Solving the Car Sequencing Problem from a Multiobjective Perspective

Ricardo José de Oliveira dos Reis

December 2007

Abstract

In 2003, the French Society of Operations Research jointly with Renault launched the ROADEF2005 Challenge. The purpose of this competition was to develop an algorithm to solve a new multiobjective version of the well known Car Sequencing Problem (CSP). This problem consists in sequencing a given set of vehicles along an assembly line while minimizing the number of violations of ratio constraints and the number of color changes.

The aim of this paper is to solve the ROADEF2005 CSP version, but using a different notion of optimality, called Pareto optimality. The advantage of this approach is that the decision maker does not have to judge the objectives about their priority. In addition it allows him to learn more about the trade-offs between the objectives. In order to solve the CSP we start by proposing an exact algorithm (ε-constraint) that allows us to obtain all the nondominated points. However, since this exact approach takes an infeasible amount of time to solve hard instances we further propose an heuristic scheme, based on local search procedures that enable us to obtain, at least, an approximation to the optimum set in a much shorter period of time.

Keywords: Car Sequencing Problem, Pareto optimality, ε-Constraint, Heuristic.
1. Introduction

In order to better answer to customers demand most European car manufacturers, like Renault (see Nguyen et. al. [2005]), rely on a build-to-order production system, instead of a build-to-inventory system. In the build-to-order system, vehicles are ordered by customers and not by dealerships like in the second one. This gives customers much more freedom to choose their vehicles configuration. However, from the manufacturer point of view this build-to-order perspective represents a logistic challenge, because it tends to generate a diversified flow of vehicles in assembly plants. If these vehicles are not placed in an appropriate order, the working bays along the assembly line, where options (components) are installed, might become overconstrained. As a consequence the production costs increases greatly. This is the industrial context in which the Car Sequencing Problem (CSP) is based on.

The Car Sequencing Problem was presented for the first time by Parello et. al. in 1986 (see Parello et. al. [1986]). In this first version the goal was to schedule vehicles along an assembly line, in order to minimize the number of ratio constraint violations on the working stations where employees install options (e.g. sunroof, air-conditioning) on vehicles. This combinatorial optimization problem has been shown to be NP-complete by Gent [1998], which means that there is no known algorithm that takes polynomial time in the instance size.

More recently, in 2003, the French Operations Research Society, jointly with Renault, proposed a challenge called ROADEF Challenge 2005, where a new multi-objective version of the Car Sequencing Problem was presented. This new version does not take into account only the minimization of the number of violations of assembly capacity constraints, but also the minimization of the number of paint color changes in the production line. In addition a constraint concerning the paint batch limit has been added.

In this paper we proposed two different algorithms to solve this new version of the Car Sequencing Problem (with some minor modifications regarding the number of objectives), but using a different notion of optimality, called Pareto optimality. The purpose of solving multiobjective combinatorial problems in terms of Pareto optimality is to find solutions (Pareto optimal) that cannot be improved in one objective without deteriorating their performance in at least one of the others. This notion is particularly useful in those situations, which the decision maker, due to some reason is not able to rank the objectives. We start by presenting an exact algorithm called $\varepsilon$-Constraint that solves an integer linear programming model several times until has found all the non-dominated points and the respective solutions. Due to several computational limitations this
algorithm is just able to solve simple instances in a feasible amount of time. For more complicated instances we propose an heuristic based on local search procedures and inspired in Evolutionary Algorithms that enable us to obtains, at least, an approximation to the nondominated set in a much shorter period of time.

2. Concepts Definition

In the following subsections we cover some important concepts related with multiobjective optimization problems.

2.1. Multiobjective Optimization

Most real-life problems require taking into account several conflicting objectives. Multiobjective optimization consists in optimizing several objectives, simultaneously, that are subject to several constraints. Usually these objectives are in conflict with each other and have different natures, i.e., they are measured in different units. In multiobjective optimization, normally, does not exist one single solution that simultaneously optimizes each objective. Instead, the goal of solving a multiobjective optimization problem is to find solutions that can only be improved in one objective by degrading at least one of the remaining objectives. This notion of optimality is called Pareto optimality. The multiobjective optimization problems are very hard to solve with exact methods, even if they are derived from easy single objective optimization problems (see Ehrgott [2000]).

A general Multiobjective Optimization Problem can be defined as follows

\[ \min_{s \in S} (f_1(s), \ldots, f_p(s)), \]

where \( S \subset \mathbb{R}^n \) is a feasible set and \( f : \mathbb{R}^n \to \mathbb{R}^p \) is a objective function vector, with \( P \) objectives. By \( Y = f(s) \subset \mathbb{R}^p \) we denote the image of the feasible set in the objectives space. In order to understand the meaning of optimal solutions in terms of multiobjective optimization the following concepts have to remain in mind:

Efficient solution or Pareto optimal: A feasible solution \( s \in S \) is efficient if there is no other solution \( s' \in S \) such that \( f_k(s') \leq f_k(s) \) for all \( k = 1, \ldots, P \) and \( f_j(s') < f_j(s) \) for some \( j \).

