INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Solving the Car Sequencing Problem from a Multiobjective Perspective

Ricardo José de Oliveira dos Reis

Dissertação para obtenção do grau de Mestre em

Engenharia e Gestão Industrial

Júri

Presidente:	Prof. Dra. Ana Póvoa
Orientadores:	Prof. Dr. José Rui Figueira
	Prof. Dr. Luís Paquete
Vogal:	Prof. Dr. Paulo Correia

Dezembro 2007

Acknowledgments

I would like to thank my supervisor Prof. Dr. Rui Figueira for his help, enthusiastic support and encouragement throughout this thesis. Furthermore I also want to thank Prof. Dr. Luis Paquete for the enlightening discussions that we had on several research subjects, for his encouragement and hospitality during my stay in Universidade do Algarve. I am also thankful to my parents and sisters for all the support given in the last few months. Last, but not the least, I would like to thank to all my friends that in some way on another help me doing this thesis.

Abstract

In 2003, the French Society of Operations Research jointly with Renault launched the ROADEF'2005 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 thesis is to solve the ROADEF'2005 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 and respective efficient solutions. 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 nondominated set in a much shorter period of time.

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

Sumário

Em 2003 a Sociedade Francesa de Investigação Operacional organizou, com o apoio da Renault, um concurso designado ROADEF'2005 Challenge. O objectivo desta competição consistia em desenvolver um algoritmo para resolver uma nova versão multi-objecivo do já conhecido Car Sequencing Problem. Este problema consiste em ordenar um conjunto de veículos ao longo da linha de produção procurando minimizar o número de violações dos rácios de restrição e o número de mudanças de cor. O objectivo desta tese consiste em resolver esta nova versão multi-objectivo do Car Sequencing Problem, mas usando uma noção de optimalidade diferente, designada de optimalidade de Pareto. A vantagem desta noção de óptimo é que o decisor não necessita de realizar qualquer tipo de julgamento sobre a relevância dos objectivos. Para além desta o conhecimento das diversas soluções de Pareto permitem ao decisor aprender mais sobre as contrapartidas entre as diversas soluções. Para resolver este problema começamos por propor um algoritmo exacto, designado de ε-constraint, que nos permite obter soluções eficientes cuja imagem no espaço dos objectivos corresponde ao conjunto não-dominado também conhecido por frente de Pareto. Contudo, devido a determinadas limitações computacionais este método revelou-se incapaz de resolver instâncias mais exigentes. Tendo em consideração as limitações dos métodos exactos acima referidos propomos um segundo método, uma heurística baseada em processos de procura local que nos permitem obter, pelo menos, soluções que estão próximas das óptimas num curto período de tempo.

Palavras chave: Car Sequencing Problem, Optimalidade de Pareto, ε-Constraint, Heurística.

Contents

1. Introduction.. 1

2. Multiobjective Combinatorial Optimization.. 4
- 2.1 Combinatorial Optimization... 4
- 2.2 Multiobjective Combinatorial Optimization.................................... 5
  - 2.2.1 Problems... 5
  - 2.2.2 Pareto Optimality.. 7
  - 2.2.3 Other Notions of Optimality.. 10

3. Solution Methods for Multiobjective Combinatorial Optimization Problem....... 12
- 3.1 Exact Algorithms.. 12
  - 3.1.1 Scalarized Techniques... 13
  - 3.1.2 The ε-Constraint Method... 14
- 3.2 Heuristics and Metaheuristics... 16
  - 3.2.1 Tabu Search... 17
  - 3.2.2 Simulated Annealing... 18
  - 3.2.3 Multiobjective Evolutionary Algorithms.................................. 19

4. The Car Sequencing Problem.. 24
- 4.1. The ROADEF'2005 Car Sequencing Problem....................................... 24
  - 4.1.1. Planning and Scheduling Process.. 25
  - 4.1.2. Paint Shop Requirements.. 25
  - 4.1.3. Assembly Line Requirements... 26
  - 4.1.4. ROADEF Objective Function.. 27
- 4.2. Solving ROADEF'2005 CSP from a Pareto Perspective............................ 29
- 4.3. Integer Linear Programming Formulation. 29

5. The ε-Constraint Algorithm... 37
- 5.1. Lexicographic Optimization... 37
- 5.2. Implementation of the ε-Constraint.. 38
- 5.3. Performance of the ε-Constraint Algorithm..................................... 39

6. Heuristic.. 46
- 6.1 Implementation of the Heuristic... 46
  - 6.1.1 Creating Initial Population... 48
  - 6.1.2 Neighborhood.. 49
  - 6.1.3 Performing Neighboring Moves.. 51
  - 6.1.4 Speeding up the Evaluation Procedure.................................... 51
  - 6.1.5 Selection Procedure... 53

7. Results of the Heuristic... 55

8. **Conclusions and Future Research……………………………………………………… 61**

Bibliography…………………………………………………………………….. 64

Appendix…………………………………………………………………………. 67

List of Figures

2.1 Efficient set and nondominated set………………………………………………... 6
2.2 Nondominated set, Ideal and Nadir point…………………………………………. 7
2.3 Relation between objective function value vectors………………………………... 8
2.4 A strictly dominates B……………………………………………………………….. 9
2.5 A dominates B……………………………………………………………………….. 10
2.6 A weakly dominates B……………………………………………………………….. 10

3.1 Illustration of a) convex nondominated set and b) non-convex nondominated set… 14
3.2 Illustration of the ε-Constraint method……………………………………………. 15
3.3 MOGA………………………………………………………………………………. 22
3.4 NSGA……………………………………………………………………………….. 22
3.5 SPEA2………………………………………………………………………………. 23

4.1 Body shop, Paint shop and Assembly shop……………………………………….. 25
4.2 Computing color changes…………………………………………………………. 26
4.3 Computing color changes and ratio violations……………………………………. 28

5.1 Nondominated set of instances 2, 6, 8 and 11…………………………………….. 41
5.2 Number of vehicles versus execution time………………………………………... 42
5.3 Number of nondominated points versus execution time………………………….. 43
5.4 Range width versus execution time………………………………………………... 44

6.1 Neighboring moves………………………………………………………………….. 50
6.2 Evaluating car sequence for the first time…………………………………………. 52
6.3 Evaluating a car sequence after applying a neighboring move…………………….... 52
6.4 Scheme of the selection procedure………………………………………………... 54

7.1 Best solutions found on instance 6, a) in the initial population,
b) at iteration 1500, c) at iteration 5000 and d) iteration 30000…………………..... 57
7.2 Best solutions found on instance 12, a) in initial population
b) at iteration 7000, and c)at iteration 50000……………………………………… 58
7.3 Solutions generated by the heuristic and ε-constraint algorithm
on a CSP instance with 70 vehicles……………………………………………….. 59
7.4 Instance 13………………………………………………………………………….. 59
7.5 Instance 14………………………………………………………………………….. 59

List of Tables

5.1 Features of the CSP instances………………………………………………….. 39
5.2 Results of the ε-constraint……………………………………………………… 40
5.3 Comparing utilization rates…………………………………………………….. 42
5.4 Objectives range width………………………………………………………… 43
5.5 Results of the ε-constraint with different Δε………………………………………. 44

7.1 Results of the heuristic and ε-constraint…………………………………………... 56
7.2 Number of optimal solutions in the initial population…………………………….. 56
7.3 Execution time of instances 13 and 14……………………………………………. 60

List of Abbreviations

CSP	-	Car Sequencing Problem
CC	-	Color Changes
RC	-	Ratio Constraints
HPRC	-	High Priority Ratio Constraint
LPRC	-	Low Priority Ratio Constraint
COP	-	Combinatorial Optimization Problem
MCOP	-	Multiobjective Combinatorial Optimization Problem
TS	-	Tabu Search
SA	-	Simulated Annealing
EA	-	Evolutionary Algorithm
MEA	-	Multiobjective Evolutionary Algorithm
GA	-	Genetic Algorithm
MOGA	-	Multi-Objective Genetic Algorithm
NSGA	-	Non-dominated Sorting Genetic Algorithm
SPEA	-	Strength Pareto Evolutionary Algorithm

1. Introduction

The increasing global competition in the automobile market is changing the industry environment, creating new and more difficult challenges to car manufacturers. In order to secure their market position, car manufacturers are not just concerned about improving their vehicles performance or adding more features to it, they are also interested in optimizing the production processes, in order to reduce operation costs. Most European automotive companies like Renault (see Nguyen *et. al.* [2005]), rely on a build-to-order production system, instead of a build-to-inventory. This production approach tends to generate a diversified flow of vehicles in assembly plants, which means a wide range of different vehicles and a great variety of car sequences from day to day, due to the fact that cars are ordered by customers and not by dealerships. If these vehicles are not scheduled in an appropriate way, the working bays along the assembly line, where components are installed, might become overconstrained, which means extra work for employees. Workers with too much work get tired and make mistakes. These represent extra costs for car companies. Taking this into consideration the goal is to schedule vehicles along the assembly line, in order to reduce costs while satisfying as well as possible the capacity constraints (assembly line constraints to be more precise), caused by resources limitations. This is the industrial context in which the Car Sequencing Problem (CSP) is based on.

A solution for this problem consists of a permutation of the vehicles to be scheduled in the production day. Therefore we easily identify the Car Sequencing Problem as being a typical Combinatorial Optimization Problem (COP). The distinctive features of a COP are the finite number of solutions and the fact that each solution has a combinatorial property, such as an arrangement or a permutation of objects (see Papadimitriou and Steiglitz [1982]).

This decision problem, known by the Car Sequencing Problem, has been introduced for the first time by Parello *et. al.* in 1986 (see Parello *et. al.* [1986]). In this first version the capacity constraints were imposed solely by the assembly line. More recently, in 2003, the French Operations Research Society, jointly with Renault, proposed a challenge called ROADEF Challenge 2005, with a multiobjective version of the Car Sequencing Problem as subject. The CSP version presented in the challenge is more complex than the original one since, besides the assembly line constraints other constraints related with the paint workshop were added. Furthermore the objective function is composed of three terms that correspond to the three objectives to be minimized: number of Color Changes (CC), number of violations of High Priority Ratio Constraints (HPRC) and number of violations of Low Priority Ratio Constraints (LPRC). In terms of computational complexity, Gent [1998] has shown that the problem is NP-complete, that is, there is no known algorithm that solves the problem in a polynomial time.

In this thesis we aim to solve the Car Sequencing Problem version proposed in the ROADEF challenge. However, for simplification purposes, instead of three objectives (CC, HPRC and LPRC) we optimize just two objectives, by joining HPRC and LPRC in one single objective called violations of Ratio Constraint (RC). In other words, the objectives to optimize are: CC and RC. Furthermore, in this thesis we use a different notion of optimality called Pareto optimality. This notion of optimality is usually applied when there is no information, a priori, about the decision maker preferences. The idea of Pareto optimality is to find the efficient (or Pareto optimal) solutions that cannot be improved in one objective without deteriorating their performance in at least one of the others. The set of all efficient solutions is called efficient set and the image of it, in the objective space, is called nondominated set or Pareto front.

One goal of this thesis is to develop an exact algorithm, based on ε-Constraint approach that finds the nondominated set and the respective efficient solutions for every instance of the CSP. Unfortunately, this exact approach is only able to solve small instance in a reasonable amount of time. For instance, this approach may take several days to solve instances of size larger than 60. Moreover, as the problem becomes more constrained, the exact approach tends to take an infeasible amount of time even for small instances. As an alternative to the exact approach, we propose an heuristic based on local search procedures and evolutionary algorithms to solve this problem. Our experimental results indicate 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.

Thesis Overview

In the next chapter we introduce the definition of Multiobjective Combinatorial Optimization Problem (MCOP) as well as some important concepts related with Pareto optimality. Furthermore in Chapter 3, we present some of the most well known exact and approximate algorithms used to solve MCOPs. In Chapter 4, we give a detailed description of the Car Sequencing Problem presented in the ROADEF'2005 Challenge. In the same chapter we introduce an Integer Linear Programming (ILP) model which will be used by the ε-Constraint algorithm to find the various nondominated points. In Chapter 5, we introduce the ε-Constraint algorithm which runs several times the ILP morel until have found all the nondominated points. In addition we present some results of this algorithm on the instances tested. In Chapter 6, we propose an innovative heuristic to tackle the CSP. Then, in Chapter 7, we analyze the results provided by this heuristic approach and compare them with the results of the exact algorithm presented before. Finally, in Chapter 8, we present the conclusions of this thesis and discuss future developments.

2. Multiobjective Combinatorial Optimization

The purpose of this chapter is to introduce and explain some fundamental concepts related to Multiobjective Combinatorial Optimization Problem. Firstly, we introduce some notions about Combinatorial Optimization Problems. Further, we approach the same problems, but from a multiobjective perspective. In addition we cover some fundamental concepts related with multiobjective optimization, such as, efficiency and nondominance. Finally we discuss several notions of optimality that can be used in Multiobjective Combinatorial Optimization, with special emphasis on Pareto optimality, which is the notion of optimality adopted for this thesis.

2.1. Combinatorial Optimization

Combinatorial Optimization Problems (COP) are discrete problems, in which we are concerned with the efficient allocation of limited resources in order to meet desired objectives and where the values of some or all of the variables are integer, due to resources indivisibility (see Ehrgott and Gandibleux [2004]). Combinatorial optimization models are often referred as *integer programming* models where programming refers to "planning" so these are models used in planning where some or all the decisions can take only a finite number of alternative possibilities.

The purpose of Combinatorial Optimization is to find one or more optimal solutions, according to a given objective function, in a well-defined discrete problem space. The word combinatorial means that the set of feasible solutions is finite and any solution has some combinatorial property, such as a permutation (which is the case of the Car Sequencing Problem), an arrangement or partition of objects, a subset which corresponds to a selection of a set of objects, etc.. According to Garey and Johnson [1979], a COP can be formally described as:

- a set of *instances*;
- for each instance, a finite set S of *feasible solutions*;
- and an *objective function f* which assigns to each instance and to each feasible solution, an *objective function value*.

The purpose of any optimization process is to find, for each instance, a feasible solution, called *global optimum*, that minimizes (or maximizes) the objective function. Formally speaking, when the objective is to be minimized, a solution $s \in S$ is a global optimum if and only if there is no other solution $s' \in S$ such that $f(s') < f(s)$.

2.2. Multiobjective Combinatorial Optimization

In real-life problems there are usually more than one objective to be met. Multiobjective Combinatorial Optimization Problems (MCOPs), is the common designation for combinatorial problems with more than one objective to optimize. In MCOP we are interested in finding a feasible solution that minimizes (maximizes) a certain objective function vector according to some notion of optimality. Like in COPs, the feasible solutions of a MCOP have some combinatorial property and the number of feasible solutions is finite.

2.2.1. Problems

Many real-life applications have been formulated as combinatorial optimization problems. However these models often neglected the fact that most real-life problems require taking into account several conflicting criteria corresponding to multiple objectives. For instance:

- In the well known Traveling Salesman Problem, we might not only be interested in minimizing the total distance between cities, but also the traveling time, total cost, and so forth (see Paquete [2005]).
- In airline operations, scheduling technical and cabin crew has a major effect on cost and small percentage improvements may translate to multi-million dollar savings. However, cost is not the only concern in airline operations. Robust solutions are desired, to avoid the propagation of delays due to crew changing aircraft (see Ehrgott and Ryan [2002]).

Since, most MCOP leads with different conflicting objectives, it is crucial to know what kind of solutions we are looking for. In other words, how do we define an optimal solution? The answer for this question is on the definition of two fundamental concepts: *efficiency* and *nondominance*.

A general Multiobjective Optimization Problem can be defined as follows

$$\min_{s \in S}(f_1(s), \ldots, f_P(s)), \tag{2.1}$$

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 objective space. A feasible solution $s \in S$ is called *efficient* if there is no other solution $s' \in S$ such that $f_i(s') \leq f_i(s)$ for every $i = 1, 2, \dots, p$ and $f_j(s') < f_j(s)$ for some j. If s is a efficient solution than $f(s)$ is called a *nondominated* point. The set of all efficient solutions $s \epsilon S$ is called *efficient set* and the set of all nondominated points $f(s) \in Y$ is called *nondominated set* or *Pareto front*. Efficiency refers to solutions s in the decision space while nondominance is used for objective vectors $f(s) \in \mathbb{R}^P$in the objective space. If *s* is efficient then $f(s) = (f_1(s), \dots, f_P(s))$ is called nondominated (see Figure 2.1).

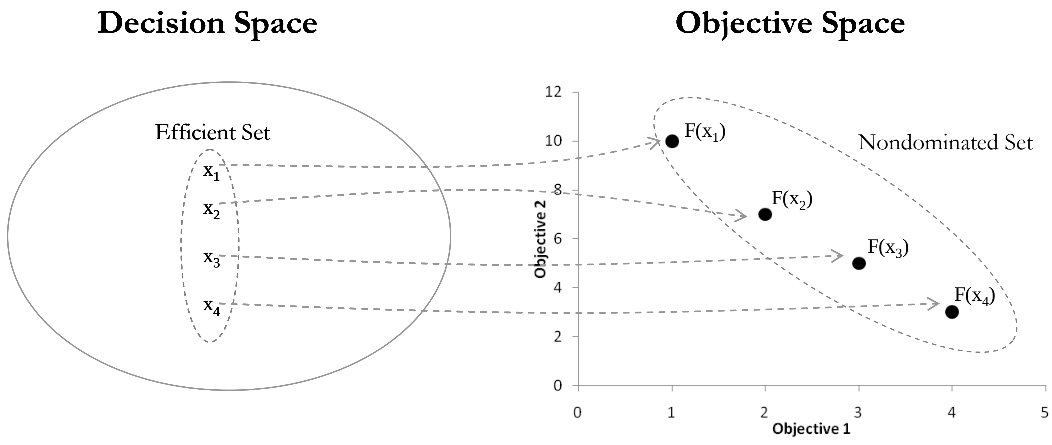


Figure 2.1: Efficient set and nondominated set

Two other important concepts in multiobjective optimization are Ideal and Nadir points. The Ideal point represents the best of both objectives while the Nadir point represents the worst of both objectives. These points are used as references to define the lower and upper bound of the nondominated set. The knowledge of these bounds gives us an estimation of the range of the values which nondominated points can assume. In Figure 2.2 we see a representation of a nondominated set, and the respective Ideal and Nadir points.

