FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Integrated production planning and scheduling optimization

Daniel Carvalho

Mestrado Integrado em Engenharia Electrotécnica e de Computadores

Supervisor: Pedro Amorim

Co-Supervisor: Eduardo Curcio

August 14, 2019

Resumo

Este trabalho propõe um método de solução iterativa para abordar a integração do planeamento táctico (dimensionamento de lotes) e operacional (sequenciamento) numa produção industrial com setups dependentes da sequência. Este método decompõe o problema da integração em dois. No primeiro sub-problema, planeamento táctico, o plano de produção é optimizado sem ter em conta setups necessários, usando um modelo matemático. O sequenciamento dos produtos é depois definido usando estratégias de pesquisa local. O resultado final será usado como feedback na formulação de regras adicionais para complementarem o modelo do primeiro sub-problema. De seguida, o planeamento táctico é repetido, considerando as novas regras definidas anteriormente. O algoritmo continua iterativamente até que as funções objectivo dos dois níveis convirjam. Os principais ganhos deste método são o seu procedimento intuitivo e fácil de implementar, e o pouco tempo computacional necessário. De modo a analisar resultados obtidos, dois experimentos computacionais são propostos. O primeiro para comparar o método iterativo com outros métodos de solução encontrados na literatura para problemas similares, nomeadamente meta-heuristicas e modelos de programação inteira. Por fim, a investigação foi focada num caso de uma indústria de nutrição animal, onde o setup de produção é dependente da sequência e não-triangular. O propósito do segundo experimento é avaliar os eventuais ganhos desta abordagem no planeamento de produção nesta indústria, usando os dados de uma empresa real. Os respectivos resultados demonstraram que o método iterativo é uma boa solução para problemas de produção extensivos, nos quais um modelo MIP necessita de tempos computacionais impraticáveis, derivado ao elevado número de variáveis binárias.

Abstract

This work proposes an iterative solution method to address the integration of the tactical (lot-sizing) and operational (scheduling) levels in production planning with sequence dependent setups. This method breaks the integrated lot-sizing and scheduling problem into two. In the first sub-problem, at the tactical level, the production plan is optimized with production setups disregarded using a mathematical model. The production scheduling solution is then defined using local search strategies. The final solution will serve as feedback to formulate additional rules to complement the first sub-problem model. After that, the tactical level is again optimized, considering the rules defined from the operational level. The algorithm continues iteratively until the objective functions from both levels converge. The main advantages of this method are it's intuitive and easy to implement procedure, and the low computational time required. In order to analyze results, two computational experiments are proposed. The first is performed to compare the solution method proposed with mixed-integer programming models and meta-heuristics from the literature. Then the research will focus on an animal feed industry case, in which production setup is sequence dependent and non-triangular. The purpose of the second experiment is to evaluate the potential gains to the production planning in this industry, using a real company dataset. The respective results showed that the iterative method is a good solution to extensive production problems, in which the MIP models require intractable computational times, resulting from the high number of binary variables.

Acknowledgements

Firstly I want to thank both of my supervisors. Eduardo Curcio, for the constant support throughout all the development of this work. I'm grateful for always being willing to help me to keep this thesis on track, without which this work would not exist. His professionalism and kindness in my guiding process will always inspire me in my professional career. I would also like to thank professor Pedro Amorim, for helping me with important insights in the writing process and key advice in the structure of this work.

Moreover, I express my gratitude to the all the co-authors that contributed to this work, particularly, professor Luis Guimarães, that helped me in the early stage of this work, defining the problem addressed and the possible solution methods.

I also want to thank the institution INESC TEC, for providing me with a place for the development of this thesis, where I always found an excellent working and friendly environment. Additionally I have to mention FEUP, where everyday I learned to be a better person and for making the last 5 years, the best of my life.