Efficient set: Set of all efficient solutions \( s \in S \).

Nondominated point: If \( s \) is an efficient solution than \( f(s) \) is called a nondominated point.

Nondominated set or Pareto front: The set of all nondominated points \( f(s) \in Y \)

In resume, efficiency refers to solutions \( s \) in the decision space while nondominance is used for objective vectors \( f(s) \in \mathbb{R}^p \) in the objectives space (see Figure 2.1)

![Efficient set and nondominated set](image)
In terms of optimization, the relation between objective function value vectors of two feasible solutions $s$ and $s'$, can be defined as follows:

- if $f(s) < f(s')$, then $f(s)$ dominates $f(s')$, i.e., $f_i(s) \neq f_i(s')$ and $f_i(s) \leq f_i(s')$, $i = 1, \ldots, P$;

- if $f(s) \leq f(s')$, then $f(s)$ weakly dominates $f(s')$, i.e., $f_i(s) \leq f_i(s')$, $i = 1, \ldots, P$;

- if $f(s) < f(s')$, then $f(s)$ strictly dominates $f(s')$, i.e., $f_i(s) < f_i(s')$, $i = 1, \ldots, P$;

2.3. Combinatorial Optimization Problems

A Combinatorial Optimization Problem (COP) like any other optimization problem is concerned with the efficient allocation of limited resources in order to meet desired objectives, but where some or all decision variables must be integer. Another distinctive feature of COP is the fact that solutions derive always from some combinatorial operation, such as an arrangement or a permutation or a partition of a limited number of objects. Because of this fact there are always a finite number of solutions (see Garey and Johnson [1979]).

The multiobjective version of a COP, is called Multiobjective Combinatorial Optimization Problem (MCOP).

3. The Car Sequencing Problem

As we said in the beginning of this paper the CSP version presented in the ROADEF’2005 challenge differs from the original version. Indeed, besides capacity constraints imposed by assembly line, it introduces paint batching constraints and it considers two classes of capacity constraints to take into account their priority. In the following section we explain in more detail the daily tasks of a car plant.

3.1 The ROADEF’2005 Car Sequencing Problem

Car plants received orders from their clients in real-time. The first task is to assign a production day to each ordered vehicle according to delivery dates and capacity constraints. The second task consists in sequencing the vehicles assigned to each production day, in order to satisfy as well as possible the requirements established by plant shops: paint shop and assembly shop.

Paint shop requirements

The objective in the paint shop is to minimize the consumption of paint solvent, which is used to wash the spray guns every time that there is a color change. Therefore to minimize the number of color changes we are interested in gathering vehicles, with the same color, together in a single batch. However the number of consecutive vehicles having the same color can not exceed a
certain value, designated by paint batch limit. This is considered a hard constraint, which means that can not be violated. The last vehicle from the previous production day is taken into account, while counting color changes.

**Assembly shop requirements**

The vehicles that require special operations have to be uniformly distributed throughout the assembly line, to avoid overloading the stations where options are assembled. This need of space between constrained vehicles is modeled by a ratio constraint \( \frac{N_{cp}}{Q_{cp}} \). Each component (or option) that requires extra operations has a ratio constraint associated to it. As we explained previously a ratio constraint \( \frac{N_{cp}}{Q_{cp}} \) means that should not exist more than \( N_{cp} \) vehicles affected by a certain component \( cp \) in each consecutive sequence of \( Q_{cp} \) vehicles. Contrarily to the paint batch limit, a ratio constraint can be violated, since in some circumstances, there is no sequence that satisfies all ratio constraints. Therefore the optimization objective is to minimize the violations number of ratio constraints.

To obtain an evenly distribution of constrained vehicles through the assembly line, we compute the number of ratio constraint violations on gliding subsequences throughout the daily sequence. The following example describes how ratio violations are evaluated:

Given the sequence _ X _ _ X X, where 'X' denotes a vehicle that requires an option (with ratio constraint 1/3) and '_' denotes a vehicle who does not. We start by splitting the sequence into four subsequences of size 3. Then we have to evaluate each of the subsequences according to the following formula:

\[
\text{Number of violations on a subsequence}^1 = (\text{Number of vehicles associated with the ratio constrain on the subsequence}) - (\text{ratio constraint numerator})
\]

Applying this formula to each of the subsequences we verify that the first three subsequences have no violations but the last one has one violation. Hence the whole sequence is evaluated to one violation. In this example we did not take into account the cars from the previous day, but it is necessary to consider the last \( Q-1 \) vehicles from the previous production day. See the example bellow

![Example](image)

**Figure 3.1: Counting ratio constraint violations**

In this new multiobjective version of the CSP there are two classes of ratio constraints: High Priority Ratio Constraints (HPRC) violations and Low Priority Ratio Constraints (LPRC) violations. However, for simplification purposes we assume that the HPRC and LPRC are incorporated in a single

---

1 Note: if the number of vehicles associated with the ratio constraint on the sequence is smaller than the ratio constraint numerator the result is obviously zero.
objective called Ratio Constraint (RC) violations.