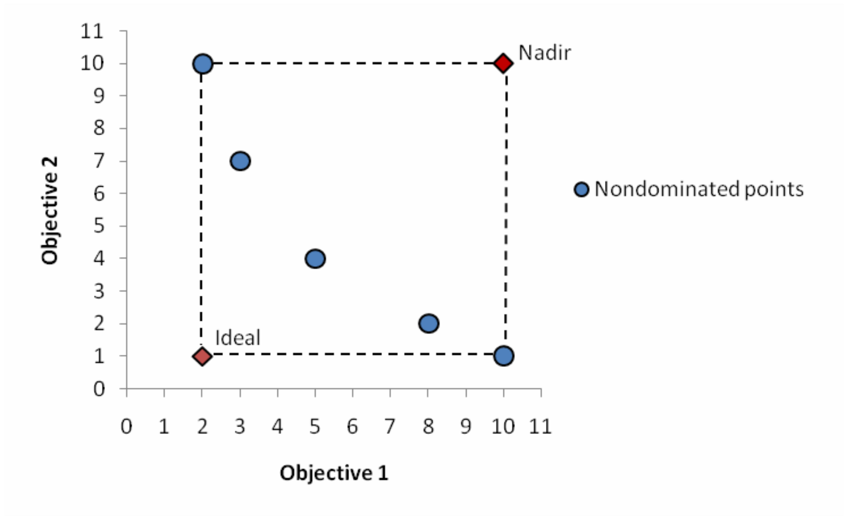


Figure 2.2: Nondominated set, Ideal, and Nadir point.

MCOP has received much attention from operational research community, in the last decade. Multiple objective optimization differs from traditional single objective counterpart in several ways, as mentioned in Ehrgott and Gandibleux [2004]:

- The usual meaning of the optimum makes no sense in the multiple objective case because does not exist, in most of the cases, a solution optimizing all objectives simultaneously. Instead, we search for feasible solutions yielding the best compromise among objectives that constitutes a so-called efficient solution set.

- The identification of a best compromise solution requires the preferences expressed by the decision maker to be taken into account.

- Multiple objective real-life problems can be expressed as mathematical functions of different forms. This means that we do not only deal with conflicting objectives, but with objectives of different structures.

- 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]).

2.2.2. Pareto Optimality

In many real-life decision problems it is not possible to know in advance the relevance of each objective. Even when the decision maker has this information, some times he does not know what weights he should assign to the objectives. In such situations the goal of solving MCOPs from a

Pareto optimality perspective is to find solutions (Pareto optimal) that can not be improved in one objective without deteriorating their performance in at least one of the others. The knowledge of all these optimal solutions allow the decision maker to be more conscious of the trade-offs among the different objectives while taking their decisions. As we said before a feasible solution $s \in S$ is considered an efficient or Pareto optimal solution if there is no other feasible solution $s' \in S$ such that $f_i(s') \leq f_i(s)$ for every $i = 1, 2, \dots, p$ and $f_j(s') < f_j(s)$ for some j. In other words, no solution is at least as good as *s* for all criteria, and strictly better for at least one.

We define the relation between objective function value vectors of two feasible solutions *s* and *s'*, in terms of dominance, as follows:

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

For better understanding these relations, we provide the following example: observing solutions in Figure 2.3 we conclude that solution *a dominates a'*, solution *b strictly dominates b'*, and solution *c weakly dominates c'*.

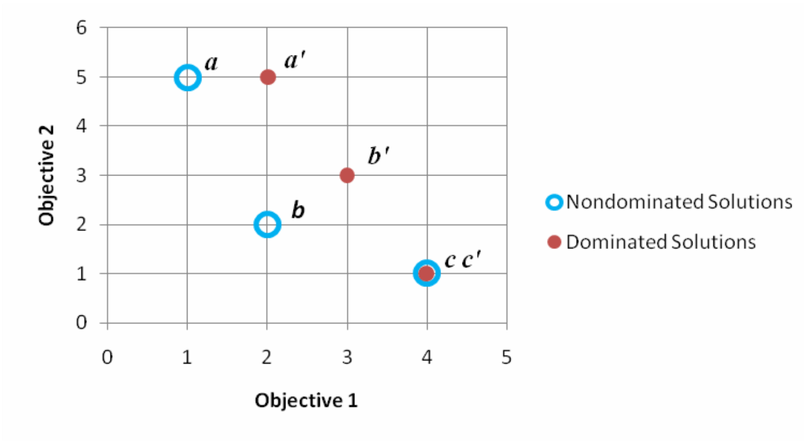


Figure 2.3: Relation between objective function value vectors

As we said in the previous section, the notion of optimal solution is very different from the single-objective counterpart. Therefore we introduce the following definitions of *Pareto global optimum solution* and *Pareto global optimum set*, stated in Ehrgott [2005]:

Pareto global optimum solution: A solution $s \in S$ is a Pareto global optimum if and only if there is no $s' \in S$ such that $f(s') \prec f(s)$.

Pareto global optimum set: $S' \subseteq S$ is a Pareto global optimum set if and only if it contains *only* and all Pareto global optimum solutions.

This Pareto global optimum set is particularly difficult to find (see Ehrgott [2005]). Therefore, it could be preferable to have an approximation to that set in a reasonable amount of time, where the objective function value vectors returned should be nondominated. Hansen and Jaszkiewick [1998], and later complemented by Zitzler *et. al.* [2003], introduced outperformance relations between sets of non-dominated objective function value vectors. Here we review some of these notations. Given two sets of objective function value vectors, $A = \{a^1, \ldots, a^m\}$ (represented by blue circles) and $B = \{b^1, \ldots, b^n\}$ (represented by red dots), we use the following relations:

- $A < B$ if every $b \in B$ is strictly dominated by at least one $a \in A$, i.e., A *strictly dominates* B (see Figure 2.4).
- $A \prec B$ if every $b \in B$ is dominated by at least one $a \in A$, i.e., A *dominates* B (see Figure 2.5).
- $A \leq B$ if every $b \in B$ is weakly dominated by at least one $a \in A$, i.e., A *weakly dominates* B (see Figure 2.6).

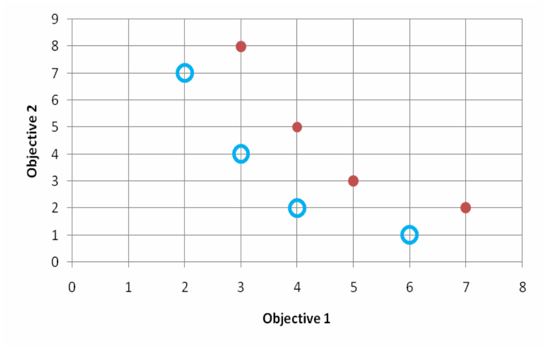


Figure 2.4: A *strictly dominates* B

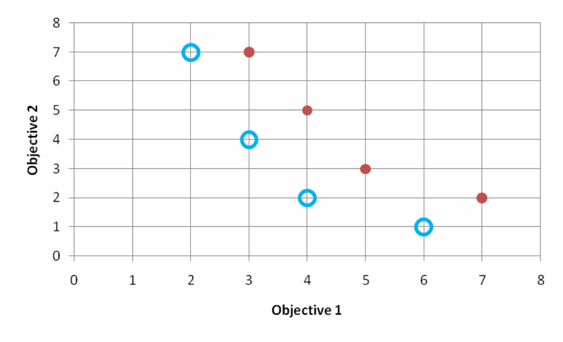


Figure 2.5: *A dominates B*

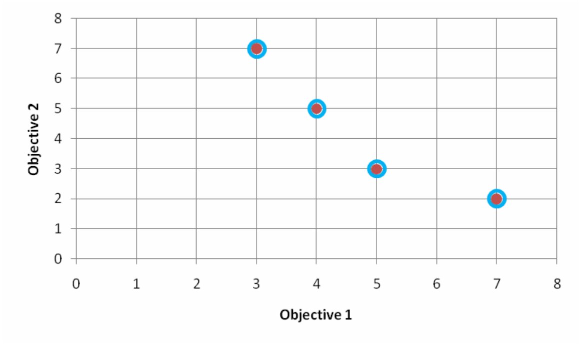


Figure 2.6: *A weakly dominates B*

2.2.3. Other Notions of Optimality

Besides Pareto Optimality there are two other notions of optimality that are commonly used, which are Lexicographic Optimality and Scalarized Optimality. Contrarily to the first one in these last two notions the decision maker is able to express, in some way, the relevance of the various objectives. Next we present the basic ideas behind these notions.

Lexicographic Optimality: This kind of optimality is commonly used when the decision maker is able to rank the objectives according to their lexicographic order, i.e., their relevance. An important feature of this notion is that no weights are assigned to the objectives.

The most common technique used to find the lexicographic solutions is solving each objective sequentially by decreasing order of importance and using optimal solutions of higher importance objectives as constraint values. This methodology is described in more detail in Chapter 5.

Scalarized Optimality: This notion of optimality is used when the decision maker is able to quantify, through weights, the relevance of each objective. The objective function vector is scalarized according to some weight vector. Due to the fact that the objectives are measured in different scales the weighted vectors need to be normalized.

3. Solution Methods for Multiobjective Combinatorial Optimization Problems

There are many approaches that can be used for solving MCOPs, that is, to find the Pareto optimal solutions or at least an approximation to it. These algorithms can be categorized into two major classes of algorithms, the *exact* and the *approximate*. These last ones are also called *heuristics*. Exact algorithms are guaranteed to find an optimal solution and to prove its optimality for every instance of an MCOP. The run-time, however, often increases dramatically with the instance size, and often only small or moderately-sized instances can be solved in practice to provable optimality. In this case, the only possibility for larger instances is to trade optimality for run-time, yielding heuristic algorithms. In other words, the guarantee of finding optimal solutions is sacrificed for the sake of getting good solutions in a limited amount of time.

In this chapter we present the main working principles of these classes as well as their advantages and limitations. Furthermore, we give some examples of algorithms for each class, in order to better understand the ideas introduced before.

3.1. Exact Algorithms

Since its appearance in Second World War to its recent resurgence stimulated by the increasing computing performance, operational research has developed various exact techniques to solve some of the most difficult real-life problems. Among the most well known exact techniques are *branch-and-bound*, *Lagrangian relaxation* based methods, and linear and integer programming based methods, such as *branch-and-cut*, *branch-and-price* and *branch-and-cut-and-price*. These last three methods are an implementation of *branch and bound* in which linear programming is used to derive valid bounds during construction of the search tree. Furthermore problem-specific cutting planes are used to strengthen the linear programming relaxations used for bounding.

Although exact algorithms might seem the perfect method for solving COPs, however this is not true for many cases. This is due to the fact that for larger instances, exact algorithms take a large amount of time to solve it. In MCOPs defined in terms of Pareto optimality, the situation is even worse, since many problems for which the single objective version is solvable in polynomial time become NP-hard when one more objective is added (see Ehrgott [2000]).

There are two well known exact approaches for solving MCOPs from a Pareto optimality perspective: through the use of scalarized methods, and the ε-constraint method. In the following subsections we describe in more detail the principles of each method.

3.1.1 Scalarized Methods

The most widely used scalarized method for multiobjective optimization is the weighted sum method. This method combines multiple objectives into an aggregated scalar objective function by multiplying each objective function by a weighting factor and summing up all terms (see formula 3.1):

$$\min_{x \in X} \sum_{k=1}^{p} \lambda_k f_k(x) \tag{3.1}$$

The weighted sum uses a vector of weights $\lambda \in \mathbb{R}^p$ as a parameter. By varying these weights we can find all nondominated points, for problems with a *convex nondominated set.* We mean by *convex nondominated set*, a problem whose nondominated set has a convex shape, like in Figure 3.1 a). The main advantages of this method are its simplicity and efficiency. Furthermore solving a weighted sum is as hard as solving the corresponding single objective problem. However for those problems whose the nondominated set has not a convex shape, like in Figure 3.1 b), this approach is inadequate, since is not able to find *unsupported* solutions, i.e., solutions whose objective value vectors does not lies in the borders of the convex-hull. For instance, in Figure 3.1 b) the nondominated point (5, 6) corresponds to a Pareto optimal but does not lye in the border of the convex-hull. Therefore it is an unsupported solution, which means that the weighted sum algorithm will not able to find it. Besides this drawback, for some problems it is not clear which combination of weights should be used, in order to find some of the nondominated points that lye on the convex hull.

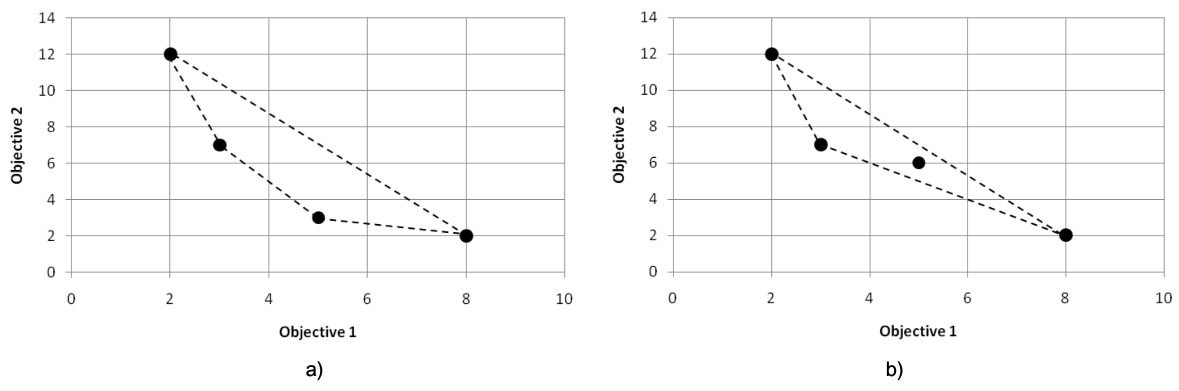


Figure 3.3: Illustration of a)convex nondominated set and b)non-convex nondominated set

Despite this drawback, the main principles are still applied to MCOPs, where the convexity of the efficient set in the objective space does not hold, for example, for generating supported solution that will be used for finding other solutions.

3.1.2 The ε-Constraint Method

The ε-Constraint is another well known technique used to solve MCOPs. In this approach, we optimize one of the objective functions using the other objective functions as constraints. Then by varying constantly the constraint bounds we can obtain all nondominated points. Figure 3.2 illustrates how this procedure works. The ε-constraint problem can be formulated as follows:

$$\min_{x \in X} f_j(x) \tag{3.2}$$

$$\text{subject to } \; f_k(x) \leq \varepsilon_k \quad k = 1, \dots, p \text{ and } \; k \neq j$$

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 ε_k. The working principle of the classical ε-Constraint is to iteratively decrease the constraint bound by a pre-defined constant $\Delta\varepsilon$. This constant is usually considered a major drawback of this methodology. Since only one solution can be found in each interval, these intervals should be as fine as possible to ensure that all nondominated points are found. However if the interval $\Delta\varepsilon$ is too small, the algorithm may spend a lot of redundant runs looking for nondominated points in some regions of the objective space where these ones do not exist. Another disadvantage of this method is that requires an initial lexicographic solution, otherwise this initial solution would be dominated by other more efficient solutions.

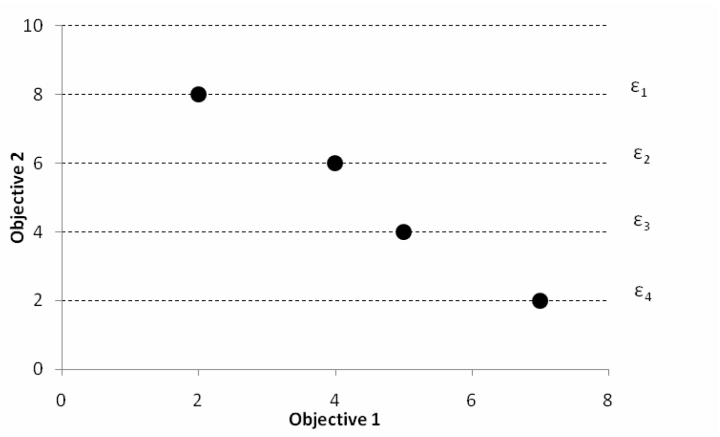


Figure 3.2.: Illustration of the ε-Constraint method

The ε-constrain method has several advantages over the weighted sum method, as shown by Chankong and Haimes [1983]:

1. In linear problems, the weighted sum is applied to the original feasible region and results to a corner solution (extreme solution), which means that generates only extreme efficient solutions. On the contrary, the ε-constraint method adjusts the original feasible region and is able to produce non-extreme efficient solutions. Consequently, with the weighted sum we can spend a lot of runs that are redundant in the sense that there can be a lot of combination of weights that result in the same efficient extreme solution. On the other hand, with the ε-constraint we can exploit almost every run to produce a different efficient solution, thus obtaining a more rich representation of the efficient set.

2. The weighted sum cannot generate *unsupported* efficient solutions in multiobjective problems where the shape of the nondominated set is *non-convex*, while the ε-constraint method does not have this inconvenience (see Steuer [1986] and Miettinen [1999]).

3. In the weighted sum the scaling of the objective functions has some impact in the results. Therefore, we need to scale the objective functions to a common scale before forming the weighted sum. In the e-constraint this is not necessary.

4. Another advantage of the ε-constraint method is that we can control the number of the generated efficient solutions by properly adjusting $\Delta\varepsilon$ in each one of the objective function ranges. This is not so easy with the weighted sum (see Erhgott [2000]).

The most notorious weakness of this method is the fact that when we add constraints to the single objective formulation, many of the COPs, which originally could be solved in polynomial time, become NP-hard (see Ehrgott [2005]).

3.2. Heuristics and Metaheuristics

The exact methodologies described in the previous chapter, among others, have already proved to be very useful for finding optimal solutions in many practical applications. However, in harder problems, with many objectives and large instances, these algorithms might not be able to solve it or when they do it they take too much time, which means, in some way, that they are not efficient. In such circumstances it is important to find a good feasible solution, which is at least approximate to an optimal solution.

A heuristic is a simple procedure that provides a good feasible solution in a reasonable computation time, but not necessarily an optimal one. In most of the cases we can not guarantee anything about the quality of the solution obtained. However a well designed heuristic method is able to provide a solution that is at least approximate to an optimal solution.