Lastly, I have to thank my family, especially my parents, for all the continuous patience and support that always kept me focus in this work. I want to also thank my close friends for always being there when I needed, in particular my college friends from FEUP, that everyday helped keeping me in a good mood, not only during the months of this work, but throughout all my college journey, and which friendships will continue for the last of my life.

Daniel Carvalho

“The people who are crazy enough to think they can change the world are the ones who do”

Steve Jobs

Contents

List of Figures

List of Tables

Abbreviations and symbols

ATSP	Asymetric travelling salesman problem
BRKGA	Biased random-key genetic algorithms
CLPS	Capacitated lot-sizing problem
CSLP	Continuous setup lot-sizing problem
DLSP	Discrete lot-sizing and scheduling problem
FO	Fix-and-Optimize
GA	Genetic Algorithm
GLSP	General Lot-sing and Scheduling problem
HIER	Hierarchical planning strategy
ILSP	Iterative Lot-Sizing and Scheduling Planning
ILSP_{nR}	Iterative Lot-Sizing and Scheduling Planning without resolution
ILSP_{R}	Iterative Lot-Sizing and Scheduling Planning with resolution
LS	Local Search
MA	Memetic Algorithm
MIP	Mixed integer programming
PLSP	Proportional lot-sizing and scheduling problem
RF	Relax-and-Fix
SA	Simulated Annealing
SC	Supply chain
TS	Tabu Search

Chapter 1

Introduction

1.1 Context

Production planning is one of the most challenging subjects in operations management, with great importance in the company's reduction of cost. In a production planning problem, the timing and sizes of production orders must be determined in order to fulfill the market demand and minimize the associated costs [4]. The two main production planning problems addressed in this work are lot-sizing and scheduling.

- Lot-sizing - determines the quantities of production necessary to satisfy deterministic product demand over a finite planning horizon.
- Scheduling - establishes the order in which lots are produced within a time period, accounting for the sequence-dependent setup time and costs.

1.1.1 Lot-Sizing and Scheduling in the Supply Chain

The supply chain (SC) of a manufacturing company is a network of organizations with the following main functions: acquisition of raw materials, transformation of raw material into finished products, and distribution of the products to costumers, having the goal to achieve high service level at low costs. The planning problems that have to be solved to achieve this purposes cover a wide range of time scales as described by [1]:

- ■ **Long-term** - Determines the structure of the supply chain (e.g., facility location).
- ■ **Medium-term** - Makes decisions such as the assignment of production targets to facilities and the transportation from them to warehouses or distribution centers. In the production stage, the lot-sizing problem is placed here.
- ■ **Short-term** - Carried out on a daily or weekly basis to determine the assignment of tasks and their sequence. In the production level, short-term planning is referred to as scheduling.

The production planning (medium-term) and scheduling (short-term) problems positions in the supply chain, previously described, are represented in Figure 1.1.

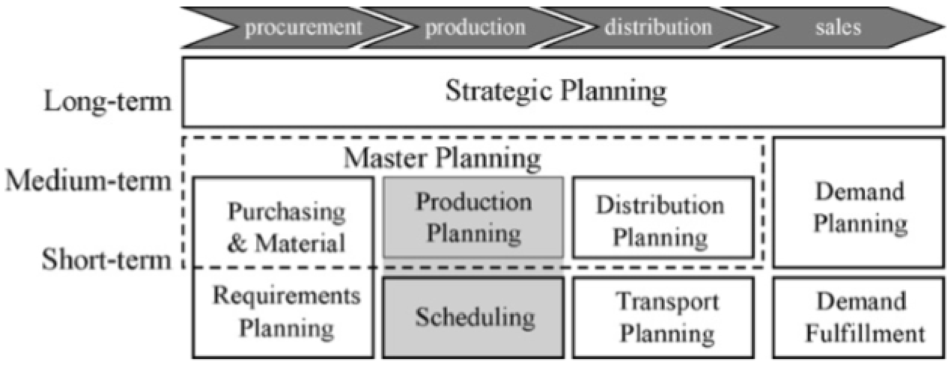


Figure 1.1: Production planning and scheduling positions in the supply chain [1].