3.2 Solving the ROADEF’2005 CSP from a Pareto Optimality Perspective

The goal of the ROADEF’2005 Challenge was to find a single solution that minimized all the objectives according to some weighted aggregated objective function. This notion of optimality is called scalarized optimality.

In the formula used in the event it is not very clear how the organizers calculated the weights, i.e., what techniques did they use? Furthermore we do not understand why the objectives were not normalized, since we are leading with different types of objectives. This lack of scientific support rises serious doubts about the effectiveness of this formula.

Due to the reasons stated above we decide to adopt the notion of Pareto optimality. The main advantage, among others, of using this notion of optimality is that the decision maker does not have to make any kind of judgment about the objectives relevance, a priori. In addition, the objectives do not need to be normalized.

4. Exact Approach

For solving the CSP introduced before we start by proposing an exact approach which consists in an exact algorithm called ε-constraint that runs several times an Integer Linear Programming (ILP) model until have found all the nondominated points. The goal of this exact approach is to find the nondominated set for any CSP instance.

4.1 The ε-Constrained Algorithm

The ε-constraint algorithm is one of the most well known exact methods to solve MCOPs. Besides this method there are various scalarized techniques that are also commonly used to solve this kind of problems. However these last techniques are unable to find unsupported optimal solutions, i.e., solutions whose objective value vector does not lie in the border of the convex hull. This was the main reason why we choose to implement the ε-constraint.

The general ε-Constrained method consists in pre-defining a virtual grid in the objectives space and solving different single objective constraint problems constrained to each grid cell (see Haimes et. al. [2004]). Then by decreasing (for minimization) systematically the constraint bound ε through a pre-defined constant (interval) Δε, we can obtain different nondominated points that constitute the nondominated set, also called Pareto front. The ε-Constrained problem can be formulated as:

\[ \min_{x \in \mathbf{X}} f_j(x) \]

subject to \( f_k(x) \leq \varepsilon_k \)

\( k = 1, \ldots, p \) and \( k \neq j \)

where \( f_j(x) \) represents the objective function to be minimized, and \( f_k(x) \) represents the
remaining objective functions that are used as constraints through a constraint bound $\varepsilon$.

One of the disadvantages of this method is that it requires an initial lexicographic solution, which is used to define the initial value of the constraint bound $\varepsilon$. To find this value we need to optimize lexicographically the different objectives. This can be done by solving each objective sequentially by decreasing order of relevance and using optimal solutions of higher relevance objectives as constraint.

In our implementation we choose the number of color changes ($Z_1$) as the objective to minimize while the number of ratio constraint violations ($Z_2$) works as a constraint. The first stage of our algorithm consists in finding the lexicographic solutions. We start by solving the ILP model minimizing $Z_1$. Then we use $Z_1$ as a constraint and solve the ILP model again but minimizing $Z_2$. The value of $Z_2$ is then used to initialize the constraint bound $\varepsilon$. To find the other lexicographic solution we just have to exchange the lexicographic order of the objectives and repeat the whole process. After finding the lexicographic solutions it begins the second stage of the method, which consists in finding the $\varepsilon$-constraint solutions. This is an iterative algorithm that in each iteration performs the following operations: solve the ILP model minimizing $Z_1$ with $Z_2$ as constraint; if the new solution weakly dominates the previous one then we replace the dominated one by this new solution, otherwise we simply add it to the nondominated set; then before terminating the iteration we decrease the constraint bound by an amount of $\Delta \varepsilon$. The algorithm repeats the process until it reaches the other lexicographic solution that corresponds to the minimum value of objective 2. The following algorithm illustrates this second stage.

<table>
<thead>
<tr>
<th>$\varepsilon$-Constraint</th>
</tr>
</thead>
</table>
| BEGIN; 
NoNon-dominatedSet := \{\}; 
$\varepsilon = second\ maximal\ value\ among\ given\ by\ \text{solnum}(L_1, L_2)$; 
$\Delta \varepsilon = 1$ 
WHILE not each solution L_1, L_2 DO 
Solve model minimizing Z_1; 
IF the new solution dominates the current one THEN replace the dominated one by the new one; 
ELSE add new solution to Non-dominatedSet; 
Update constraint bound $\varepsilon = \varepsilon - \Delta \varepsilon$; 
and FOR; 
RETURN Non-dominatedSet; |

4.2 Performance of the $\varepsilon$-Constraint method

The ILP model was modeled through Mixed Integer Programming (MIP) and implemented in GAMS (General Algebraic Modeling System) as well as the $\varepsilon$-constraint algorithm. For solving the model we used CPLEX 10.0. For testing the performance of this exact algorithm we build several CSP instances with different features. In the following table (Table 4.1) we present for each instance, the execution time taken by the procedure, the number of nondominated points found, the bounds on the nondominated set and the corresponding $\Delta \varepsilon$. 