Nowadays, the most effective heuristics follow *local search* approaches. These heuristics start from an initial solution and iteratively try to substitute the current solution by a new improved solution in an appropriately defined neighborhood of the current solution. The idea is quite intuitive. We evaluate the neighbored solutions of the current solution, according to an objective function and if there is solution that is better than the current one, then we replace it. This process terminates as soon as no better neighbored solutions can be found anymore. This method is also called *iterative improvement*. Another particularity of heuristics is that they are not supposed to return the same outcome for different runs. This is due to the fact that they are based on a randomized search processes that use different random seeds for the random number generator.

Another well known kind of approach, is *constructive algorithms* that are used to generate good initial solutions for the subsequent application of local search algorithms. This kind of algorithms creates solutions from scratch by adding to an initial empty solution components in some order until a solution is complete. These methods are quite fast, however the quality of the solutions obtained are inferior to those generated by local search algorithms (see Hoos and Stützle [2004]).

Local search algorithms have been used since the early sixties, but only in the last two decades they have received more attention from the operational research community. A reason for that is the simplicity and intuitiveness of the concepts behind the algorithms, which make them easier to implement than the exact algorithms. Another reason is there ability to provide very good solutions when trying to solve large instances, thanks to the development of more sophisticated data structures.

Many of these algorithms are based on general concepts which mean that they can be applied to many different problems. These general search schemes are often called *metaheuristics*. They can be defined as an iterative generation process that uses local search procedures and other higher level strategies that enable a general heuristic to escape from local optimum solutions, in order to find near-optimal solutions (see Hoos and Stützle [2004]). Metaheuristics are designed to be general-purpose algorithms that can be applied to many problems with small modifications. Among the most well known metaheuristics are Tabu Search, Iterated Local Search, Variable Neighborhood Search, Simulated Annealing, Ant Colony Optimization and Evolutionary Algorithms. Curiously, some of these general-purpose schemes are inspired by natural processes, for instance the Evolutionary Algorithms and Ant Colony Optimization. As in many other scientific areas, nature has been an important source of inspiration for researchers. Next in this chapter we introduce very briefly the main concepts behind some of these metaheuristics.

3.2.1 Tabu Search

In 1986, Fred Glover proposed a new heuristic approach called Tabu Search (TS), in order to allow local search methods to overcome local optima (Glover [1987]). The basic idea of Tabu Search is to avoid local optima by allowing non-improving moves. Another distinctive feature of the Tabu Search is the use of (short-term and long-term) memory to save recent solutions or move attributes. The use of memory prevents the algorithm from choosing previously visited solutions.

As any other algorithm the TS tries, at each step, to make the best possible move from a current solution *s* to a neighboring solution *s'* even if it prejudices the objective value function. In order to prevent the process to immediately return to previously visited solutions and to avoid cycling, moves to recently visited solution are not allowed. This can be done by memorizing previously visited solutions, using some data structure, and forbidding moving to those. The number of iterations that reverse moves are forbidden should be carefully chosen, since it can influence the performance of the algorithm. For instance, if this number is too high the algorithm may become too restrictive and may miss important, unvisited solutions. To avoid such undesirable situation the algorithm, normally, has some criteria to override the tabu status of certain moves. The following scheme illustrates the main steps of a TS algorithm:

Tabu Search

```
BEGIN;
Initialize memory structures, Generate initial solution $s$ ;
    WHILE termination condition not met DO
        Generate neighboring solutions from the current solution $s$ ;
        Select the best neighboring solution $s'$ from the solutions generated previously;
        Update memory structures;
```

```
            **IF** solution $s'$ is better than the current solution $s$
                **THEN** $s = s'$;
    **end WHILE;**
**RETURN** $s$;
```

The efficiency of Tabu Search can be further improved by using methods that exploits the long-term memory of the search process. The purposes of these methods are to intensify and diversify the search. By intensification we mean exploiting promising regions of the search space either by recovering elite solutions (that is, the best solutions obtained so far) or attributes of these solutions. Diversification refers to exploring new search space regions corresponding to the introduction of new attribute combination (see Hoos and Stützle [2004]).

3.2.2 Simulated Annealing

Simulated Annealing is an iterative local search method inspired by a mechanical process used in the metallurgic industry called physical annealing. This process consists in melting a substance and then lowering the temperature very slowly in order to grow solid crystals with a perfect structure, which corresponds to a minimum energy state. Analogously, the set of solutions of a certain problem is associated with the energy states of the physical system, the objective function corresponds to the physical energy of the solid and the ground state corresponds to a global optimal solution (see Kirkpatrick [1983]).

The algorithm starts, by creating an initial solution, which may be a random or a already good solution. Then at each step a solution s' is generated and evaluated. A solution s' is always accepted if it improves the objective function value, otherwise it may be accepted with a probability which depends on the objective function difference $f(s) - f(s')$ and a parameter T, called temperature. In analogy with the real-life process this parameter T is lowered during the run of the algorithm and consequently the probability to accept worsening moves decreases. Formally, a move is accepted according to the following probability distribution:

$$p_{accept}(T, s, s') = \begin{cases} 1 & if\ f(s') < f(s) \\ e^{-\frac{f(s')-f(s)}{T}} & otherwise, \end{cases} \tag{3.3}$$

Initially, T is set to a high value and it decreases at each step according to some *annealing schedule* – which may be specified by the user but must end with $T = 0$ toward the end of the allotted time. See the generic SA algorithm outline presented next:

Simulated Annealing

```
BEGIN;
Generate initial solution s, initial value for T,
    WHILE termination condition not met DO
        WHILE number of iterations to be performed not met DO
            Generate neighboring solution s' from the current solution s ;
            Accept or reject solution s' according to equation 3.3;
            IF solution s' is better than the current solution s
              THEN s = s';
        end WHILE
        Update temperature T;
    end WHILE;
RETURN s ;
```

Simulated Annealing guarantees a convergence after running a sufficiently large number of iterations. However this theoretical result is not very helpful, since the annealing time required to ensure a significant probability of success will usually exceed the time required for a complete search of the decision space (see Hajek [1988]).

3.2.3 Multiobjective Evolutionary Algorithms (MEAs)

Evolutionary Algorithms, also called Genetic Algorithms (GA) have become very popular in the last ten years due mainly to the successful application of GAs to a wide range of combinatorial optimization problems. Evolutionary algorithms are inspired by the natural selection theory proclaimed by Charles Darwin. This theory says that individuals that are more adapted to a specific environment have more chances to survive.

The optimization techniques seen so far usually generates one single solution at each time, due to that fact they need to be executed several times in order to obtain all Pareto optimal solutions. In addition, the efficiency of these methods depends much on the discreteness of the search space among other features. On the other hand evolutionary algorithms use a population-based approach. By population we mean a set of individuals which represents feasible solutions. Each individual has a fitness value, which is assigned according to its dominance. This approach has shown to be very useful for solving multiobjective optimization problems, mainly because of their ability to find multiple optimal solutions in a single run. Such promising results lead the researchers to a new class of evolutionary algorithms called Multiobjective Evolutionary Algorithms (MEAs), especially design to tackle multiobjective optimization problems.

An evolutionary algorithm is composed of three main stages recombination, mutation and selection. In recombination stage, two individuals (parents), are chosen according to some criteria, and combined by some crossover operator to generate new individuals, called *offspring*. The idea of the crossover operator is to transmit the good features of the parent solutions to the offspring (each couple can generate more than one offspring). The role of this operator is determinant to search for good solutions in an efficient way. 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 complete cycle of reproduction, mutation and selection is called *generation*. All these operations are illustrated in the following scheme.

Genetic Algorithm

```
BEGIN;
Generate initial population p,
    WHILE termination condition not met DO
        Choose some solutions/elements (parents) according to some ranking procedure;
        Apply recombination operator to the parent solutions;
        Apply mutation to some solutions of the population;
        Selection of the best solutions according to some ranking procedure;
    end WHILE;
RETURN p ;
```

There are two major questions that must be analyzed while creating a MEA: how to assign the fitness to solutions and how to keep the diversification of solutions? In order to answer these questions some researchers developed different MEA approaches with different ranking procedures. A ranking procedure consists of assigning a single fitness value to each solution according to its position in the objective space. Afterwards, a set of solutions is chosen by some stochastic sampling procedure according to the solutions ranking. Usually the solutions that are closer to the Pareto front have lower fitness values and higher chances of being chosen. Next in this chapter we introduce some of the most well known MEA approaches, focusing in the selection procedure.

- Fonseca and Fleming [1993] suggests a quite simple approach called *Multiobjective Genetic algorithm* (MOGA) that assigns to each solution a fitness value that is equal to the number of solutions that dominates it in the archive plus one. According to these values, solutions are ranked, where ties result in ranks being averaged. Then a sampling algorithm chooses the next set of solutions to remain in the archive. Figure 3.3 shows this procedure, before averaging tied ranks.

- Srinivas and Deb [1994] proposed a MEA approach, called *Non-dominated Sorting Genetic Algorithm* (NSGA), which was based on a previous work done by Goldberg. The major difference of this method from others is the ranking process, which identifies nondominated solutions in the population at each generation, to form nondominated fronts based on the concept of nondominance. The first step of the ranking procedure is to identify the nondominated solutions in the current population. These solutions are assumed to compose the first non-dominated front, thus the same fitness value is assigned to all of them. In order to maintain diversity in the population, a sharing method is then applied. Furthermore, the solutions of the first front are ignored and the rest of the population is processed the same way to identify the solutions that constitute the second nondominated front. This process continues until the whole population is classified into nondominated fronts. An example of this procedure is shown in Figure 3.4.

  Some years later Deb *et al.,* [2002] introduced a new version of this algorithm designated NSGA-II with significant improvements concerning elitism, diversity among solutions and computational speed, with respect to the ranking procedure.

- Zitzler *et al.* [2001] presented the *Strength Pareto Evolutionary Algorithm 2* (SPEA2), an improved elitist multi-objective evolutionary algorithm that employs an enhanced fitness assignment strategy compared to its predecessor SPEA. The ranking procedure works as follows:
  - Each solution in the archive and in the set of current solutions, is assigned a value that corresponds to the number of current solutions that are dominated by it.
  - then for each solution from the current population and archive, its rank is given by the sum of the values computed for the solutions from both sets that weakly dominate it.

  This ranking procedure is illustrated in Figure 3.5, where circles represent the current solutions and the squares corresponds to the solutions in the archive. Solutions, whose rank is less than one, are added to the archive. Another particularity of this method is that the size of the archive is fixed, so if the number of solutions becomes larger than the archive size, solutions which are clustered in the objective space are removed (except the extreme solutions); on the other hand if the archive is not full, the best current solutions are added until the archive is full.

Note that the basic concepts of MEA presented here might differ slightly from other literature. The number of variations that have been suggested in the last years is enormous. That does not happen just because of the problems itself but also because of the interpretation given by researchers to some of these concepts. Probably there aren't two MEA identical. Many variations in population size, in initialization methods, in fitness definition, in selection and replacement strategies, in crossover and mutation are possible. Such flexibility makes of MEA one of the most promising techniques to tackle MCOPs.

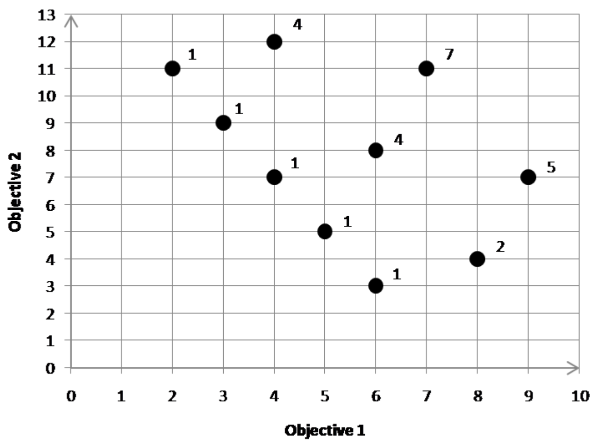


Figure 3.3: MOGA

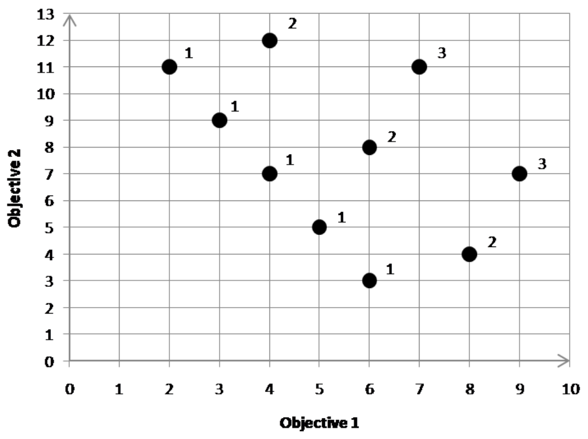


Figure 3.4: NSGA

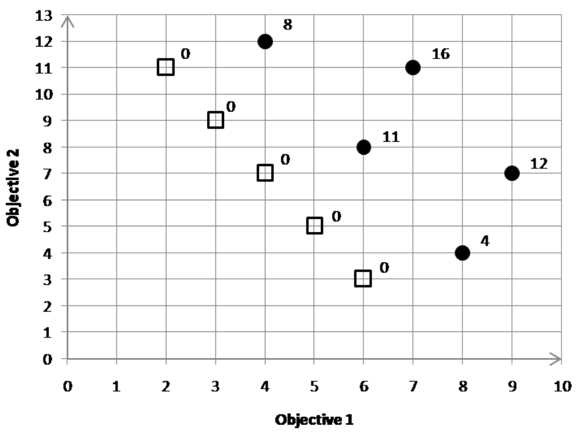


Figure 3.5: SPEA2

4. The Car Sequencing Problem

The Car Sequencing Problem (CSP) was introduced for the first time in 1986 by Parello [1986], and consists in scheduling a set of vehicles along an assembly line in order to install car components such as, air-conditioning, sun-roof, etc., while satisfying capacity constraints. This problem has been studied by many researchers, who developed a wide range of exact and heuristic approaches to solve this problem. In this chapter we give a detailed description of the Car Sequencing Problem version presented in the ROADEF'2005 Challenge as well as an integer linear programming formulation based on a previous work of Prandstetter [2005].

4.1 The ROADEF'2005 Car Sequencing Problem

The automotive industry has changed considerably in the last twenty years. For instance, nowadays the number of car options available and consequently the number of possible car configurations/versions is much larger than before. In addition, in order to reduce costs related to car stocks and, and at the same time, to maintain a high level of responsiveness to customers demand, most European automotive companies adopted a build-to-order production system, instead of a build-to-inventory system typical of American (see Nguyen *et. al.* [2005]). In conclusion all these changes, among others, have generated new needs, which were not taken into consideration by the classical CSP.

In 2003, the French Society of Operations Research and Decision Analysis organized a competition with the CSP as subject, called *ROADEF Challenge 2005*, which was proposed and sponsored by the automobile manufacturer Renault.

The problem proposed by Renault for the ROADEF'2005 Challenge differs from the classical problem discussed previously. Besides capacity constraints imposed by assembly shop, it also introduces paint batching constraints, and it considers two categories of capacity constraints to take into account their priority (see Nguyen *et. al.* [2005]).

In the following subsections we explain very briefly the typical tasks performed in a general car plant as well as the objectives that each plant shop aims to optimize.

4.1.1 Planning and Scheduling Process

The first task of each car factory is to assign a production day to each ordered vehicle, according to delivery dates and production line capacities. The set of vehicles which is determined at this stage cannot be changed in any circumstance. Then, each car factory has to schedule the order in which these vehicles will be put in the production line, while satisfying at best the requirements of the plant shops: *paint shop* and *assembly line*. Currently these two tasks are performed by a software that uses *Linear Programming* and *Simulated Annealing* for each task respectively (see Nguyen *et al.* [2005]).

For those who are not familiarized with automobile production plants, these are composed of three major areas: the *body shop*, the *paint shop* and the *assembly shop*. The body shop is where robots and operators weld metal panels together to form the vehicle structure. The paint shop is the place where cars are painted by robots with spray guns. Finally the assembly shop is divided into smaller work stations, also called working bays, where various processes take place and components or options are added to vehicles. Figure 4.1 illustrates the three major areas of a car production plant.

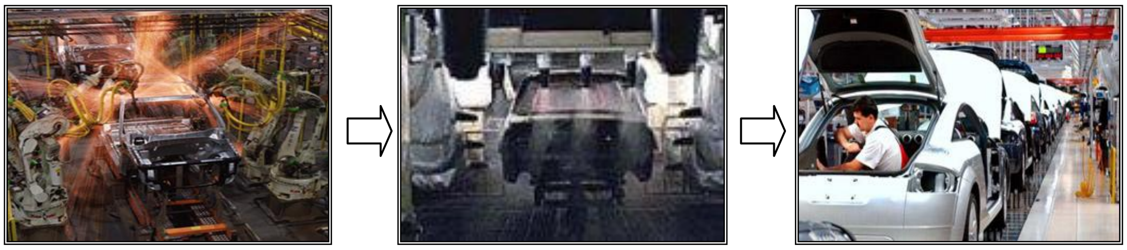


Figure 4.1: Body shop, Paint shop, and Assembly shop respectively.
Source: www.evworld.com

4.1.2 Paint Shop Requirements

The paint shop objective is to *minimize the number of color changes* in order to reduce the consumption of solvent used to wash the spray guns every time that there is a color change. This means that to minimize the consumption of paint solvent, there should be as few paint color changes as possible in the sequence. This can be achieved by grouping vehicles with the same color into batches and sequencing the batches in such way that minimizes the number of paint changes. The last vehicle from the previous production day has to be taken into account when evaluating the color changes, i.e., if the last vehicle produced in the previous production day has a different color from the first vehicle of the current day, then we should count one color change.

However, there is a maximum number of consecutive vehicles that are allowed to have the same color, because spray guns have to be cleaned periodically in order to ensure high quality painting results. This paint batch limit is a *hard constraint*, i.e., all feasible sequences/solutions have to satisfy this constraint. The vehicles from the previous production day are not taken into account for checking the paint batch limit. Considering the two sequences illustrated in Figure 4.2, we verify that both sequences have 2 color changes. However, if the paint batch limit was 3, sequence a) would not be considered feasible, because it has 4 consecutive cars with the same color.