1.1.1.1 Tactical planning

The medium-term planning in production is also called tactical planning, where it is decided the products to be produced in each macro-period (e.g., days) and the amounts necessary to fulfill the demand.

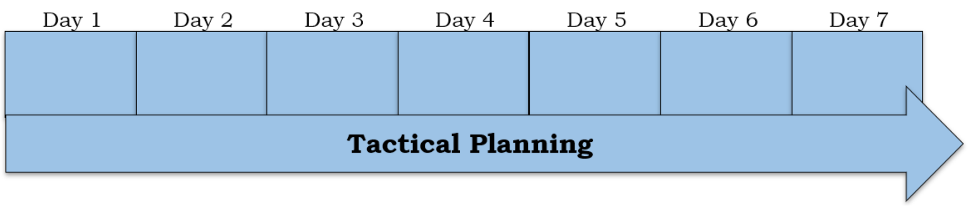


Figure 1.2: Tactical planning framework.

Decision: What products and how much should be produced in each one of the days?
Main objectives:

1. Minimize inventory costs.
2. Minimize backlog costs.
3. Minimize overtime costs.

1.1.1.2 Operational planning

The operational planning (short-term) has the goal to find the best sequence of production for the products defined previously in the tactical planning. This problem is called scheduling, defined as the act of determine priorities and arranging activities with the purpose of minimizing the production time and costs [5].

Figure 1.3 represents this operational phase, the previous macro-periods (days) are divided into micro-periods (e.g., hours) to plan the production sequence.

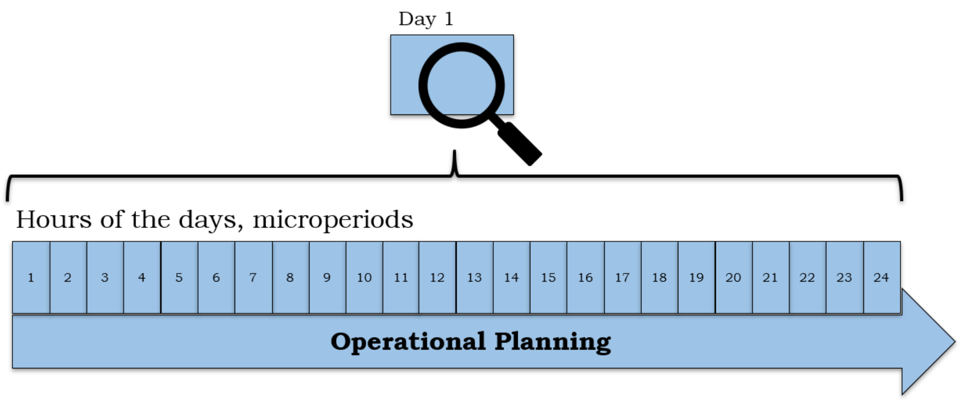


Figure 1.3: Operational planning framework.

Decision: What is the best the production sequence in each one of the days?
Main objectives:

1. Minimize the setup costs.

In sequence-based manufacturing, a changeover from one product to another usually causes setup costs as well as setup times which are sequence-dependent. If those those setups are not well managed it may result in significant losses in production capacity, which may lead to unmet demands and costumer dissatisfaction. This problem is frequently found in distinct industries like animal feed, automobile, chemical and electronics.

This work will later focus on a case of an animal feed company, in which it is frequent to have the possibility of contamination in the production of two consecutive products from different families, requiring a cleaning batch in between those products.

1.1.2 Integrated planning

Primarily most commercial production planning and control systems tried to construct feasible production plans in a step-wise manner, so the manufacturing resource planning (MRP II) logic was implemented. It can be divided in 3 main phases, as described by [6].

∗ Phase I: Starting with the final products, lot sizes are computed level by level, ignoring capacity constraints.

∗ Phase II: The results of the first phase usually exceeds the capacity in some periods. In this phase some lots are shifted to find a plan that meets the capacity limits.