4.2 Performance of the $\varepsilon$-Constraint method
Analyzing the results shown in Table 4.1, we can take the following conclusions about the ε-constraint algorithm performance:

- Most of the instances tested require several hours to be solved. Only very small instances like 1 and 2 can be solved in a short period of time.
- Instances with a higher number of vehicles tend to take more time to be solved because the decision space is bigger.
- Instances 10 and 11 are equal to each other, but the last one has a higher paint batch limit and one more nondominated point. This observation leads us to conclude that less constrained instances tend to have more nondominated points and optimal solutions.
- There exist several instances with a small number of vehicles that requires much more time than others with more vehicles. The difficulty level on an instance does not depend just on the number of vehicles but also on the paint batch limit, ratio constraints and the number of vehicles affected by those ratios, expressed by utilization rates. This means that instances highly constrained may be harder to solve than instances with a higher number of vehicles but less constrained (see the example in Table 4.2). In conclusion, the difficulty level inherent to an instance depends on the size of the decision space but also on the constraints.

Table 4.1: Results on the ε-constraint

<table>
<thead>
<tr>
<th>Instance</th>
<th>Execution time</th>
<th>Number of Nondominated Points</th>
<th>Bounds on the Nondominated Set (Ideal and Nadir Points)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance 1</td>
<td>3sec</td>
<td>5</td>
<td>(0, 0); (4, 4)</td>
</tr>
<tr>
<td>Instance 2</td>
<td>1min 7sec</td>
<td>9</td>
<td>(0, 0); (4, 11)</td>
</tr>
<tr>
<td>Instance 3</td>
<td>2h 49min 4sec</td>
<td>10</td>
<td>(0, 0); (4, 22)</td>
</tr>
<tr>
<td>Instance 4</td>
<td>47sec</td>
<td>3</td>
<td>(0, 0); (4, 29)</td>
</tr>
<tr>
<td>Instance 5</td>
<td>1h Min 2sec</td>
<td>10</td>
<td>(0, 0); (4, 22)</td>
</tr>
<tr>
<td>Instance 6</td>
<td>12h 57min 7sec</td>
<td>14</td>
<td>(0, 0); (5, 26)</td>
</tr>
<tr>
<td>Instance 7</td>
<td>6h 51min 12sec</td>
<td>10</td>
<td>(0, 0); (5, 27)</td>
</tr>
<tr>
<td>Instance 8</td>
<td>3h 46min 3sec</td>
<td>14</td>
<td>(0, 0); (5, 35)</td>
</tr>
<tr>
<td>Instance 9</td>
<td>14h 20min 3sec</td>
<td>11</td>
<td>(0, 0); (6, 29)</td>
</tr>
<tr>
<td>Instance 10</td>
<td>6h 22min 42sec</td>
<td>7</td>
<td>(0, 0); (6, 32)</td>
</tr>
<tr>
<td>Instance 11</td>
<td>9h 19min 32sec</td>
<td>8</td>
<td>(0, 0); (6, 35)</td>
</tr>
<tr>
<td>Instance 12</td>
<td>11h 33min 40sec</td>
<td>10</td>
<td>(0, 0); (6, 42)</td>
</tr>
</tbody>
</table>

Table 4.2: Results of the ε-constraint

Making an overview, the results clearly indicate that the major drawback of this approach is the time required to find the nondominated set. One possible alternative to reduce the execution time is to increase the pre-defined factor Δε which is equivalent to reduce the number of search intervals. However, since only one solution can be found per interval, the algorithm will certainly “miss” some nondominated points. To verify this fact we tested some of the previous instances using a different Δε (see Table 4.3).

Table 4.3: Results of the ε-constraint with a different Δε

<table>
<thead>
<tr>
<th>Instance</th>
<th>Execution Time</th>
<th>% of Nondominated Points found</th>
<th>% of the Execution Time (on the original time)</th>
<th>Δε</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance 3</td>
<td>2h 49min 4sec</td>
<td>50%</td>
<td>61.3%</td>
<td>2</td>
</tr>
<tr>
<td>Instance 5</td>
<td>5h 48min 42sec</td>
<td>60%</td>
<td>33.7%</td>
<td>3</td>
</tr>
<tr>
<td>Instance 8</td>
<td>3h 32min 31sec</td>
<td>40%</td>
<td>25.6%</td>
<td>4</td>
</tr>
</tbody>
</table>
Next we tried to solve instances with a size up to 60-70 vehicles but the algorithm was unable to find some of the nondominated points.