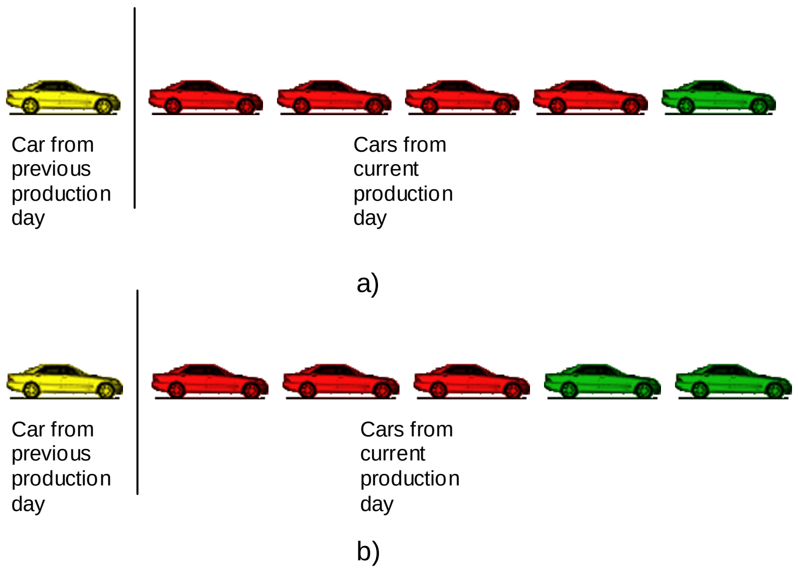

Car from previous production day
Cars from current production day
a)
Car from previous production day
Cars from current production day
b)

Figure 4.2: Computing color changes

4.1.3 Assembly Line Requirements

The objective in the assembly line is to *smooth the workload* along it, by balancing the effort in the different working stations. In other words, the vehicles that require special operations have to be uniformly distributed throughout the assembly line, to avoid overloading the stations where these vehicles are assembled. This need of space between constrained vehicles is modeled by a ratio constraint N_{cp}/Q_{cp}. Each component (or option) that requires extra operations has a ratio constraint associated to it. Given a ratio constraint N_{cp}/Q_{cp} there should not exist more than N_{cp} vehicles affected by a certain component cp in each consecutive sequence of Q_{cp} vehicles. For instances, if $N_{cp}/Q_{cp} = 1/4$, there must not be more than one constrained vehicle on any consecutive sequence of four vehicles. It means also that two constrained vehicles must be separated by at least Q_{cp} -1 vehicles, which corresponds to 3 vehicles in the previous example.

In the CSP version of the ROADEF, there are two classes of ratio constraints, *High Priority Ratio Constraints* (HPRC) and *Low Priority Ratio Constraints* (LPRC). The only difference between these two classes of ratios is the workload required to assemble the components. For instance, the components that involve more critical operations (HPRC) are more important than others that cause small inconvenience to production (LPRC).

In some cases there might be no sequence that satisfies all ratio constraints. In these situations we say that the problem is over-constrained. Therefore, the objective is to *minimize the number of violations of ratio constraints*. Due to the fact that ratio constraints can be violated, they are classified as Soft Constraints.

In order to obtain an evenly distribution of constrained vehicles we compute ratio constraint violations on gliding subsequences of the production day. The purpose is to space out vehicles as much as possible, because the lesser the space between constrained vehicles, the higher the number of violations will be.

The best way to explain how ratio constraint violations are computed is by giving an example. Lets consider a 1/3 ratio and the sequence of cars _X _ X X _ , where 'X' denotes a vehicle that requires the option and '_' denotes a vehicles that does not require the option. To evaluate this sequence we have to split the sequence into four subsequences of size 3, the first is _ X _, the second is X _ X, the third is _ X X and, finally the forth is X X _. Then we have to evaluate each of the subsequences according to the following formula:

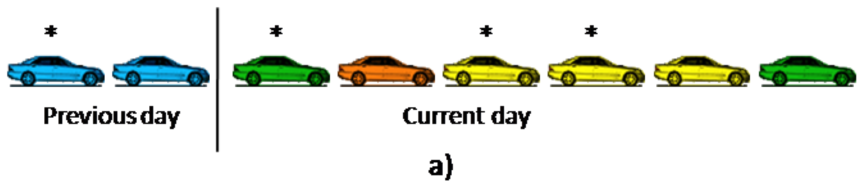
Number of violations on a subsequence[1] = (Number of vehicles associated with the ratio constraint on the subsequence) – (ratio constraint numerator)

By applying this formula to each of the subsequences, we verify that there are no violations on the first one but we have one for each of the other three. Hence, the whole sequence is evaluated to 3 violations.

In the previous objective (color changes) we had to take into account the last vehicle from the previous production day. Analogously, when we are computing the number of violations of ratio constraints on production day, we have to take into consideration the last Q_{cp}-1 vehicles from the previous production day. These ultimate vehicles are already scheduled, which means that their positions cannot be changed.

[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.

The following example illustrates how color changes and ratio constraint violations are computed. Assume the car sequence illustrated in Figure 4.3, where vehicles with an asterisk need a special option with a ratio constraint 1/3. Performing calculations as explained before we obtain the values stated in the tables bellow the car sequence. At the end, the total number of violations for sequence a) is 4. If the paint batch limit was four this sequence would be considered feasible.



Violations	1	0	1	1	1	0

Color Changes	1	1	1	0	0	1

Figure 4.3.: Computing color changes and ratio violations

As we mentioned before in this new multiobjective version of the CSP there are two classes of ratio constraints: HPRC violations and LPRC violations. However, for simplification purposes we assume that the HPRC and LPRC are incorporated in a single objective called Ratio Constraint (RC) violations. In other words we have now a bi-objective problem to optimize, with the following objectives: Color Changes and Ratio Constraint violations.

4.1.4 ROADEF Objective Function

Since there are different levels of priority between objectives, the organization of the event build a hierarchical objective function with three terms, representing the three objectives to optimize: the number of color changes, the number of HPRC violations and the number of LPRC violations. As we have seen before the HPRC has a higher priority than LPRC, which means that there are three possible objective hierarchies:

- Color Changes, HPRCs, LPRCs;
- HPRCs, LPRCs, Color Changes;
- HPRCs, Color Changes, LPRCs.

The ROADEF organizers, for evaluation purposes, decided to assign weights to the each type of constraint in the following way :

$$Z = 10^6 \times N^{o}\,Color\ Changes + 10^3 \times N^{o}\,HPRCs\ violations + 1 \times N^{o}\,LPRCs\ violations$$

Analyzing this formula carefully it is not very clear how the organizers calculated the weights. This is an important issue because this combination of weights has a great influence in the results obtained. Furthermore we do not understand why the objective functions are not normalized, since we are leading with different types of objectives. This lack of scientific accurateness rises serious doubts about the reliability of this aggregative objective function.

4.2 Solving the ROADEF'2005 CSP from a Pareto Perspective.

The goal of the ROADEF'2005 Challenge was to find a single solution that minimizes all the objectives according to some aggregated objective function. This notion of optimality adopted in the ROADEF Challenge, is called *scalarized optimality* and consists in weighting the objectives according to their relevance. This weighting procedure is done by assigning coefficients to each objective in the objective function.

We feel however that assigning weights to each objective is far more complicated than it seems. Defining the priority of each objective is not always an easy task. Moreover, transforming these priorities into weights it is far from being a straight forward job, since it requires some decision analysis skills. In this thesis we defined this problem in terms of Pareto optimality, assuming that no knowledge of the decision maker preferences is available a priori. From a Pareto perspective it is not necessary to make any kind of assumption about the objectives priority. Furthermore the objectives do not need to be normalized.

4.3 Integer Linear Programming Formulation

The Integer Linear Programming (ILP) formulation presented next, was based on various integer programming models developed by the participants Nouioua *et al.* [2005], Ribeiro *et al.* [2005] and Prandstetter [2005], with some minor adaptations to our specific circumstances. As it was mentioned in the last chapter our objectives as well as some of the assumptions are slightly different from those presented in ROADEF'2005 Challenge. The ILP formulation presented in this chapter was modeled through mixed integer programming (MIP).

Indices

The main part of any car factory plant is its production line, where each vehicle occupies one certain position i where several core operations are performed. In the paint shop each vehicle is painted with one single color cl, and then, in the assembly shop, components cp are added to vehicles. In order to reduce the number of variables, vehicles that share the same components and the same color are grouped into configuration classes k. One configuration k is a combination of one color and one or more components. However, some vehicles may not require special components cp. Furthermore, to compute color changes and ratio constraint violations it is necessary to take into consideration some vehicles m from the previous production day.

Taking these basic notions into account we can define the following indices:

$i \in \{1, \dots, NPos\}$,	where $NPos$ is the number of positions to fulfill, that is equivalent to the number of cars to schedule;
$m \in \{1, \dots, NCrP\}$,	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
$cp \in \{1, \dots, NCp\}$,	where NCp is the number of components;
$cl \in \{1, \dots, NCl\}$,	where, NCl is the number of colors;
$k \in \{1, \dots, NConf\}$,	where, $NConf$ is the number of configurations;

Input Parameters

Any instance of the car sequencing problem comprises various parameters that provides the necessary information to solve the problem, such as the demand δ_k for each configuration k, the color of each configuration $AL_{cl,k}$, the components that equip each configuration $AP_{cp,k}$, the color of the last vehicles from the previous production day $EL_{cl,m}$ as well as the components used by these ones $EP_{cp,m}$, the paint batch limit s and the ratio constraints attached to each special component N_{cp}/P_{cp} where $N_{cp} \leq P_{cp}$ are positive integers. Furthermore, using some of these parameters we can compute the amount of components cp required for each production day as well as for colors cl.

s	is the paint batch limit;
N_{cp}/Q_{cp}	is the ratio constraint for component cp;
δ_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;
$EL_{cl,m}$	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 $$d_{cl} = \sum_{k}^{NConf} \left(al_{cl,k} \cdot \delta_k\right) \qquad \forall cl \in \{1, \dots, NCl\}$$
d_{cp}	is the number of times that component cp is needed $$d_{cp} = \sum_{k}^{NConf} \left(ap_{cp,k} \cdot \delta_k\right) \qquad \forall cp \in \{1, \dots, NCp\}$$

Decision Variables

Note that each position i along the production line is occupied by a vehicle with one configuration k. Hence we create a binary matrix $P_{k,i}$ with $NConf \times NPos$ binary variables, where each variable $p_{k,i}$ is 1 if the car at position i has configuration k and 0 otherwise.

For counting the number of ratio constraints violations, we introduce matrix $R_{cp,i}$ with $NCp \times NPos$ positive variables $r_{cp,i}$, where each of these indicates the quantity of component cp used so far until position i. Further we use the values of these variables to calculate the actual number of violations for each subsequence, which will be stored in variables $g_{cp,i}$ of matrix $G_{cp,i}$.

To evaluate the number of color changes we use a binary matrix $B_{cl,i}$ with $NCl \times NPos$ binary variables $b_{cl,i}$ that specifies the color cl of the car that occupies position i in the production line.

This means that variable $b_{cl,i}$ is equal to 1 if car at position i is painted with color cl and 0 otherwise.

With this last matrix we are able to compute the number of color changes. For this purpose we introduce another binary matrix $W_{cl,i}$ with $NCl \times NPos$ binary variables $w_{cl,i}$ that are equal to 1 if a change to color cl occurs at position i, and 0 in all other cases.

The last variables are Z_1 and Z_2 which corresponds to the objective values.

Summarizing all 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;
$b_{cl,i}$	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;

Performance Criteria

The objectives that we want to minimize are the number of color changes as well as the total number of ratio constraint violations. The total number of violations is easily computed by summing all entries of matrix $G_{cp,i}$. For the color changes the process is analogous for matrix $W_{cl,i}$. Formally speaking, we have the following two objective functions (1) and (2):

$$Z_1 = \sum_{cl=1}^{NCl} \sum_{i=1}^{NPos} w_{cl,i} \quad (1) \qquad\qquad Z_2 = \sum_{cp=1}^{NCp} \sum_{i=1}^{NPos} g_{cp,i} \quad (2)$$

Constraints

First of all, we have to ensure that all vehicles that belong to the same configuration are assigned to a position and that a position is occupied by one and only one vehicle of a configuration class. These constraints are guaranteed by equation (3) and (4), respectively.

$$\sum_{i=1}^{NPos} p_{k,i} = \delta_k \qquad \forall k \in \{1, \dots, NConf\} \quad (3)$$

$$\sum_{k=1}^{NConf} p_{k,i} = 1 \qquad \forall i \in \{1, \dots, NPos\} \quad (4)$$

In addition the number of times that each component and color is used must be equal to the number of times that it is required, which is ensured by equation (5) and (6) respectively.

$$\sum_{i=1}^{NPos} \sum_{k=1}^{NConf} al_{cl,k} \cdot p_{k,i} = d_{cl} \qquad \forall cl \in \{1, \dots, NCl\} \quad (5)$$

$$\sum_{i=1}^{NPos} \sum_{k=1}^{NConf} ap_{cp,k} \cdot p_{k,i} = d_{cp} \qquad \forall cp \in \{1, \dots, NCp\} \quad (6)$$

For counting the number of ratio constraint violations, firstly we need to count the number of times that component cp is used until position i, which is done by equation (9) and (10). Inequation (8), ensures that $r_{cp,i}$ is a non-negative variable.

$$r_{cp,0} = 0 \qquad \forall cp \in \{1, \dots, NCp\} \quad (7)$$

$$r_{cp,i} \geq 0 \qquad \forall cp \in \{1, \dots, NCp\}, \forall i \in \{1, \dots, NPos\} \quad (8)$$

$$r_{cp,1} = \sum_{k=1}^{NConf} (ap_{cp,k} \cdot p_{k,1}) \qquad \forall cp \in \{1, \dots, NCp\} \quad (9)$$

$$r_{cp,i} = r_{cp,(i-1)} + \sum_{k=1}^{NConf} \left(ap_{cp,k} \cdot p_{k,i}\right) \qquad \forall cp \in \{1, \dots, NCp\}, \forall i \in \{2, \dots, NPos\} \quad (10)$$

The number $g_{cp,i}$ of violations on component cp for the window finishing at position i is done via the inequation (12) and (13). The first inequation (12) is used for those windows where it is necessary to take into consideration the vehicles from the previous day, and the second one for the remaining windows. Non-negativity has also to be ensure, which is done by inequation (11).

$$g_{cp,i} \geq 0 \qquad \forall cp \in \{1, \dots, NCp\}, \forall i \in \{1, \dots, NPos\} \quad (11)$$

$$g_{cp,i} \geq r_{cp,i} + \sum_{m=1}^{Q_{cp}-i} ep_{cp,m} - N_{cp} \qquad \forall cp \in \{1, \dots, NCp\}, \forall i \in \left\{1, \dots, Q_{cp} - 1\right\} \quad (12)$$

$$g_{cp,i} \geq r_{cp,i} - r_{cp,i-Q_{cp}} - N_{cp} \qquad \forall cp \in \{1, \dots, NCp\}, \forall i \in \left\{Q_{cp}, \dots, NPos\right\} \quad (13)$$

The constraints concerning color changes are defined in a similar way. First of all we have to calculate, through equation (14) the value for each binary variable $b_{cl,i}$, which assume the value 1 if car at position i is painted with color cl, and 0 otherwise.

$$b_{cl,i} = \sum_{k=1}^{NConf} (al_{cl,k} \cdot p_{k,i}) \qquad \forall cl \in \{1, \dots, NCl\}, \forall i \in \{1, \dots, NPos\} \quad (14)$$

Furthermore, inequation (15) guarantees that there are not more than s consecutive cars painted with the same color, which means that the paint batch limit constraint is fulfilled.

$$\sum_{n=i-s}^{i} b_{cl,i} \leq s \qquad \forall i \in \{s+1, \dots, NPos\} \quad (15)$$

Now, we are able to evaluate the number of colors changes. This can be easily done by comparing all pairs of consecutive vehicles. If the pair has the same color $w_{cl,i}$ is equal to 0, otherwise is 1. Inequations (16) and (17) ensures that color changes are correctly computed. The inequation (16) is just used for the first window, because it is necessary to consider the last vehicle from the previous day.

$$w_{cl,1} \geq b_{cl,1} - el_{cl,1} \qquad \forall cl \in \{1, \dots, NCl\} \quad (16)$$

$$w_{cl,i} \geq b_{cl,i} - b_{cl,i-1} \qquad \forall cl \in \{1, \dots, NCl\}, \forall i \in \{2, \dots, NPos\} \quad (17)$$

Summarizing all the formulations presented before we got 7 decision variables, 2 objective functions and 15 constraints.