∗ Phase III: The plan sequence decisions are made and the orders sent to the shop floor.

The MRP II concept, by having Phase I, II and III disconnected, may present some problems: long lead times, high work-in-progress, and unfulfilled orders. So sophisticated approaches are necessary to solve production planning and scheduling problems.

Strategies to solve this problem can be classified in 3 categories, illustrated in the Figure 1.4. Having one master and one slave, the communication can be unidirectional from the master to the slave (Hierarchical), or with feedback (Iterative). In the case that this division does not exist, and the problem formulation refers to all working periods, the solution will have all the necessary information to both the lot-sizing and scheduling problem (Full-space).

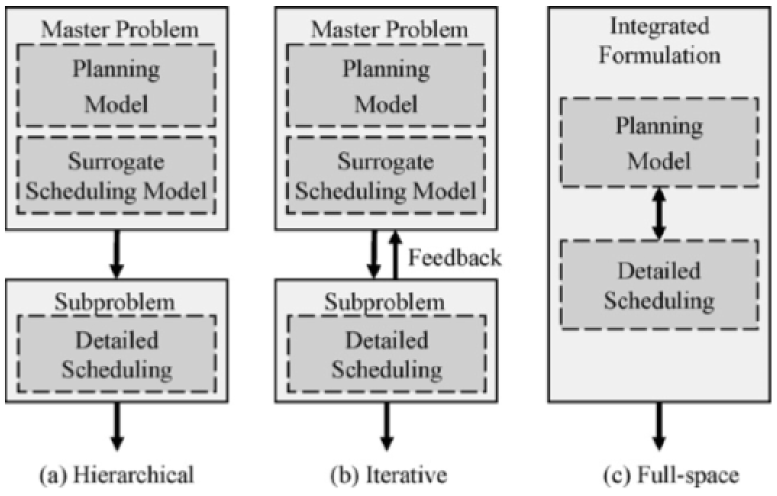


Figure 1.4: Solution strategies for production planning [1].

This study reviews these three approaches for the integration of medium-term production planning and short-term scheduling that enable the creation of better production plans than those obtained when solving the two problems independently by introducing the solution of the lot-sizing problem in the scheduling planning level.

The goal of this integration is to construct the production plan for all the planning horizon. Production plans are created with the objective of minimizing the overall costs consisting mainly of inventory holding, setup, overtime and backlog, while satisfying the available capacity and meeting the demand in each time period.

In the Figure 1.5 we can see a representation of an integrated planning solution that will be explored in this work.

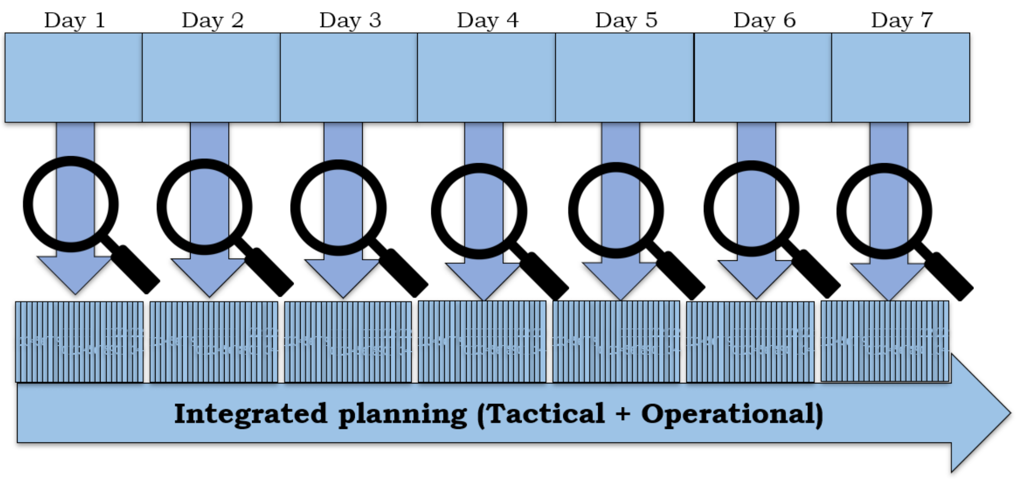


Figure 1.5: Integrated planning.