5. Heuristic Approach

Due to the limitations of the exact method presented previously we decided to develop an heuristic, based on local search methods that enable us to solve harder and bigger CSP instances.

During the ROADEF’2005 Challenge a wide range of general-purpose heuristic schemes, commonly called metaheuristics, were presented by the competitors, like Simulated Annealing (see Risler [2004]), Tabu Search (see Cordeau [2005]), Ant Colony Optimization (see Gravel et al. [2005]) among others, with very interesting results.

5.1. Implementation of the Heuristic

A general evolutionary algorithm comprises three stages: recombination, mutation, and selection. In recombination, two individuals (parents), are chosen according to some criteria, and combined by some crossover operator to generate new individuals, called offspring. The mutation operator changes some characteristics of individuals, for instance by performing random neighborhood moves. Finally, the selection operator is used to keep the population at a constant size by choosing individuals with higher fitness and discarding the other ones.

The heuristic scheme presented here performs just two of these operations: mutation and selection. There are several reasons for not using the recombination operator. First of all, it is not clear how can we transmit certain features from the parents to the offsprings; secondly this operator is not very simple to implement and requires some computational effort. Furthermore we aim to show that the mutation operator, sometimes underestimated (see Bäck [1996]), has a much more important role in Genetic Algorithms than most researchers believe. To improve the potential of the mutation operator we suggest some simple and intuitive techniques which aspire to increase the effectiveness of this operator as well as to speed up the evaluation procedure. The heuristic introduced next uses solely a mutation operator followed a selection operator capable of preserving elitism and diversification among population individuals. The following scheme resumes very briefly the main steps of our heuristic.

<table>
<thead>
<tr>
<th>Heuristic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BEGIN</strong></td>
</tr>
<tr>
<td>Create initial Population;</td>
</tr>
<tr>
<td>Archive // archive where the best solutions (sequences) are stored;</td>
</tr>
<tr>
<td>it // number of iterations without updating the archive;</td>
</tr>
<tr>
<td>WHILE time limit is not reached DO</td>
</tr>
<tr>
<td>IF it &gt; 2000</td>
</tr>
<tr>
<td>THEN Return Population;</td>
</tr>
<tr>
<td>FOR each element of the Population DO</td>
</tr>
<tr>
<td>Choose randomly two positions;</td>
</tr>
<tr>
<td>Apply mutation operation;</td>
</tr>
<tr>
<td>Evaluate new solution;</td>
</tr>
<tr>
<td>end FOR;</td>
</tr>
<tr>
<td>Rank original and transformed solutions through MOGA procedure;</td>
</tr>
<tr>
<td>Update current Population and Archive;</td>
</tr>
<tr>
<td>end WHILE;</td>
</tr>
<tr>
<td>RETURN Best solutions in the Archive;</td>
</tr>
</tbody>
</table>
The algorithm starts from an initial population, which is created through a constructive heuristic. Then, while the time limit is not reached, we perform a neighboring move for each sequence/element. This is equivalent to generate a new solution. After evaluating the new solutions, we select solely the best ones to update the Population as well as the Archive. If the archive is not updated within 2000 iterations, then we restart population. When the time limit is reached the algorithm terminates and returns the non-dominated solutions in the Archive.

To implement the algorithm some important parameters need to be defined:

Population Size: The population size is 10 solutions. This is the size used for relatively small instances.

Termination Condition: The termination condition is defined in terms of computational time. The time limit to terminate the algorithm is 600 seconds.

Number of iterations until restart population: To escape from local optima we restart the population after 2000 iterations without inserting a solution in the archive (experimentations have shown that this value is reasonable for small instances).

In the following subsections we explain in more detail some relevant aspect that contribute for the good performance of this heuristic.

Creating initial population

The initial population consists of a set of car sequences. The constructive heuristic used to generate the initial population is very similar to the insertion method proposed by Risler [2004], with some minor adaptations in respect to the evaluation function. We start from an empty population with empty sequences and then for each sequence we insert vehicles one by one at the best possible position. This is done by computing the evaluation function for all possible insertion positions and then choosing the one with minimal value. Ties results in choosing the leftmost position. The evaluation of the two criteria is done through a weighted sum function that aggregates the objectives, color changes and ratio constraint violations, in the same evaluation function. To ensure diversification among population sequences, we use a different weight vector for each sequence. The sum of the weights must be equal to one.