Objective functions

$$min\ Z_1 = \sum_{cl=1}^{NCl} \sum_{i=1}^{NPos} w_{cl,i} \tag{1}$$

$$min\ \ Z_2 = \sum_{cp=1}^{NCp} \sum_{i=1}^{NPos} g_{cp,i} \tag{2}$$

Subject to

$$\sum_{i=1}^{NPos} p_{k,i} = \delta_k \qquad \forall k \in \{1, \dots, NConf\} \tag{3}$$

$$\sum_{k=1}^{NConf} p_{k,i} = 1 \qquad \forall i \in \{1, \dots, NPos\} \tag{4}$$

$$\sum_{i=1}^{NPos} \sum_{k=1}^{NConf} al_{cl,k} \cdot p_{k,i} = d_{cl} \qquad \forall c \in \{1, \dots, NCl\} \tag{5}$$

$$\sum_{i=1}^{NPos} \sum_{k=1}^{NConf} ap_{cp,k} \cdot p_{k,i} = d_{cp} \qquad \forall c \in \{1, \dots, NCp\} \tag{6}$$

$$r_{cp,0} = 0 \qquad \forall cp \in \{1, \dots, NCp\} \tag{7}$$

$$r_{cp,i} \geq 0 \qquad \forall cp \in \{1, \dots, NCp\}, \forall i \in \{1, \dots, NPos\} \tag{8}$$

$$r_{cp,\mathbf{1}} = \sum_{\boldsymbol{k=1}}^{NConf} (ap_{cp,k} \cdot p_{k,\mathbf{1}}) \qquad \forall cp \in \{1, \dots, NCp\} \tag{9}$$

$$r_{cp,i} = r_{cp,(i-\mathbf{1})} + \sum_{\boldsymbol{k=1}}^{NConf} \left(ap_{cp,k} \cdot p_{k,i}\right) \qquad \forall cp \in \{1, \dots, NCp\}, \forall i \in \{2, \dots, NPos\} \tag{10}$$

$$g_{cp,i} \geq 0 \qquad \forall cp \in \{1, \dots, NCp\}, \forall i \in \{1, \dots, NPos\} \tag{11}$$

$$g_{cp,i} \geq r_{cp,i} + \sum_{\boldsymbol{m=1}}^{Q_{cp}-i} ep_{cp,m} - N_{cp} \qquad \forall cp \in \{1, \dots, NCp\}, \forall i \in \left\{1, \dots, Q_{cp} - 1\right\} \tag{12}$$

$$g_{cp,i} \geq r_{cp,i} - r_{cp,i-Q_{cp}} - N_{cp} \qquad \forall cp \in \{1, \dots, NCp\}, \forall i \in \left\{Q_{cp}, \dots, NPos\right\} \tag{13}$$

$$b_{cl,i} = \sum_{k=1}^{NConf} (al_{cl,k} \cdot p_{k,i}) \qquad \forall cl \in \{1, \dots, NCl\}, \forall i \in \{1, \dots, NPos\} \quad (14)$$

$$\sum_{n=i-s}^{i} b_{cl,i} \leq s \qquad \forall i \in \{s+1, \dots, NPos\} \quad (15)$$

$$w_{cl,1} \geq b_{cl,1} - el_{cl,1} \qquad \forall cl \in \{1, \dots, NCl\} \quad (16)$$

$$w_{cl,i} \geq b_{cl,i} - b_{cl,i-1} \qquad \forall cl \in \{1, \dots, NCl\}, \forall i \in \{2, \dots, NPos\} \quad (17)$$

5. The ε-Constraint Algorithm

In Chapter 3 we introduced two well known exact methods to solve MCOPs, the weighted sum and the ε-constraint method. One goal of this thesis is to obtain the nondominated set for any instance of the CSP. To achieve this, firstly we need to find the Pareto optimal solutions whose objective value vector corresponds to the nondominated points. These Pareto optimal solutions can be supported or unsupported. Recalling from Chapter 3, unsupported solutions are those solutions whose objective value vector does not lie in the border of the convex hull. Since scalarized methods are not able to obtain these last ones, we decided to implement the ε-constraint method that is able to find both supported and unsupported solutions.

In this chapter we describe the main operations performed by the ε-constraint algorithm. Before describing the algorithm itself we introduce a simple procedure to find the lexicographic solutions. These solutions are essential to know the bounds of the nondominated set and therefore to restrict the search process. Then we discuss in more detail some implementation issues of the of ε-constraint algorithm. At the end we evaluate the performance of the algorithm on some CSP instances proposed by ourselves.

5.1. Lexicographic Optimization

In Chapter 3 we mentioned that one of the disadvantages of the ε-Constraint method was the need for determining the constraint value *a priori*. In order to obtain 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. For instance, in a bi-objective problem, assuming that objective 1 is more important than objective 2, we start by minimizing the first objective. After that, we define this objective as a constraint and minimize objective 2. This resulting solution is then used to set the initial value of the ε constraint. If, instead of this, we had just optimized objective 1 the resulting solution would be dominated by other more efficient one.

Before explaining how we solved the CSP from a lexicographic perspective, we introduce the following notation: the objectives, *number of color changes* and *number of ratio constraint violations* are denoted by Z_1 and Z_2, respectively. In relation to the lexicographic solutions, we denote the lexicographic solution with lexicographic order (Z_1, Z_2) by Lex(1, 2) and the other one by Lex(2, 1).

The procedure to find the lexicographic solutions is quite simple; the first step consists of choosing one objective to optimize according to some lexicographic order of the objectives. Let us assume for instance that objective 1 is more important than objective 2. We solve the ILP model (presented in the previous chapter) minimizing Z_1. Then the objective value optimized is fixed (it means that from now on Z_1 is a constraint). In the next iteration we solve the ILP model, but instead of minimizing objective 1, we minimize objective 2, without changing the value of objective 1. The idea is to improve one objective without deteriorating the other one. After these second run we got the objectives values of the lexicographic solution Lex(1, 2). For obtaining the lexicographic solution Lex(2, 1) we just have to exchange the lexicographic order of the objectives and repeat the whole process.

5.2. Implementation of the ε-Constraint

Recalling Chapter 3, the general ε-Constraint 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. Since only one solution in each virtual grid cell can be found, the only way to ensure that all the nondominated points are found is by defining a virtual grid as fine as possible. This can be done by setting $\Delta\varepsilon = 1$.

In our implementation we choose Z_1 as the objective to minimize while Z_2 works as a constraint. Since the lexicographic values were calculated previous we know the maximum and minimum value of each objective, which are particularly important to define the initial constraint value ε. Since we defined Z_2 as a constraint, the initial value of ε is equal to the maximum value of this objective in the nondominated set, which is equivalent to the maximum number of ratio constraint violations. The operations performed by the ε-Constraint algorithm in each iteration can be resumed as follows: 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. Finally, before terminating the iteration the constraint bound ε decreases by an amount of Δε. The algorithm repeats the process until it reaches the other lexicographic solution that corresponds to the minimum value of objective 2.

ε-Constraint

```
BEGIN;
NondominatedSet = {};
ε = second objective value function, given by solution Lex(1, 2);
Δε = 1
    WHILE not reach solution Lex(2, 1) DO
        Solve model minimizing $Z_1$;
        IF this new solution dominates the previous one
            THEN substitutes the dominated one by this new one;
        ELSE add new solution to NondominatedSet;
        Update constraint bound ε = ε – Δε;
    end FOR;
RETURN NondominatedSet;
```

5.3. Performance of the ε-Constraint Algorithm

The ILP model presented in the previous chapter was modeled through Mixed Integer Programming (MIP) and implemented in GAMS (General Algebraic Modeling System). For solving the model we used CPLEX 10.0. The CPLEX uses a sophisticated "branch-and-cut" algorithm which "splits" the major integer problem in a series of smaller linear programming subproblems.

For testing the performance of the ε-Constraint algorithm we developed various CSP instances with different features and different difficulty levels. In Table 5.1 we resume the main features of each CSP instance.

Instance	Number of vehicles	Number of colors	Number of components	Ratio Constraints	Paint batch limit
Instance 1	10	2	2	1/3; 2/3	4
Instance 2	15	3	3	1/3; 2/3; 1/3	4
Instance 3	20	4	4	1/3; 1/3; 2/3; 1/3	5

Instance 4	23	3	4	1/3; 1/2; 2/3; 2/4	7
Instance 5	23	4	4	1/3; 1/3; 2/4; 1/4	10
Instance 6	25	4	5	1/3; 1/3; 2/3; 1/2; 1/4	6
Instance 7	30	4	7	1/3; 1/4; 1/2; 1/3; 2/3; 1/4; 1/2	5
Instance 8	30	4	7	1/3; 1/4; 1/2; 1/3; 2/3; 1/4; 1/2	20
Instance 9	35	4	5	2/3; 1/3; 2/3; 1/2; 1/4	8
Instance 10	40	5	6	1/3; 1/3; 1/2; 1/3; 2/3; 2/4	10
Instance 11	40	5	6	1/3; 1/3; 1/2; 1/3; 2/3; 2/4	20
Instance 12	50	5	6	3/4; 2/5; 2/4; 1/4; 2/4; 1/3	15

Table 5.1: Features of the CSP instances

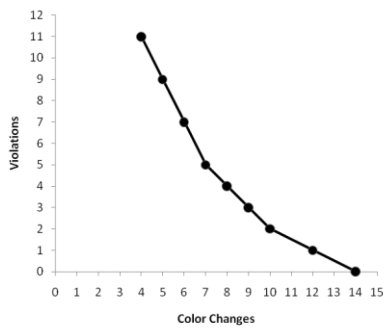
Note that both instances 4 and 5 have 23 vehicles but distribution of the various vehicles through the configuration classes is different. In addition instance 5 has lower ratio constraints and a higher paint batch limit. Finally, note that instances 10 and 11 are equal, except on the paint batch limit.

In the following table (Table 5.2) 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 $\Delta\varepsilon$. Furthermore, in Figure 5.1 we illustrate the nondominated set for some of these instances.

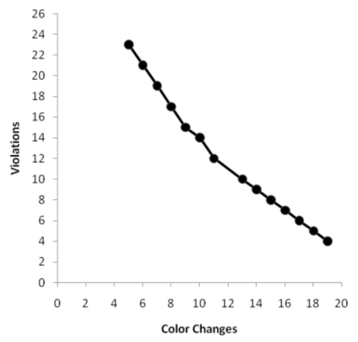
Instance	Execution time	Number of Nondominated Points	Bounds on the Nondominated Set (Ideal and Nadir Points)	$\Delta\varepsilon$
Instance 1	3sec	3	(3, 3) ; (6, 6)	1
Instance 2	1min 7sec	9	(4, 0) ; (14, 11)	1
Instance 3	2h 49min 6sec	10	(5, 9) ; (14, 22)	1
Instance 4	47sec	5	(3, 14) ; (9, 19)	1
Instance 5	1h 38min 24sec	10	(4, 9) ; (14, 22)	1
Instance 6	12h 57min 37sec	14	(5, 4) ; (19, 23)	1
Instance 7	5h 51min 12sec	10	(7, 14) ; (16, 27)	1
Instance 8	9h 46min 9sec	14	(4, 14) ; (16, 35)	1

Instance 9	14h 20min 32sec	11	(4, 1) ; (14, 20)	1
Instance 10	8h 21min 42sec	7	(6, 40) ; (12, 52)	1
Instance 11	9h 19min 22sec	8	(5, 40) ; (12, 53)	1
Instance 12	18h 33min 49sec	10	(6, 0) ; (16, 22)	1

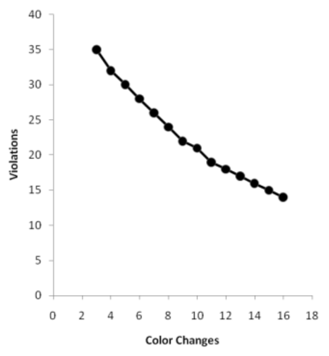
Table 5.2: Results of the ε-Constraint



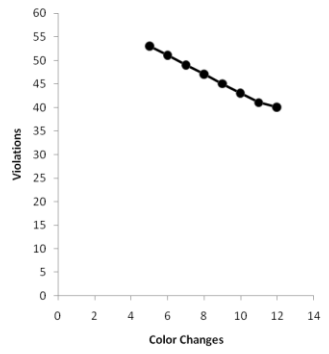
Instance 2



Instance 6



Instance 8



Instance 11

Figure 5.1: Nondominated set of instances 2, 6, 8 and 11

Analyzing the results shown in Table 5.2 we can take the following conclusion about the ε-Constraint algorithm performance:

- The epsilon-constraint method takes several hours to solve most of the instances (see instance 6, 9, 12). Only very simple instances with a small number of vehicles and few components can be solved in few minutes, like instance 1 and 2.

- By increasing the number of vehicles the problem becomes harder to solve, due to the enlargement of the decision (or search) space. This might not be very clear since we have several instances with a lower number of vehicles but that take much more time to be solved. However, if we compare the number of vehicles with the execution time, as shown in Figure 5.2 and calculate the correlation between these two variables we obtain a correlation coefficient of 0.82. This coefficient indicates that there is a direct correlation between these two variables which supports our initial statement.

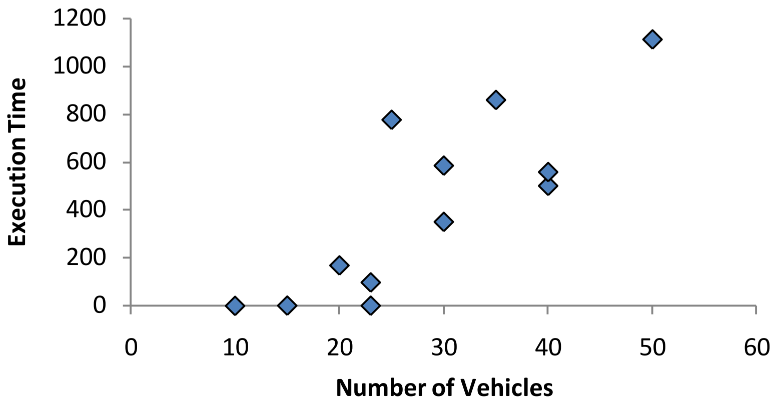


Figure 5.2: Number of vehicles versus execution time

- Instances 10 and 11 are equal to each other, but the last one has a higher paint batch limit and one more nondominated point. We take the same conclusion for instances 7 and 8. This observation leads us to conclude that less constrained instances tend to have more nondominated points and optimal solutions.

- There are several instances with a small number of vehicles that take much more time to solve than other ones with a larger number of vehicles. The reason for this has to do with the constraints, which depends on the ratio constraint of each component and the number of vehicles that requires each component. This relation can be expressed through a ratio, called utilization rate. Higher utilization rates more constrained the problem will be. 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 5.3. In conclusion, the difficulty level inherent to an instance depends on the size of the decision space but also on the constraints.

Utilization rate for component cp = (Number of vehicles requiring component cp / Max number of cars requiring component cp without violating ratio constraint)

Instance	Number of vehicles	Execution time	Utilization Rates	Paint batch limit
Instance 3	20	2h 49min 6sec	10/7; 6/7; 4/14; 5/7	5
Instance 4	23	47sec	13/8; 5/12; 6/16; 1/12	7

Table 5.3: Comparing utilization rates

Analyzing Table 5.3 we verified that the utilization rates in instance 3 are higher than in instance 4. In addition the paint batch limit is also smaller. All this features makes instance 3 more constrained than instance 4.

- Instances with a higher number of nondominated points seem to require more time to solve than others with a lower number of nondominated points. Computing the correlation coefficient between these two variables (see Figure 5.3) we obtained the value 0.56. This value means that there is a medium-high dependency between these two variables. In other words, a larger nondominated set tend to take more time to find than a smaller one.

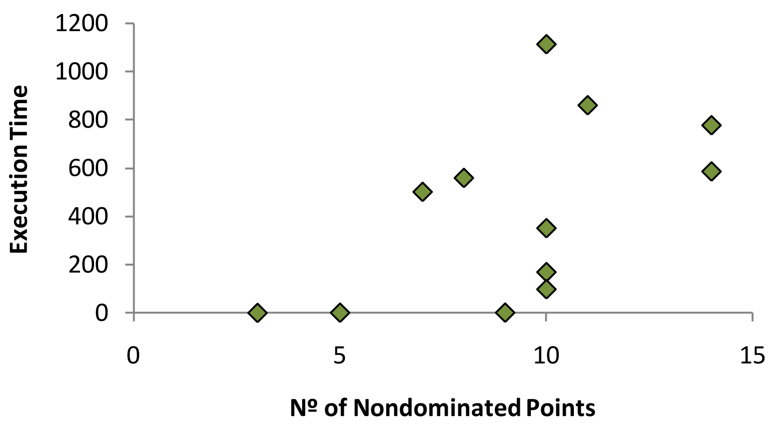


Figure 5.3: Number of nondominated points versus execution time

- We also observe, through the Ideal and Nadir points that the range of values of objective 2 tends to be wider than the range of values of objective 1 (see Table 5.4). If the objective that is used as constraint has a wide range of values it takes more time to be solved. To support this statement we calculate the correlation coefficient between the execution time and the width of the range of objective 2 (see Figure 5.4). The result was a correlation coefficient of 0.84, which can be interpretated as a very strong correlation between these two variables. Therefore, if we exchange the role of the objectives, i.e., using Z_2 as the only objective to minime and Z_1 as a constraint, it could reduce the execution time, since the number of redundant iterations would be lower.

Instance	1	2	3	4	5	6	7	8	9	10	11	12
Objective 1 Range Width	3	10	9	6	10	14	9	12	10	6	7	10
Objective 2 Range Width	3	11	13	5	13	19	13	21	19	12	13	22

Table 5.4: Objectives range width

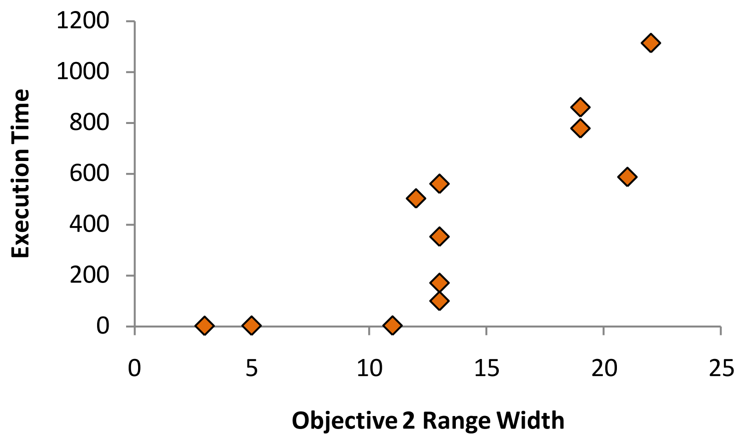


Figure 5.4: Range width versus execution time

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 overall execution time is to increase the pre-defined interval 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 these facts we tested some of the previous instances using different intervals Δε (see Table 5.5).

Instance	Execution Time	Number of Nondominated Points found	% of Nondominated Points found	% of the Execution Time (on the original time)	Δε
Instance 5	1h 00min 32sec	8	80%	61.3%	2
Instance 5	33min 42seg	6	60%	33.7%	3
Instance 5	25min 31seg	4	40%	25.6%	4
Instance 12	9h 21min 43 sec	5	50%	50.4%	2
Instance 12	6h 21min 48sec	5	50%	33.5%	3
Instance 12	5h 13min 35sec	3	30%	28.1%	4

Table 5.5: Results of the ε-Constraint with different Δε

Furthermore, we tried to solve harder instances, with a larger number of vehicles and more components. However, these instances require a huge amount of time to be solved. For instance, this algorithm may take several days to solve instances of size larger than 60. From an industrial perspective this amount of time is usually considered infeasible.

Due to these limitations exact methods may not be the best approach to be used in an industrial context, where the production capacity can easily reach 1000 vehicles. In order to handle larger and harder CSP instances we propose, an heuristic scheme that enable us to find, at least good approximate solutions. This does not mean that the results obtained before with the ε-constraint are completely useless. On the contrary they can be used as a benchmark, to evaluate the performance of the heuristic scheme.

