptive local search: TSP as a case study, in: Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation, Morgan Kaufmann Publishers Inc.. pp. 987–994.
[43] Lourenço, H.R., Martin, O.C., Stützle, T., 2003. Iterated Local Search. Springer US, Boston, MA. chapter 11. pp. 320–353.
[44] Mladenović, N., Hansen, P., 1997. Variable neighborhood search. Computers & Operations Research 24, 1097–1100.
[45] Morin, S., Gagné, C., Gravel, M., 2009. Ant colony optimization with a specialized pheromone trail for the car-sequencing problem. European Journal of Operational Research 197, 1185–1191.
[46] Moscato, P., 1989. On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms. Technical Report 826. Caltech Concurrent Computation Program. Pasadena, USA.
[47] Moscato, P., Cotta, C., Mendes, A., 2004. Memetic algorithms, in: New optimization techniques in engineering. Springer, pp. 53–85.
[48] Moya, I., Bautista, J., Chica, M., Damas, S., Cordón, O., 2018. Metaheurísticas constructivas para Car Sequencing Problem con flotas de vehículos especiales, in: Advances in Artificial Intelligence - 18th Conference of the Spanish Association for Artificial Intelligence, CAEPIA 2018, Granada, Spain, October 23-26, 2018, Proceedings, pp. 614–619.
[49] Papakostas, N., Pintzos, G., Giannoulis, C., Nikolakis, N., Chryssolouris, G., 2014. Multi-criteria assembly line design under demand uncertainty. Procedia CIRP 25, 86–92. 8th International Conference on Digital Enterprise Technology - DET 2014 Disruptive Innovation in Manufacturing Engineering towards the 4th Industrial Revolution.
[50] Parrello, B.D., Kabat, W.C., Wos, L., 1986. Job-shop scheduling using automated reasoning: A case study of the car-sequencing problem. Journal of Automated reasoning 2, 1–42.
[51] Prandtstetter, M., Raidl, G.R., 2008. An integer linear programming approach and a hybrid variable neighborhood search for the car sequencing problem. European Journal of Operational Research 191, 1004–1022.
[52] Puchta, M., Gottlieb, J., 2002. Solving Car Sequencing Problems by Local Optimization, in: Cagnoni, S., Gottlieb, J., Hart, E., Middendorf, M., Raidl, G.R. (Eds.), Applications of Evolutionary Computing, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 132–142.

[53] Ribeiro, C.C., Aloise, D., Noronha, T.F., Rocha, C., Urrutia, S., 2008. An efficient implementation of a VNS/ILS heuristic for a real-life car sequencing problem. European Journal of Operational Research 191, 596–611.
[54] Russell, S., Norvig, P., 1995. Artificial Intelligence: a modern approach. Prentice-Hall, Egnlewood Cliffs 25, 27.
[55] Siala, M., Hebrard, E., Huguet, M.J., 2015. A study of constraint programming heuristics for the car-sequencing problem. Engineering Applications of Artificial Intelligence 38, 34–44.
[56] Solnon, C., 2008. Combining two pheromone structures for solving the car sequencing problem with Ant Colony Optimization. European Journal of Operational Research 191, 1043–1055.
[57] Solnon, C., Cung, V.D., Nguyen, A., Artigues, C., 2008. The car sequencing problem: Overview of state-of-the-art methods and industrial case-study of the ROADEF'2005 challenge problem. European Journal of Operational Research 191, 912–927.
[58] Talbi, E.G., 2009. Metaheuristics: from design to implementation. John Wiley & Sons.
[59] Thiruvady, D., Ernst, A., Wallace, M., 2014. A Lagrangian-ACO matheuristic for car sequencing. EURO Journal on Computational Optimization 2, 279–296.
[60] Thiruvady, D.R., Meyer, B., Ernst, A., 2011. Car sequencing with constraint-based ACO, in: Proceedings of the 13th annual conference on Genetic and evolutionary computation, ACM. pp. 163–170.
[61] Warwick, T., Tsang, E.P., 1995. Tackling car sequencing problems using a generic genetic algorithm. Evolutionary Computation 3, 267–298.
[62] Zinflou, A., Gagné, C., Gravel, M., 2007. Crossover Operators for the Car Sequencing Problem, in: Proceedings of the 7th European Conference on Evolutionary Computation in Combinatorial Optimization, Springer-Verlag, Berlin, Heidelberg. pp. 229–239.
[63] Zinflou, A., Gagné, C., Gravel, M., 2010. Genetic algorithm with hybrid integer linear programming crossover operators for the car-sequencing problem. INFOR: Information Systems and Operational Research 48, 23–37.
[64] Zufferey, N., Studer, M., Silver, E.A., et al., 2006. Tabu Search for a Car Sequencing Problem., in: Proceedings of the 19th international Florida artificial intelligence research society conference, pp. 457–462.