Neighborhood

The mutation is the only operator that we got to exploit the decision space. Basically the mutation operator performs a local search within a specific neighborhood. The definition of the neighborhood it is crucial for the performance of any stochastic local search algorithm. While taking a decision about what neighboring moves should be applied we should be concerned about the effectiveness of those moves but also with the evaluation speed of the new solutions that result from those moves. Taking these
two principles into consideration we defined the following neighborhood:

- Exchange 1: exchanging two random vehicles in a sequence;
- Exchange 2: exchanging two consecutive vehicles;
- Insert: moving one vehicle to a different position;

**Trying to improve neighboring moves**

The decision about what neighboring move should be applied in each situation, in order to maximize the chances of success, depends mostly of the positions chosen. The fastest way to select two positions \( x \) and \( y \) is picking them randomly. Then according to the features of the vehicles in that positions and some other information concerning number of violations of each ratio constraint in each window, we take a decision about what should be the best move. Next we present in which situation we apply each neighboring move.

- If vehicle in position \( x \) has the same color of vehicle in position \( y \) or if they have some components in common, exchanging them might be a good idea since the chances of deteriorating the original sequence is reduced.
- If vehicle in position \( x \) provokes too many violations, it might be beneficial to insert him in a different position \( y \).
- When none of the situations stated above are verified exchanging adjacent vehicles in the sequence reduces the risk of deterioration while making faster the evaluation procedure.

**Speeding up the evaluation procedure**

Executing neighboring moves is not a time consuming operation. Like mentioned in Nouioua et al. [2004], the bottleneck in terms of complexity is clearly the evaluation of the neighboring solutions. When we evaluate a sequence of vehicles for the first time we have to compute the number of violations for each window as shown in Figure 5.1 (assume a ratio constraint 1/3). However when you perform a neighboring move, for instance exchanging the last two vehicles in the sequence shown in Figure 5.1, we just need to reevaluate those windows that are perturbed by the move, which depends only on the denominator of each ratio constraint. This can be done thanks to the special data structure introduced before (see Figure 5.2)

![Figure 5.1: Evaluating the sequence for the first time](image1)

![Figure 5.2.: Evaluating neighboring sequence](image2)
After the generation and evaluation of the neighboring solutions it is time to select which of them will be incorporated in the current population and which will be eliminated. The goal of this operator is to ensure elitism and diversity among the populations elements.

In the context of multiobjective evolutionary algorithms, there exist various selection procedures. The most well known are the MOGA (see Fonseca and Fleming [1993]), NSGA-II (see Deb et al., [2002]) and SPEA-II (see Zitzler et al. [2001]). From all these techniques we decided to implement the MOGA (Multi-Objective Genetic Algorithm) procedure. This procedure is very simple to implement and at the same time very efficient. In addition do not require so much computation like some of the other methods referred before.

The MOGA consists in a scheme in which the fitness value of a solution corresponds to the number of solutions in the current population by which it is dominated plus one. Then solutions are ranked according to those values. The better ones are then inserted in the current population and the worst solutions are removed.

Besides the current population there is an archive where the best solutions created so far are stored. In practice only those solutions with a fitness value equal to 1 are added to the archive, unless there is already a solution with the same objective function values.

6. Results of the Heuristic

The only way to evaluate the performance of our heuristic is by comparing its results with the ones provided by the exact algorithm. Besides solutions quality we are also interested in comparing the time required by the heuristic to find the optimal or near optimal solutions. For this purpose we applied our heuristic to the same instances used before to characterize the performance of the ε-constraint. Since the heuristic may not return the same outcome twice we decide to run the algorithm three times for each instance and select the worst case, i.e., the one with the worst execution time. Table 6.1 summarizes the results obtained with the heuristic approach as well as the results obtained previously with the exact approach.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Number of Dominated Points (Heuristic)</th>
<th>Execution Time (Heuristic)</th>
<th>Number of Dominated Points (ε-constraint)</th>
<th>Execution Time (ε-constraint)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance 1</td>
<td>3</td>
<td>9 sec</td>
<td>3</td>
<td>9 sec</td>
</tr>
<tr>
<td>Instance 2</td>
<td>5</td>
<td>1 sec</td>
<td>5</td>
<td>5 sec</td>
</tr>
<tr>
<td>Instance 3</td>
<td>10</td>
<td>3 sec</td>
<td>10</td>
<td>5 sec</td>
</tr>
<tr>
<td>Instance 4</td>
<td>5</td>
<td>1 sec</td>
<td>5</td>
<td>4 sec</td>
</tr>
<tr>
<td>Instance 5</td>
<td>10</td>
<td>10 sec</td>
<td>10</td>
<td>10 sec</td>
</tr>
<tr>
<td>Instance 6</td>
<td>14</td>
<td>12 sec</td>
<td>14</td>
<td>12 sec</td>
</tr>
<tr>
<td>Instance 7</td>
<td>10</td>
<td>6 sec</td>
<td>10</td>
<td>6 sec</td>
</tr>
<tr>
<td>Instance 8</td>
<td>14</td>
<td>12 sec</td>
<td>14</td>
<td>12 sec</td>
</tr>
<tr>
<td>Instance 9</td>
<td>11</td>
<td>14 sec</td>
<td>11</td>
<td>14 sec</td>
</tr>
<tr>
<td>Instance 10</td>
<td>5</td>
<td>5 sec</td>
<td>5</td>
<td>5 sec</td>
</tr>
<tr>
<td>Instance 11</td>
<td>10</td>
<td>10 sec</td>
<td>10</td>
<td>10 sec</td>
</tr>
</tbody>
</table>