6. Heuristic

During the ROADEF'2005 Challenge a wide range of general-purpose heuristic schemes, commonly called metaheuristics, were presented by the competitors. All these schemes were mostly based on local search methods integrated into high-level strategies which guide an underlying more specific heuristic, like Simulated Annealing (see Risler [2004]), Tabu Search (see Cordeau [2005]), Ant Colony Optimization (see Gravel *et. al.* [2005]) among others. Besides these classical methodologies, other innovative approaches, like the hybrid heuristic (combination of integer linear programming with heuristic techniques) proposed by Ribeiro *et. al.* [2007] and a very large neighborhood search developed by Nouioua *et. al.* [2005], proved to be excellent alternatives to tackle the CSP. Influenced by some of these ideas we propose an heuristic, based on local search methods and evolutionary algorithms, for solving the problem in terms of Pareto optimality. Furthermore, we describe in more detail some skillful strategies used to speed up the algorithm.

6.1 Implementation of the Heuristic

The algorithm presented here for solving the Car Sequencing Problem was inspired in some principles of the MEAs. As we said before, this class of algorithms uses a population-based approach capable of generating multiple solutions in a single run. Such features allow us to explore several regions of the decision space simultaneously. That is why this algorithm seems to us the most appropriate to find the approximate optimal solutions.

Recall from Chapter 2, Genetic Algorithms comprises three main stages: recombination, mutation and selection. The complete cycle of these stages corresponds to one generation. Our algorithm comprises just two of these stages, which are mutation and selection. There are two main reasons for not using the recombination stage, first of all it is not clear how can we transmit certain features from the parents solutions to the offsprings, specially in this particular multiobjective CSP, secondly this operator usually requires more computation effort than other more simple operators.

Generally, researchers concentrate all their efforts in the recombination stage relegating mutation to a second plan. This happens because recombination is usually considered the key operator to explore the decision space in an efficient way (Goldberg [1989]). In our case the situation is completely the opposite; we concentrate our attention in the mutation stage and simply ignore the recombination. Our idea is to show that mutation plays a much more important role in MEAs than most researchers believe. What we are trying to say is that the role of mutation, in MEAs, tends to be underestimated by programmers. One decade ago Bäck [1996] had already mentioned this fact. As you will see further in this thesis, the mutation operator presented here is capable of performing a quite fast exploitation of the decision space.

Usually the mutation operator performs random modifications in individuals without a well defined strategy, hoping that these modifications will improve the objective value functions. The problem of performing random moves is that the chance of success is very small, since they are completely arbitrary. The only way to increase the chance of success is by reducing the randomness and designing strategic moves based on some information about the neighborhood.

The heuristic introduced next in this chapter shows that it is possible to perform a reasonable efficient search of the decision space in a short period of time by using solely a mutation operator followed by a selection operator capable of preserving elitism and diversification among population individuals.

The following scheme resumes very briefly the main steps of our heuristic.

Heuristic

```
BEGIN;
Create initial Population;
Archive  //archive where the best solutions (sequences) are stored;
it  //number of iterations without updating the archive;
        WHILE time limit is not reached DO
                IF it >2000
                    THEN Restart Population;
                FOR each element (sequence) of the Population DO
                        Choose randomly two positions;
                        According to positions obtained apply mutation operation;
                        Evaluate new solutions;
                end FOR;
                Rank original and transformed solutions through MOGA procedure;
                Update current Population and Archive;
        end WHILE;
RETURN Best solutions in the Archive;
```

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 best solutions in the Archive.

To implement this algorithm some parameters have to be specified:

Population Size: Usually the population size is 10 for small instances. However for larger instances this number should be larger. Since the instances solved are relatively small, less than 60 vehicles, we decided to maintain the population size 10.

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 relatively small instances. However we are conscious that this value depends, in some way, from the instances features.

In the next subsections we discuss in more detail the most important steps of this algorithm, starting with the creation of the initial population and ending with the selection procedure. In addition we discuss some algorithmic aspects that enable us to speed up the solutions evaluation process.

6.1.1. Creating Initial Population

The initial population consists of a set of sequences that can be generated in two ways: the first one is by ordering vehicles randomly and the second one is by ordering vehicles according to some criteria, for instances through a constructive heuristic. We tested both methods and we verified that the first one is faster than the second one, but the second one performs better in respect to solutions quality. Has mentioned by Risler [2004] the method to generate initial sequences should be fast but it should provide good quality solutions in the first place. Therefore we decided to apply the second method.

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 both criteria, 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. In other words we vary the weight vector of the objective function from sequence to sequence. The sum of the weights must be equal to one. The following algorithm summarizes all the operations described above, about the insertion method.

Insertion method to create initial population

```
BEGIN
Population = {}
Sequence = {}
PopSize //population size
N //number of vehicles to schedule
W1=1, W2=0 ← weights initial values
    WHILE |Population| < PopSize DO
        WHILE |Sequence| < N DO
        v ← random vehicle not in Population;
            Choose position x where inserting v in Sequence leads to minimal increase
            of the evaluation function value;
            Insert vehicle v in position x of the Sequence;
        end WHILE;
        Population = (Population ∪ Sequence);
        W1=W1 - 1/PopSize and W2 = W2 + 1/PopSize;
    end WHILE;
RETURN Population;
```

6.1.2 Neighborhood

As we said in the beginning of this section the mutation is the operator responsible for the exploitation of the decision space. The mutation operator performs a local search within a specific neighborhood, which consists in a set of neighboring solutions for each sequence. The definition of the neighborhood can be very relevant for the performance of a stochastic local search algorithm because it influences the computation time required for the evaluation of the neighboring solutions. In other words, while taking a decision about what neighboring moves should be applied we should be concerned about the effectiveness of those moves (What are the chances of success?) but also with the evaluation speed of the new solutions that results from those moves (How fast can we compute the objective values of that sequence?). In conclusion the neighborhood should be effective and fast to evaluate. Taking these two principles into consideration we defined the following neighborhood:

- Exchange 1: exchanging two vehicles in the sequence, i.e., it consists in exchanging the positions of two vehicles, (see Figure 6.1 a));
- Exchange 2: exchanging two consecutive vehicles, (see Figure 6.1 b));
- Insert: moving one vehicle to a different position, i.e., we pick one vehicle from its original location, and then we move all the vehicles between the origin and the destiny location one position backward and reinsert the vehicle at the position that remains unfilled, see the example in (see Figure 6.1 c));

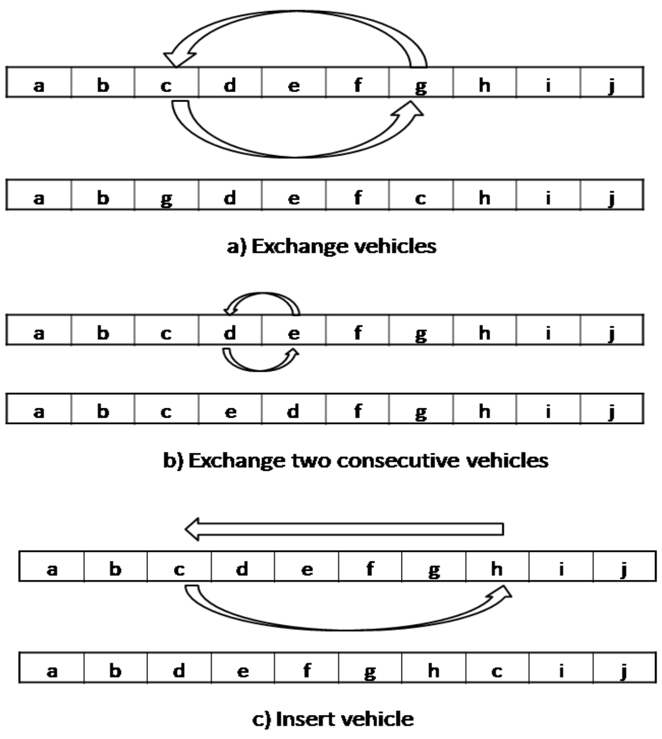


Figure 6.1: Neighboring moves

The neighborhood moves chosen are quite simple as well as fast to perform. Furthermore the degree of modification provoked by these movements is usually small, which means that the sequences do not suffer big changes, consequently their objective values vary in a smooth way.

To ensure the efficacy of this approach we have to think carefully in two major issues, firstly how to increase the chance of success of a certain move and, secondly how to increase the number of attempted movements. In the next subsections we describe some clever tips that contribute for improving the chances of success. Furthermore we explain how to accelerate the evaluation process.

6.1.3. Performing 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, which are stored in a special data structure, we choose the best move to apply.

This data structure has *N* entries, where *N* means the number of positions in the production line that are equivalent to the number of vehicles to schedule. In each of these entries we store the total number of violations that occur in the windows finishing in the position correspondent to that entry. See the data structure in Figure 6.2 and Figure 6.3.

According to vehicles features in the positions chosen and other information stored in the data structure we apply the following strategic moves.

- 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*. This movement might reduce the number of violations.
- 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.

From the various tests performed during the development of the algorithm, we verified that the exchange moves are the ones that most contributes for the generation of better solutions. In the ROADEF challenge some of the participants that used these neighboring moves, like Nouioua *et. al.* [2004], had also noticed this fact.

6.1.4. 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. Thanks to the special data structure introduced before we can evaluate very quickly the impact of neighboring moves on the objective values of the current solution. The idea is very simple: when we generate a sequence for the first time we have to evaluate the number of violations that occur in that sequence, by computing the number of violations for each window, as shown in Figure 6.2 (Ratio Constraint = 1/3).

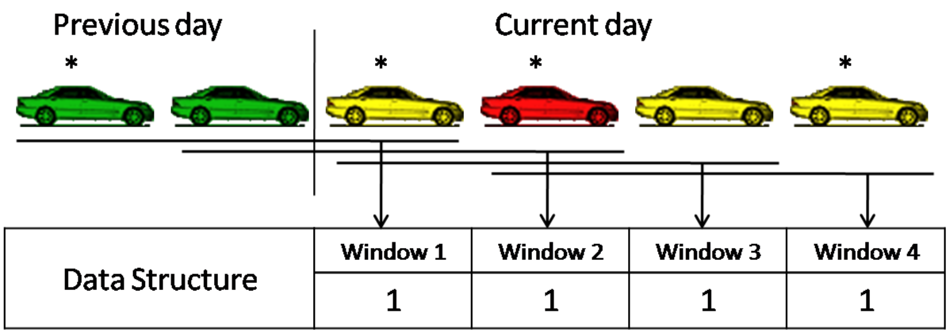


Figure 6.2: Evaluating car sequence for the first time

However when we perform a neighboring move we do not need to reevaluate all the windows, we just need to reevaluate those windows that are perturbed by the move, which depends only of the ratio constraint denominator. Taking the sequence in Figure 6.2 as an example, assume that we decide to exchange the last two vehicles. As a result only window 3 and window 4 are perturbed by this movement, which means that only these windows need to be updated. The rest of the windows remain the same, as you can see in Figure 6.3.

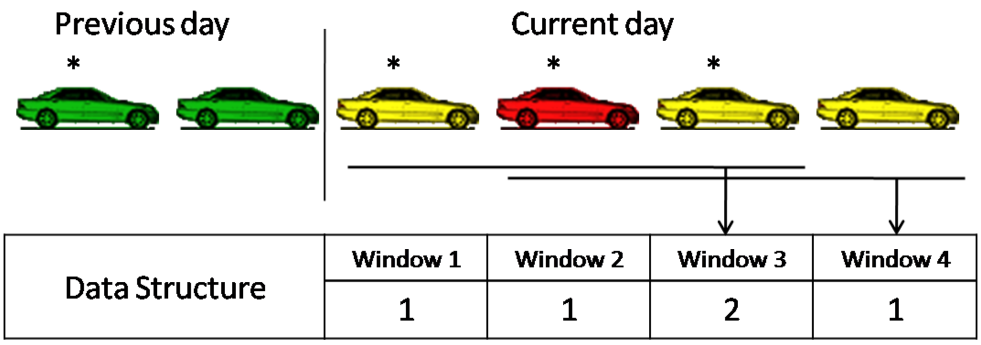


Figure 6.3: Evaluating car sequence after applying a neighboring move

For small instance like the one in the previous example, the time saved might not be substantial, but for larger instances the amount of time saved is enormous. Consequently more iterations are performed as well as attempted neighboring moves.

6.1.5. Selection Procedure

After performing the neighboring moves and evaluating the resulting solutions it is now time to select which of them will be part of the current population and which will be eliminated. The purpose of the selection procedure is to ensure elitism and diversity among the solutions that constitute the current population.

In Chapter 2 we introduced some multiobjective evolutionary algorithms with more emphasis in the selection procedure, like MOGA, SPEA-II, NSGA and NSGA-II. Among all these techniques we decide to choose the MOGA algorithm due to two main reasons: first of all is efficient and relatively easy to implement; in addition do not need so much computation as some other techniques.

As explained previously the approach 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. This means that the best solutions are those with lower fitness values. Then solutions are ranked according to those values. The better ones are then inserted in the current population and the worst solutions are removed. The whole process is resumed in Figure 6.4.

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. However during the program execution it will appear better solutions that dominate some solutions previously inserted in the archive. In order to maintain solely the best solutions, it is necessary to reevaluate periodically the solutions in the archive according to the MOGA approach. Then those solutions that are dominated are simply removed from the archive. The main idea is to keep those solutions that constitute an approximation to the Pareto front.

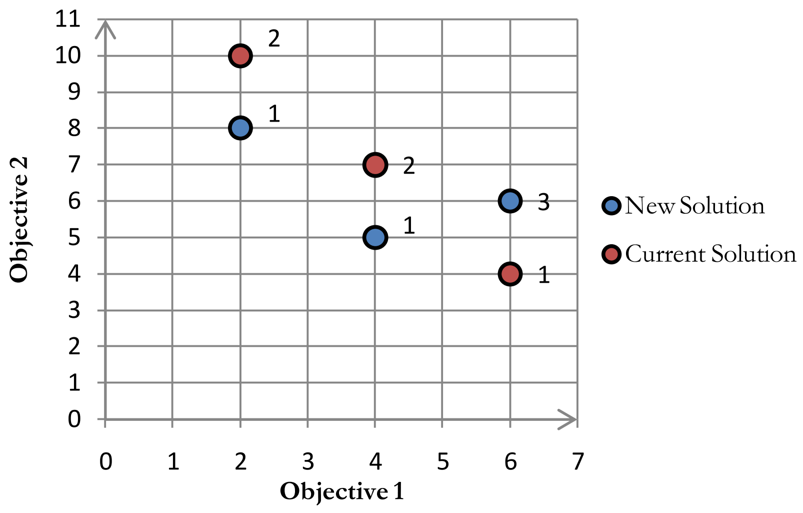
Current Population	Objective function value	
	Objective 1	Objective 2
1-2-3-4-5-6-7	2	10
2-4-6-7-1-3-5	4	7
3-1-5-2-4-7-6	6	4

Perform Mutation

New Solutions after mutation	Objective function value	
	Objective 1	Objective 2
1-2-3-4-5-7-6	2	8
7-4-6-2-1-3-5	4	5

3-5-2-4-1-7-6	6	6

Assign fitness value



New current population, after ranking

New Current Population	Objective function value	
	Objective 1	Objective 2
1-2-3-4-5-7-6	2	8
7-4-6-2-1-3-5	4	5
3-1-5-2-4-7-6	6	4

Figure 6.4: Scheme of the selection procedure

7. Results of the Heuristic

In Chapter 3 we mentioned that an heuristic is a technique which seeks "good" solutions at a reasonable computational time without any guarantee of optimality. Another interesting feature of heuristics is that they may not return the same outcome twice, since they are based on randomized processes that use different random seeds. Taking these features into consideration the best way to evaluate our heuristic is by comparing the near optimal solution provided by this one with the exact optimal solutions obtained by an exact approach. This is definitely a very important criterion, but not the only one. Another important aspect that we should take into account is the time that the algorithm requires to find these approximate solutions. In order to compare both methods 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 7.1 summarizes the results obtained with the heuristic approach as well as the results obtained previously with the exact approach.

The instances presented next were run on a PC Pentium Centrino / 1.6 GHz / 504MB RAM.

	Heuristic		**ε-constraint**	
Instance	**Nondominated Points (found)**	**Execution Time**	**Nondominated Points**	**Execution Time**
Instance 1	3	<1sec	3	3sec
Instance 2	9	1sec	9	1min 7sec
Instance 3	10	2sec	10	2h 49min 6sec
Instance 4	5	5sec	5	47sec
Instance 5	10	10sec	10	1h 38min 24sec
Instance 6	14	18sec	14	12h 57min 37sec

Instance 7	10	6sec	10	5h 51min 12sec
Instance 8	14	13sec	14	9h 46min 9sec
Instance 9	11	14sec	11	14h 20min 32sec
Instance 10	7	5sec	7	8h 21min 42sec
Instance 11	8	6sec	8	9h 19min 22sec
Instance 12	10	20sec	10	18h 33min 49sec

Table 7.1: Results of the heuristic and ε-constraint

Analyzing each instance we easily verify that the heuristic was able to find all nondominated points in a remarkable short period of time. For example, comparing instances 5 and 6 we observe that the difference between their execution time is more than 8 hours, on the ε-constraint algorithm, while on the heuristic is just 8 seconds. This huge difference between execution times gives us the impression that these two algorithms work in completely different time dimensions. In conclusion results indicate 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 reasons for such good results are due to the simplicity and intuitiveness of the heuristic processes performed. Furthermore we believe that the effectiveness of the neighboring moves chosen and the fast evaluation procedure of the neighboring solutions have also an important role in the generation of high quality solutions as well as in the time required to find them. Another remarkable fact is the ability of the insertion method to find very good initial solutions that compose the initial population. The following table (Table 7.2) illustrates the number of optimal solutions presented in the initial population for each instance. Looking at Table 7.2 we notice that most of the initial populations have at least 2 optimal solutions. These solutions are usually lexicographic.

Instance	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Nº Nondominated Points	3	3	3	2	3	2	3	3	3	2	2	3

Table 7.2: Number of optimal solutions in the initial population

Another interesting exercise is to observe the progress of the heuristic during the search process. This exercise aims to demonstrate the convergence of the algorithm to the nondominated set. For this purpose we applied our heuristic scheme to instance 6 and make it stop after a predefined 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.

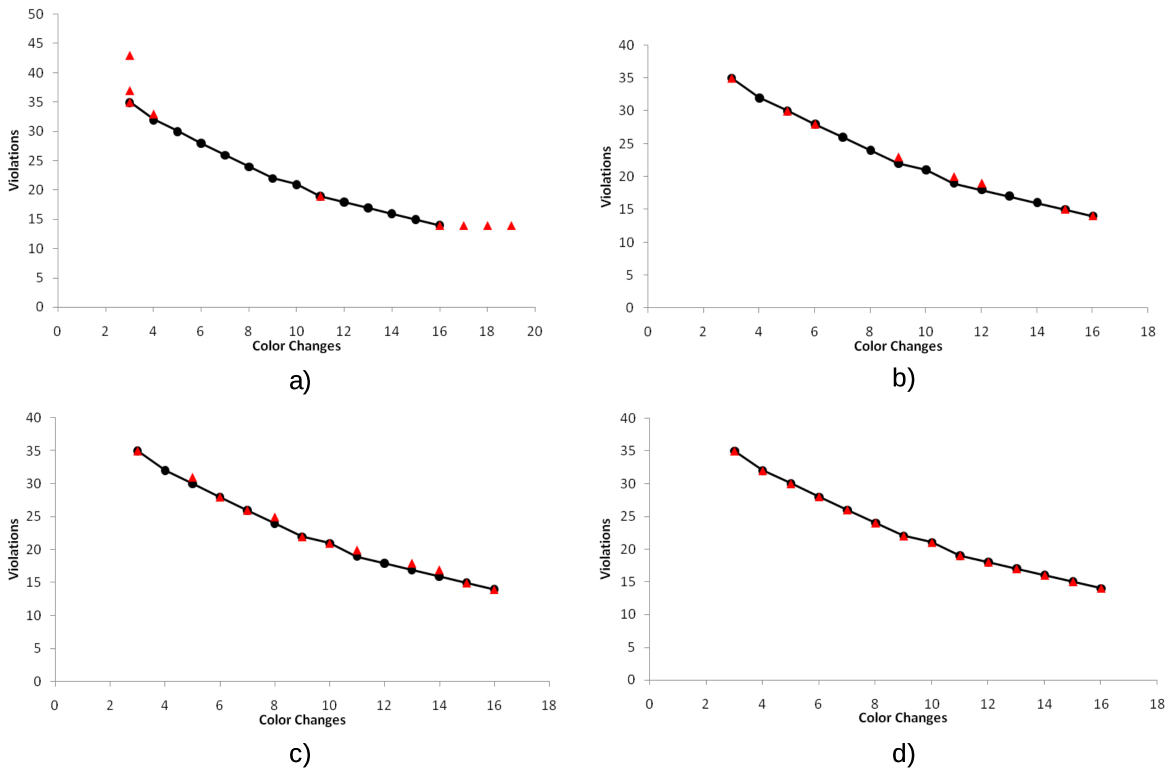


Figure 7.1: Best solutions found on instance 6, a) in the initial population, b) at iteration 1500, c) at iteration 5000 and d) iteration 30000.

In Figure 7.1 a) the red triangles corresponds to the objective value vectors of the solutions that constitutes the initial population. Next, in Figure 7.1 b) we have the best solutions found after performing 1500 iterations. In Figure 7.1 c) we got the best solutions found after 5000 iterations, which are executed in less than 3 seconds. At this moment, half of the solutions in the archive correspond to nondominated points, and the remaining ones are very close to optimality. Finally, after, approximately, 30000 iterations (18 seconds) we found the whole Pareto front. Looking at these results we can conclude that, in this case, 17% of the execution time was enough to find half of the nondominated set. Repeating the exercise for instance 12, we noticed that 8000 iterations (4 seconds) out of 50000 iterations (20 seconds) was sufficient to obtain half of the nondominated set (see Figure 7.2). In this case the percentage of time needed to find half of the nondominated set, is 20%. The values presented here are not accurate, since they can vary slightly from run to run.

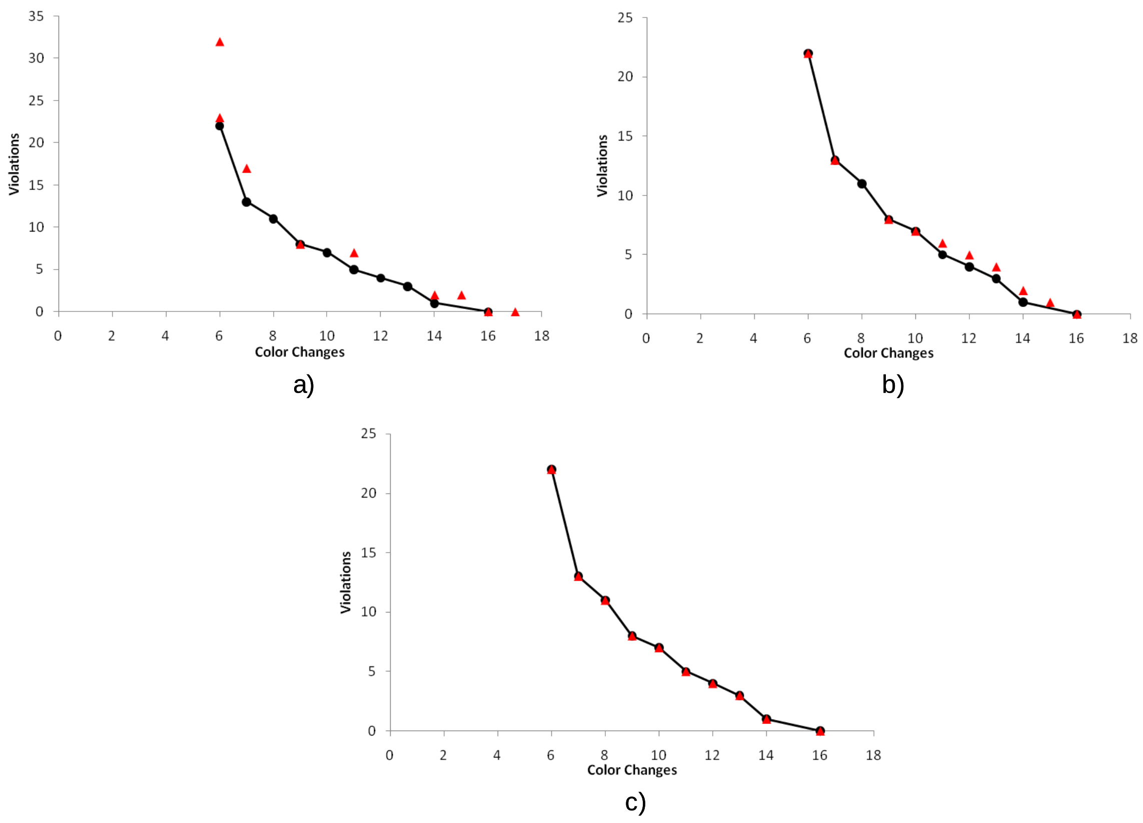


Figure 7.2: Best solutions found on instance 12, a) in initial population, b) at iteration 8000, and c)at iteration 50000.

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. Unfortunately, due to several limitations concerning the exact approach, it is not possible to know the nondominated set for instances with a size up to 60 - 70 vehicles. Even so, we tried to solve an instance with 70 vehicles, but the results were far from being optimal. The results of this attempt are shown in Figure 7.3 where the black dots correspond to solutions provided by the exact approach, and the red triangles correspond to the best solutions found by the heuristic within 120 seconds. Analyzing the results we notice that most of the solutions obtained by the ε-constraint are dominated by the solutions obtained with the heuristic, when was suppose to be the contrary. The reason why the ε-constraint algorithm did not find the nondominated points, is due to the CPLEX execution time limit. In other words, each time that the program calls CPLEX to solve the ILP model there is a pre-define run time limit to solve it, which is 1000 seconds by default. When CPLEX reach this time limit the execution terminates and returns the best solution found so far, which might not be the optimal solution. We could extend this time limit, but taking into account that the ε-constraint algorithm took more than 24 hours to find these poor quality solutions, if we did so the overall execution time would be even greater. Furthermore we have no idea how much time would be required to solve the instance to optimality.

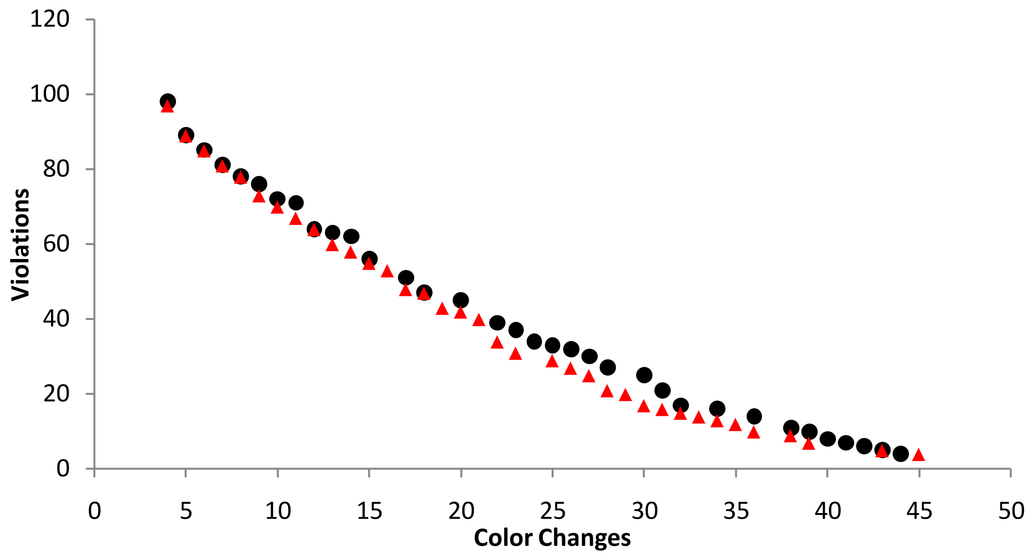


Figure 7.3: Solutions generated by the heuristic and ε-constraint algorithm on a CSP instance with 70 vehicles

We see in Chapter 5 that one possible way to reduce the overall execution time of the ε-constraint algorithm was to increase the interval Δε. The disadvantage of increasing the interval is that we "miss" some of the optimal solutions. Furthermore, instead of nondominated solutions we may obtain some weakly dominated solutions. In order to be able to evaluate our heuristic scheme in larger instances, and taking into account the last words, we applied the ε-constraint to two instances with 65 (Instance 13) and 70 (Instance 14) vehicles respectively, but using an interval Δε = 5. In addition we increased the CPLEX execution time limit from 1000 to 4000 seconds to allow the algorithm to do a deeper search. Our goal was to find just some points of the nondominated set. The results from this experience are expressed in the Figure 7.4 and 7.5.

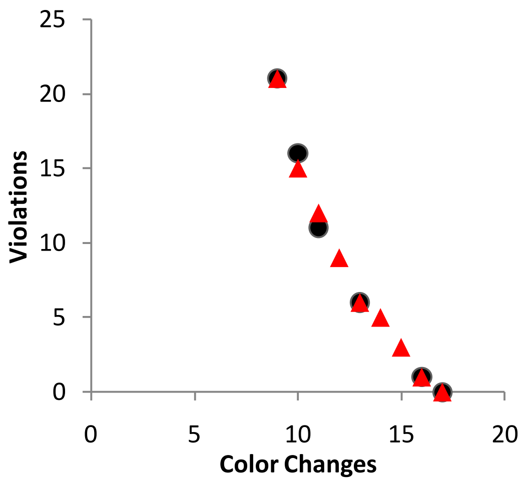


Figure 7.4: Instance 13

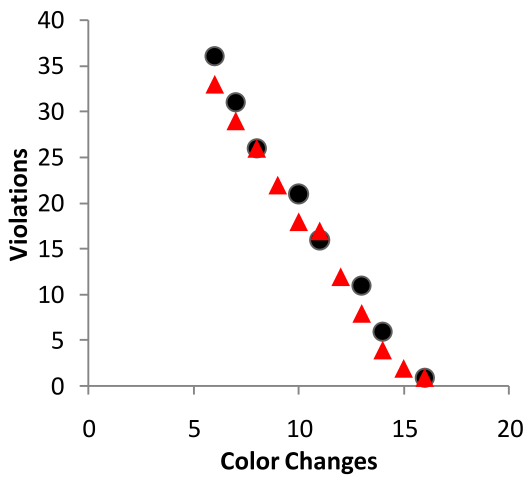


Figure 7.5: Instance 14

Instance	Execution Time (Heuristic)	Execution Time (ε-constraint)
Instance 13	48sec	7h 49min 22sec
Instance 14	147sec	9h 36min 52sec

Table 7.3: Execution time of instances 13 and 14

As we were already expecting the execution time decreased significantly (see Table 7.3) but the quality of the solutions found by the ε-constraint was not optimal, especially in instance 14. For both instances the exact method "misses" several nondominated points. In instance 14 the ε-constraint was not even able to find the lexicographic solution (6, 33), which means that 4000 seconds per run are not enough yet. An important observation that we have to remark is that in both instances (13 and 14) there is one solution that our heuristic was not able to match which are: (11, 11) in instance 13 and (11, 16) in instance 14. Later we increased the CPLEX run time to 6000 seconds but the solutions remain weakly dominated. Moreover, as the instances become more constrained, the exact approach is not even able to find the lexicographic solutions.

Despite the limitations of the ε-constraint illustrated in previous figures, these results give us a very good indication about the performance of our heuristic. We are not sure if the solutions provided by the heuristic are optimal but the fact that most of them dominate the solutions provided by the exact approach demonstrates a great ability to find high quality solutions. Another aspect that it is important to highlight is the execution time. In these last instances we noticed a significant increase in the execution time of the heuristic. This increase on the execution time, in relation to the time required to solve the smaller instances tested before, has to do with complexity issues.

Due to the reasons stated before is not possible to know the nondominated set for more difficult instances. So how can we evaluate the performance of the heuristic for more demanding instances? One possible alternative is to build other more powerful exact and heuristic or metaheuristic schemes specially design to solve the CSP. The idea is to use different methods and compare the results with each other.

8. Conclusions and Future Research

The results obtained before, have shown that the exact approach is able to find the nondominated set but just for small instances with less than 60 vehicles. Even so, there are instances with a small number of vehicles but highly constrained, whose optimality can not be proved due to computational limitations, like time and memory. Furthermore, the attempt to reduce the search intervals by increasing the constant Δε has not shown to be a very good strategy because it is not able to ensure that the solutions generated are nondominated or weakly dominated. In additions the algorithm "misses" several nondominated solutions. The CSP is a pure integer programming problem where all variables are integer. This integer program tend to be NP-complete (formally intractable) which means that are very hard to solve and are very demanding in terms of computational resources. These facts explain why most of the instances tested took so much time to be solved and why this kind of approaches is just able to solve small instances.

The local search approach proposed in this thesis, proved 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 very short period of time. Such promising results are very encouraging. However more instances with a higher difficulty level need to be tested. Unfortunately, as we said before, for harder instances the exact algorithm is not able to provide the nondominated set, consequently is not possible to evaluate the results given by the local search approach, unless we use other heuristic schemes.

Future Research

The exact approach used here has several limitations that unable it to solve harder instances, but there are some opportunities for improvement. One way to decrease the difficulty level of an integer programming model is by performing a linear relaxation. This consists in transforming some integer variables in linear variables. The idea is to "transform" an integer problem into a linear problem, which is usually much easier to solve. However the results provided by this technique should be carefully interpretated, since it does not provide nondominated points but rather approximate. These points can be interpretated as lower bounds that can be used as a reference for the heuristic approach.

Besides the previous suggestion we believe that it is possible to develop a deterministic (i.e. exact) algorithm specifically designed to solve the CSP from a Pareto optimality perspective.

In relation to the heuristic approach the results obtained are quite interesting, but we knew that sooner or later, for harder instances the algorithm would start fail finding the optimal solutions. Instead of these last ones, less approximate solutions will tend to appear. In order to develop the performance of the heuristic method presented in this thesis we suggest the following improvements:

As we said before the neighboring moves chosen for our method are very fast to perform, but they are also very similar to each other, in other words, the degree of transformation provoked by each of them are very similar. As a consequence the exploitation of some regions of the search space might be compromise. One possible way to overcome this problem is to introduce new neighboring moves that provoke a higher level of transformation in the sequences, for example inverting a subsequence of cars, or performing a random shuffle in a group of vehicles. The disadvantage of these neighboring moves is that they require more time to be evaluated than those used in the local search algorithm. Therefore they should not be applied very frequently.

The fastest and simplest way to choose the positions where to apply neighboring moves is picking them randomly. This strategy works fine at the early stages of the search process. However when it becomes harder to find better solutions, other sharper strategies need to be implemented.

One of the biggest problems of local search algorithms is getting trapped in local optimum solutions. To avoid this phenomenon we simply restart the population after a certain number of iterations. This method is very simple and perhaps too naive. For easy instances this process works well however for bigger and harder instances, this method is definitely inefficient because it does not allows to perform a deep search into some areas, since the process might be interrupted before it reaches an optimum solution. As an alternative we suggest the application of higher-level strategies through the implementation of other metaheuristic methods like Tabu Search, Simulated Annealing, Ant Colony Optimization or even Evolutionary Algorithms. All these methods have already been applied to the Car Sequencing Problem with very good results, but never from a Pareto optimality perspective. Furthermore a benchmark between these various methods would be an interesting exercise.

Another important issue is the impact of some parameters in the algorithm performance. Parameters such as the population size, time limit before restart the population and the execution time, influences the computational resources used as well as the overall performance. It's also clear that different instances require different computational resources, which means that the parameters value should vary according to the instance features. If we knew the way which these parameters affects the algorithm, we could tune their values for each instance. The idea is to do a better use of the computational resources.

Bibliography

[Bäck, 1996] T. Bäck. *Evolutionary Algorithms in Theory and Practice.* Oxford University Press, 1996.

[Chankong and Haimes, 1983] V. Chankong and Y.Y Haimes. Multiobjective Decision Making: Theory and Methodology. In Andrew P. Sage, editor, Systems Science and Engineering. North-Holland, 1983.