Table 6.1: Results of the heuristic and ε-constraint
The results presented in this table indicate us that for all the instances tested the heuristic was able to find all the nondominated points in a remarkable short period of time. In conclusion the results show that this heuristic is able to match the performance of the exact approach with respect to solution quality on the instances tested, in a few seconds of computational time.

The neighboring moves chosen as well as the techniques used to improve their hypotheses of success have given a great contribution for these results. Furthermore the method used to generate the initial solution has shown a great ability to find very good initial solutions. Last but not least the technique used to speed up the evaluation procedure allowed us to increase the number neighboring moves.

Another interesting fact that we noticed is the speed as the heuristic converge to the nondominated set. Figure 6.1 illustrates the best solutions found on instance 6 after performing a pre-defined number of iterations. The black dots represent the nondominated set (or Pareto front) and the red triangles the best solutions found so far, which are contained in the archive. Analyzing the various graphics we verify that after performing 5000 iterations, which takes 3 seconds, the heuristic has already found half of the nondominated points and some other points are very close from other optimal points. This means that 17% of the execution time was sufficient to find half of the nondominated set.

![Figure 6.1: Best solutions found on instance 6, a) in the initial population, b) at iteration 1500, c) at iteration 5000 and d) iteration 30000.](image)

The results obtained so far on the instances tested are very promising. However to confirm our expectation about the performance of this heuristic we should apply it to a more extensive and demanding collection of CSP instances.

**Conclusions**

In this paper we proposed an exact and an heuristic method to solve the CSP in terms of Pareto optimality. The exact approach consisted in a general ε-constraints algorithm which runs several times an underlying ILP model until have found all the nondominated points. This algorithm has shown to be capable of finding all nondominated points, but just for small and relatively easy CSP instances with a size up to 60 vehicles. In
addition the amount of time required to solve these instances was usually enormous which from an industrial perspective is completely infeasible. Moreover, as the problem becomes more constrained, the exact approach tends to take an infeasible amount of time even for small instances. Even so, we tried to enlarge the search intervals \( \Delta \varepsilon \) and increase the CPLEX run time, so we could solve harder CSP instances. The purpose of this attempt was not to find all nondominated points but rather a sample of these. The results of this attempt were not very positive since most of the points obtained were weakly dominated. The reason for this has to do mainly with the value of the parameter \( \Delta \varepsilon \). This is definitely a major drawback of the \( \varepsilon \)-constraints.

Due to the limitations of the exact method shown before we decided to implement an heuristic, whose purpose was to find good (approximate) solutions in a feasible amount of time. This goal was clearly achieved. Our heuristic revealed to be fast and effective in finding good approximations to the Pareto optimal, due mainly to the strategic neighboring moves applied and to the skilful techniques used to speed up the algorithm. The insertion method had also revealed to be an important tool for generating a diversified range of good initial solutions. For the same instances used on the exact algorithm, the local search algorithm has shown to be capable of finding all nondominated points in a remarkable short period of time. Such promising results are very encouraging.
Bibliography


Appendix

A. Integer Linear Programming Formulation

Indices

\( i \in \{1, \ldots, NPos\} \), where \( NPos \) is the number of positions to fulfill, that is equivalent to the number of cars to schedule;

\( m \in \{1, \ldots, NCrP\} \), where \( NCrP \) is the number of cars from the previous production day that have to be taken into account when computing the number of ratio constraint violations

\( cp \in \{1, \ldots, NCp\} \), where \( NCp \) is the number of components;

\( cl \in \{1, \ldots, NCl\} \), where \( NCl \) is the number of colors;

\( k \in \{1, \ldots, NConf\} \), where \( NConf \) is the number of configurations;

Input Parameters

\( s \) is the paint batch limit;

\( N_{cp}/Q_{cp} \) is the ratio constraint for component \( cp \);

\( d_k \) is the demand for configuration \( k \);

\( AL_{cl,k} \) is a binary matrix that indicates if vehicles with configuration \( k \) requires color \( cl \) or not;

\( AP_{cp,k} \) is a binary matrix that indicates if vehicles with configuration \( k \) requires component \( cp \) or not;