[Cordeau *et. al.*, 2005] G. Laporte, F. Pasin, J.-F. Cordeau. *Iterated Tabu Search for the Car Sequencing Problem.* European Journal of Operational Research. In press.

[Deb *et. al.*, 2002] K. Deb, S. Agrawal, A. Prataand, and T. Meyarivan. A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182-197, 2002.

[Ehrgott and Ryan, 2002] M. Ehrgott and D. M. Ryan. *Constructing robust crew schedules with bicriteria optimization.* Journal of Multi-Criteria Decision Analysis 11(3):139-150, 2002.

[Ehrgott and Gandibleux, 2004] M. Ehrgott and X. Gandibleux. *Approximative solution methods for multiobjective combinatorial optimization.* TOP 12(1):1-88, 2004.

[Ehrgott, 2000] M. Ehrgott. *Hard to say it's easy - four reasons why combinatorial multiobjective programmes are hard.* In Y.Y. Haimes and R.E. Steuer, editors, Research and Practice in Multiple Criteria Decision Making, volume 487 of Lecture Notes in Economics and Mathematical Systems, pp. 69-81. Berlin, Springer, 2000.

[Ehrgott, 2005] M. Ehrgott. *Multicritera Optimization.* Springer, Berlin, 2nd edition, 2005.

[Fonseca and Fleming, 1993] C. M. Fonseca and P. J. Fleming, Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. *In Proceedings of the Fifth International Conference on Genetic Algorithms*, S. Forrest, Ed. San Mateo, CA: Morgan Kauffman Publishers, pp. 416-423, 1993.

[Garey and Johnson, 1979] M. R. Garey and D. S. Johnson. *Computer and Intractability.* W. Freeman and Company, New York, USA, 1979.

[Gent, 1998] I.P. Gent. *Two results on car-sequencing problems.* Technical report (http://www.apes.cs.strath.ac.uk/apesreports.html), APES, 1998.

[Glover, 1987] Fred Glover, *Tabu Search Methods in Artificial Intelligence and Operations Research*, *ORSA Artificial Intelligence*, Vol. 1, No. 2, 6, 1987.

[Goldberg, 1989] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[Gravel *et. al.,* 2005] M. Gravel, C. Gagn´e, and W.L. Price. *Review and comparison of three methods for the solution of the car-sequencing problem.* Journal of the Operational Research
Society, 56(11):1287–1295, 2005.

[Hansen and Jaszkiewick, 1998] M. P. Hansen and A. Jaszkiewicz. *Evaluating the quality of approximations to the non-dominated set.* Technical Report IMM-REP-1998-7, Institute of Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark, 1998.

[Hoos and Stützle, 2004] H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications.* Morgan Kaufmann/Elsevier, 2004.

[Haimes *et. al.*, 2004] Y. Haimes, L. Lasdon, and D. Wismer. *On a bicriterion formulation of the problems of integrated system identification and system optimization.* IEEE Transactions on Systems, Man, and Cybernetics, 1:296-297, 1971.

[Kirkpatrick, 1983] S. Kirkpatrick and C. D. Gelatt and M. P. Vecchi, *Optimization by Simulated Annealing*, Science, Vol. 220, Number 4598, pp. 671-680, 1983.

[Kis, 2004] T. Kis. On the complexity of the car sequencing problem. Operations Research Letters, 32:331–335, 2004.

[Laumanns, 2004] M. Laumanns, L. Thiele, and E. Zitzler. *An Adaptive Scheme to Generate the Pareto Front Based on the Epsilon-Constraint Method.* TIK-Report No. 199, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, September 2004.

[Mietttinen, 1999] K. M. Miettinen, Nonlinear Multiobjective Optimization, Kluwer Academic Publishers, Boston, Massachusetts, 1999

[Nguyen *et. al.*, 2005] C. Solnon, V. D. Cung, A. Nguyen and C. Artigues. *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, 2007, in press.

[Nouioua *et. al.*, 2006] B. Estellon, F. Gardi et K. Nouioua. *Two local search approaches for solving real-life car sequencing problems.* European Journal of Operational Research, 2006, in press

[Papadimitriou and Steiglitz, 1982] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization – Algorithms and Complexity.* Prentice Hall, Englewood Cliffs, NJ, 1982.

[Parello *et. al.,* 1986] B.D. Parello, W.C. Kabat, and L. Wos. Job-shop scheduling using automated reasoning: a case study of the car sequencing problem. Journal of Automated Reasoning, 2:1–42, 1986.

[Paquete, 2005] L. Paquete. *Stochastic Local Search Algorithms for Multiobjective Combinatorial Optimization: Methods and Analysis.* PhD thesis, Technische Universität Darmstadt, Darmstadt, Germany, 2005.

[Prandtstetter, 2005] M. Prandtstetter. *Exact and heuristic methods for solving the car sequencing problem.* MSc thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms, August 2005.

[Ribeiro *et al.*, 2007] C.C. Ribeiro, D. Aloise, T.F. Noronha, C.T.M. Rocha e S. Urrutia, *A hybrid heuristic for a multi-objective real-life car sequencing problem with painting and assembly line constraints*, European Journal of Operational Research, 2007.

[Risler, 2004] M. Risler. *An Efficient Algorithm for the Car Sequencing Problem from the ROADEF 2005 Challenge.* MSc thesis, Technische Universität Darmstadt, Darmstadt, Germany, 2004.

[Smith, 1996] B. Smith. *Succeed-first or fail-first: A case study in variable and value ordering heuristics.* In third Conference on the Practical Applications of Constraint Technology PACT'97, pp. 321–330, 1996.

[Srinivas and Deb, 1994] N. Srinivas and K. Deb. *Multiobjective optimization using non-dominated sorting in genetic algorithms. Evolutionary Computation*, 3(2):221-248, 1994.

[Steuer, 1986] R. E. Steuer. *Multiple Criteria Optimization: Theory, Computation and Application*. John Wiley & Sons, editor, New York, 1986.

[Zitzler and Thiele, 1999] E. Zitzler and L. Thiele. *Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. IEEE Transactions on Evolutionary Computation*, 4(3):257-271, 1999.

[Zitzler *et. al.*, 2001] E. Zitzler, M. Laumanns, and L. Thiele. *SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimization.* Technical Report TIK-Report No. 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), Zürich, May 2001.

[Zitzler *et. al.*, 2003] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. Grunert da Fonseca. *Performance assessment of multiobjective optimizers: An analysis and review. IEEE Transactions on Evolutionary Computation,* 7(2):117-132, 2003.

Appendix

A.1 Instance 1 – 10 vehicles

Paint batch limit	4

	k_1	k_2	k_3	k_4	Ratios
cp_1	1	0	1	0	1/3
cp_2	0	1	1	0	2/3
Color	1	2	2	1	
Demand	3	3	2	2	

Table A.1.1.: The configurations for the current day and their demand

Position	cp_1	cp_2	Color
n	0	1	2
n-1	1	1	2
n-2	1	0	1
n-3	1	0	1

Table A.1.2.: The configuration of last cars produced in the previous day

A.2 Instance 2 – 15 vehicles

Paint batch limit	4

	k_1	**k_2**	**k_3**	**k_4**	**k_5**	**Ratios**
cp_1	1	0	1	0	0	1/3
cp_2	0	1	1	0	1	2/3
cp_3	0	1	0	1	0	1/3
Color	1	2	2	1	3	-
Demand	3	3	1	2	6	

Table A.2.1.: The configurations for the current day and their demand

Position	**cp_1**	**cp_2**	**cp_3**	**Color**
n	0	1	0	2
n-1	1	1	0	2
n-2	1	0	1	1
n-3	1	0	0	1

Table A.2.2.: The configuration of last cars produced in the previous day

A.3 Instance 3 – 20 vehicles

Paint batch limit	5

	k_1	k_2	k_3	k_4	k_5	k_6	Ratios
cp_1	1	0	0	0	0	1	1/3
cp_2	0	1	0	0	0	1	1/3
cp_3	0	0	1	0	0	1	1/2
cp_4	0	0	0	1	0	1	1/3
Color	1	2	3	2	4	4	-
Demand	8	4	2	3	1	2	

Table A.3.1.: The configurations for the current day and their demand

Position	cp_1	cp_2	cp_3	cp_4	Color
n	0	1	0	0	2
n-1	1	1	0	0	2
n-2	1	0	1	0	1
n-3	0	0	0	1	3

Table A.3.2.: The configuration of last cars produced in the previous day

A.4 Instance 4 – 23 vehicles

Paint batch limit	7

	k_1	k_2	k_3	k_4	k_5	k_6	Ratios
cp_1	1	0	1	0	0	0	1/3
cp_2	0	0	1	0	1	0	1/2
cp_3	0	1	0	1	0	1	2/3
cp_4	0	0	0	1	0	0	2/4
Color	1	2	2	1	3	3	-
Demand	12	3	1	1	4	2	

Table A.4.1.: The configurations for the current day and their demand

Position	cp_1	cp_2	cp_3	cp_4	Color
n	0	1	0	0	2
n-1	1	1	0	0	2
n-2	1	0	1	0	1
n-3	0	0	0	1	3

Table A.4.2.: The configuration of last cars produced in the previous day

A.5 Instance 5 – 23 vehicles

Paint batch limit	10

	k_1	k_2	k_3	k_4	k_5	k_6	**Ratios**
cp_1	1	0	1	0	0	0	1/3
cp_2	0	0	1	0	1	0	1/3
cp_3	0	1	0	1	0	1	2/4
cp_4	0	0	0	1	0	0	1/4
Color	1	2	2	1	4	3	-
Demand	8	3	3	3	4	2	

Table A.5.1.: The configurations for the current day and their demand

Position	**cp_1**	**cp_2**	**cp_3**	**cp_4**	**Color**
n	0	1	0	0	2
n-1	1	1	0	0	2
n-2	1	0	1	0	1
n-3	0	0	0	1	3

Table A.5.2.: The configuration of last cars produced in the previous day

A.6 Instance 6 – 25 vehicles

Paint batch limit	6

	k_1	k_2	k_3	k_4	k_5	k_6	**Ratios**
cp_1	1	0	0	0	0	1	1/3
cp_2	0	1	0	0	0	1	1/3
cp_3	0	0	1	0	0	1	2/3
cp_4	0	1	0	0	0	1	1/2
cp_5	0	0	0	0	1	0	1/4
Color	1	2	3	2	4	4	-
Demand	8	5	2	4	4	2	

Table A.6.1.: The configurations for the current day and their demand

Position	**cp_1**	**cp_2**	**cp_3**	**cp_4**	**cp_5**	**Color**
n	0	1	0	0	0	2
n-1	1	1	0	0	0	2
n-2	1	0	1	0	0	1
n-3	0	0	0	1	0	3

Table A.6.2.: The configuration of last cars produced in the previous day

A.7 Instance 7 – 30 vehicles

Paint batch limit	5

	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	Ratios
cp_1	1	0	0	0	1	0	0	0	1/3
cp_2	0	1	0	0	0	0	0	0	1/4
cp_3	0	0	1	0	0	1	0	0	1/2
cp_4	0	0	0	1	0	0	0	0	1/3
cp_5	0	0	0	0	1	0	1	0	2/3
cp_6	0	0	0	0	0	1	0	1	1/4
cp_7	0	0	0	0	0	0	1	1	1/2
Color	1	2	3	1	2	4	3	1	-
Demand	2	1	1	15	5	1	3	2	

Table A.7.1.: The configurations for the current day and their demand

Position	cp_1	cp_2	cp_3	cp_4	cp_5	cp_6	cp_7	Color
n	0	1	0	0	0	0	0	2
n-1	1	1	0	0	0	0	0	2
n-2	1	0	1	0	0	0	0	1
n-3	0	0	0	1	0	0	0	3

Table A.7.2.: The configuration of last cars produced in the previous day

A.8 Instance 8 – 30 vehicles

Paint batch limit	20

	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	Ratios
cp_1	1	0	0	0	1	0	0	0	1/3
cp_2	0	1	0	0	0	0	0	0	1/4
cp_3	0	0	1	0	0	1	0	0	1/2
cp_4	0	0	0	1	0	0	0	0	1/3
cp_5	0	0	0	0	1	0	1	0	2/3
cp_6	0	0	0	0	0	1	0	1	1/4
cp_7	0	0	0	0	0	0	1	1	1/2
Color	1	2	3	1	2	4	3	1	-
Demand	2	1	1	15	5	1	3	2	

Table A.8.1.: The configurations for the current day and their demand

Position	cp_1	cp_2	cp_3	cp_4	cp_5	cp_6	cp_7	Color
n	0	1	0	0	0	0	0	2
n-1	1	1	0	0	0	0	0	2
n-2	1	0	1	0	0	0	0	1
n-3	0	0	0	1	0	0	0	3

Table A.8.2.: The configuration of last cars produced in the previous day

A.9 Instance 9 – 35 vehicles

Paint batch limit	8

	k_1	**k_2**	**k_3**	**k_4**	**k_5**	**k_6**	**k_7**	**Ratios**
cp_1	1	0	0	0	1	1	0	2/3
cp_2	0	1	0	0	0	1	0	1/3
cp_3	0	0	1	0	0	1	0	2/3
cp_4	0	1	0	0	0	1	0	1/2
cp_5	0	0	0	0	1	0	0	1/4
Color	1	2	3	2	4	4	3	-
Demand	10	4	2	3	4	2	10	

Table A.9.1.: The configurations for the current day and their demand

Position	**cp_1**	**cp_2**	**cp_3**	**cp_4**	**cp_5**	**Color**
n	0	1	0	0	0	2
n-1	1	1	0	0	0	2
n-2	1	0	1	0	0	1
n-3	0	0	0	1	0	3

Table A.9.2.: The configuration of last cars produced in the previous day

A.10 Instance 10 – 40 vehicles

Paint batch limit	10

	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	Ratios
cp_1	1	0	0	0	0	1	1	0	1/3
cp_2	0	1	0	0	0	1	0	1	1/3
cp_3	0	0	1	0	0	1	0	0	1/2
cp_4	0	0	0	1	0	1	0	0	1/3
cp_5	0	0	0	0	1	0	1	0	2/3
cp_6	0	0	0	0	0	1	0	1	2/4
Color	1	2	3	5	4	4	3	1	-
Demand	25	2	2	2	6	1	1	1	

Table A.10.1.: The configurations for the current day and their demand

Position	cp_1	cp_2	cp_3	cp_4	cp_5	cp_6	Color
n	0	1	0	0	0	0	2
n-1	1	1	0	0	0	0	2
n-2	1	0	1	0	0	0	1
n-3	0	0	0	1	1	0	3

Table A.10.2.: The configuration of last cars produced in the previous day

A.11 Instance 11 – 40 vehicles

Paint batch limit	20

	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	Ratios
cp_1	1	0	0	0	0	1	1	0	1/3
cp_2	0	1	0	0	0	1	0	1	1/3
cp_3	0	0	1	0	0	1	0	0	1/2
cp_4	0	0	0	1	0	1	0	0	1/3
cp_5	0	0	0	0	1	0	1	0	2/3
cp_6	0	0	0	0	0	1	0	1	2/4
Color	1	2	3	5	4	4	3	1	-
Demand	25	2	2	2	6	1	1	1	

Table A.11.1.: The configurations for the current day and their demand

Position	cp_1	cp_2	cp_3	cp_4	cp_5	cp_6	Color
n	0	1	0	0	0	0	2
n-1	1	1	0	0	0	0	2
n-2	1	0	1	0	0	0	1
n-3	0	0	0	1	1	0	3

Table A.11.2.: The configuration of last cars produced in the previous day

A.12 Instance 12 – 50 vehicles

Paint batch limit	15

	k_1	k_2	k_3	k_4	k_5	k_6	k_7	**Ratios**
cp_1	1	0	0	0	0	0	1	3/4
cp_2	0	1	0	0	0	0	0	2/5
cp_3	0	0	1	0	0	1	0	2/4
cp_4	0	0	0	0	0	0	0	1/4
cp_5	0	0	0	0	1	1	0	2/4
cp_6	0	0	0	0	1	0	1	1/3
Color	1	2	3	1	4	1	5	-
Demand	35	2	3	4	1	2	3	

Table A.12.1.: The configurations for the current day and their demand

Position	cp_1	cp_2	cp_3	cp_4	cp_5	cp_6	**Color**
n	0	1	0	0	0	0	2
n-1	1	1	0	0	0	0	2
n-2	1	0	1	0	0	0	1
n-3	0	0	0	1	1	0	3

Table A.12.2.: The configuration of last cars produced in the previous day

A.13 Instance 13 – 65 vehicles

Paint batch limit	12

	k_1	k_2	k_3	k_4	k_5	k_6	**Ratios**
cp_1	0	1	0	0	1	0	2/3
cp_2	0	1	0	0	1	0	1/3
cp_3	0	0	1	0	1	0	1/3
cp_4	0	0	0	0	1	0	1/4
cp_5	0	0	0	1	1	1	2/4
Color	1	2	3	1	4	4	-
Demand	45	2	3	4	1	10	

Table A.13.1.: The configurations for the current day and their demand

Position	**cp_1**	**cp_2**	**cp_3**	**cp_4**	**cp_5**	**Color**
n	0	1	0	0	0	2
n-1	1	1	0	0	0	2
n-2	1	0	1	0	0	1
n-3	0	0	0	1	1	3

Table A.13.2.: The configuration of last cars produced in the previous day

A.14 Instance 14 – 70 vehicles

Paint batch limit	20

	k_1	k_2	k_3	k_4	k_5	k_6	k_7	**Ratios**
cp_1	0	1	0	0	1	0	0	2/3
cp_2	0	1	0	0	1	0	0	1/3
cp_3	0	0	1	0	1	0	0	1/3
cp_4	0	0	0	1	1	0	0	1/4
cp_5	0	0	0	0	1	1	0	2/4
cp_6	0	0	0	0	0	0	1	2/5
Color	1	2	3	1	4	4	5	-
Demand	45	3	2	4	2	10	4	

Table A.12.1.: The configurations for the current day and their demand

Position	**cp_1**	**cp_2**	**cp_3**	**cp_4**	**cp_5**	**cp_6**	**Color**
n	0	1	0	0	0	0	2
n-1	1	1	0	0	0	0	2
n-2	1	0	1	0	0	0	1
n-3	0	0	0	1	1	0	3

Table A.12.2.: The configuration of last cars produced in the previous day