\( EI_{cl,m} \) is a binary matrix that indicates if vehicle \( m \) from the previous production day has color \( cl \) or not;

\( EP_{cp,m} \) is a binary matrix that indicates if vehicle \( m \) from the previous production day has component \( cp \) or not;

\( d_{cl} \) is the number of times that color \( cl \) is needed

\[
  d_{cl} = \sum_{k} (AL_{cl,k} \cdot d_k) \quad \forall cl \in \{1, \ldots, NCl\}
\]

\( d_{cp} \) is the number of times that component \( cp \) is needed
\[ d_{cp} = \sum_{k} (c_{p \cdot k} \cdot d_{k}) \quad \forall cp \in \{1, \ldots, NCp\} \]

**Decision Variables**

- \( p_{k,i} \): indicates if the car at position \( i \) has configuration \( k \);
- \( r_{cp,i} \): indicates the quantity of component \( cp \) used so far until position \( i \);
- \( g_{cp,i} \): indicates the number of violations on component \( cp \) for the window finishing at position \( i \);
- \( d_{cl,i} \): specifies the color \( cl \) of the car that occupies position \( i \) in the production line;
- \( w_{cl,i} \): indicates if a change to color \( cl \) occurs at position \( i \);
- \( Z_1 \): objective 1, color changes;
- \( Z_2 \): objective 2, ratio constraint violations;

**Objective functions**

\[ \min \ Z_1 = \sum_{c_{l}=1}^{NC1} \sum_{i=1}^{NPos} w_{c_{l},i} \]

\[ \min \ Z_2 = \sum_{cp=1}^{NCp} \sum_{i=1}^{NPos} g_{cp,i} \]  

**Subject to**

\[ \sum_{i=1}^{NPos} p_{k,i} = d_{k} \quad \forall k \in \{1, \ldots, NC_{Conf}\} \]  

\[ \sum_{k=1}^{NC_{Conf}} p_{k,i} = 1 \quad \forall i \in \{1, \ldots, NPos\} \]  

\[ \sum_{i=1}^{NPos} \sum_{k=1}^{NC_{Conf}} c_{l \cdot k \cdot i} \cdot p_{k,i} = d_{c_{l}} \quad \forall c_{l} \in \{1, \ldots, NC_{l}\} \]
\[
\sum_{i=1}^{N_{op}} \sum_{k=1}^{N_{conf}} c_{cp,k} \cdot p_{ki} = d_{cp} \quad \forall c \in \{1, \ldots, N_{cp}\} (6)
\]

\[ r_{cp,0} = 0 \quad \forall cp \in \{1, \ldots, N_{cp}\} (7) \]

\[ r_{op,i} \geq 0 \quad \forall cp \in \{1, \ldots, N_{cp}\}, \forall i \in \{1, \ldots, N_{pos}\} (8) \]

\[ r_{cp,1} = \sum_{k=1}^{N_{conf}} (a_{cp,i} \cdot p_{ki}) \quad \forall cp \in \{1, \ldots, N_{cp}\} (9) \]

\[ r_{op,i} = r_{op,(i-1)} + \sum_{k=1}^{N_{conf}} (a_{cp,i} \cdot p_{ki}) \quad \forall cp \in \{1, \ldots, N_{cp}\}, \forall i \in \{2, \ldots, N_{pos}\} (10) \]

\[ g_{op,i} \geq 0 \quad \forall cp \in \{1, \ldots, N_{cp}\}, \forall i \in \{1, \ldots, N_{pos}\} (11) \]

\[ g_{op,i} \geq r_{op,i} + \sum_{m=1}^{q_{cp} - i} e_{op,m} - N_{op} \quad \forall cp \in \{1, \ldots, N_{cp}\}, \forall i \in \{1, \ldots, Q_{cp} - 1\} (12) \]

\[ g_{op,i} \geq r_{cp,i} - r_{op,i} - N_{op} \quad \forall cp \in \{1, \ldots, N_{cp}\}, \forall i \in \{Q_{cp}, \ldots, N_{pos}\} (13) \]

\[ b_{cil} = \sum_{k=1}^{N_{conf}} (a_{cil,k} \cdot p_{ki}) \quad \forall ci \in \{1, \ldots, N_{ci}\}, \forall i \in \{1, \ldots, N_{pos}\} (14) \]

\[ \sum_{n=i-s}^{i} b_{cil} \leq s \quad \forall i \in \{s + 1, \ldots, N_{pos}\} (15) \]

\[ w_{cil} \geq b_{cil} - e_{cil} \quad \forall ci \in \{1, \ldots, N_{ci}\} (16) \]

\[ w_{cil} \geq b_{cil} - b_{cil-1} \quad \forall ci \in \{1, \ldots, N_{ci}\}, \forall i \in \{2, \ldots, N_{pos}\} (17) \]