1.2 Motivation

Several companies face the problem of integrating lot-sizing and scheduling in their production planning over a given planning horizon, and the constant complexity growth in the industry's productions has urged a research from the scientific community to create new methods, more sophisticated, to attend the needs of the companies and keep competitiveness. In many of these production environments, switching between production lots triggers operations with costs for the company, such as machine adjustments and cleansing procedures.

The decisions of lot-sizing and scheduling are often made separately, which can cause operational costs and compromise the response to the demand deadlines their quality taking into account the operational costs and demand deadlines. So there is an urge in companies to integrate these phases. Therefore the two main motivations of this study are the following:

Scientific: Development of efficient solution methods to the lot-sizing and scheduling integration problems, found in the literature.

Practical: Obtain good solutions to the production planning that consider tactical and operational planning, that are aligned with the corporate objectives of the company.

1.3 Objectives

This work conducts a series of studies on mathematical models and other solution methods for integrated lot-sizing and scheduling problem and apply them in a real case of a company of animal feed. The ultimate goal is to optimize the production planning in order to reduce the company costs. Summarily our objectives are the following:

- Mathematically modeling a real optimization problem of production planning and scheduling.
- Developing and comparing different solution methods in terms of solution quality and computational complexity.
- Assessing the value of integration and the solution quality obtained over the current planning performed by an animal feed company.

1.4 Thesis structure

This thesis is organized as following. Chapter 2 presents a literature review on the subject that studies the current state of the art. The study initially focuses on the mathematical models used to represent this problem and then explores other solution methods based on heuristics and meta-heuristics. In Chapter 3, the problem addressed and its mathematical model are defined. The proposed iterative solution method is presented and detailed in Chapter 4. Chapter 5 firstly describes other solution methods commonly used for this planning problem that are used to benchmark the iterative method proposed. All methods are then assessed with the instances found in the literature and their results are compared, both in quality and computing power. Still in Chapter 5, a real case of an animal feed company in Brazil is studied, having a dataset and their planning results, the solutions and gains from the various method proposed are analyzed. In the last chapter, the conclusion of the work is presented as well as proposals for future work.

Chapter 2

Literature review

This chapter is divided in 3 sections. Firstly, a literature review on lot-sizing and scheduling problems is made, comparing the various mathematical models used to represent it, particularly the large and small bucket models, focusing then on the hybrid models that will be used on this study. Secondly, it presents a study on other solution methods using heuristics and meta-heuristics. The chapter concludes with an investigation on these solutions methods applied in the case of animal feed industry.

2.1 Mixed integer programming models

Linear programming is a mathematical optimization used to maximize (or minimize) a linear objective function subject to one or more constraints. An MIP model adds one additional condition that at least one of the decision variables can only take on integer values. The use of integer variables greatly expands the scope of useful optimization problems that you can define and solve. An important special case is a decision variable that must be either 0 or 1 at the optimal solution. Such variables are called binary integer variables and can be used to model yes/no decisions. However, integer variables make an optimization problem non-convex, and therefore far more difficult to solve. Memory and solution time may rise exponentially as more integer variables are added.

This literature review specifies the main lot-sizing and scheduling MIP models and their variants. It was based on [6] that summarizes the work in the field and details the differences of the lot-sizing and scheduling problems. They can be classified into 2 major categories [7], some based on micro-periods for short-term planning (small-bucket problems), and models which use macro-periods for medium-term planning (large-bucket problems).

2.1.1 Large-bucket problems

To represent the tactical planning level, [6] present the capacitated lot-sizing problem (CLSP) which determines the lot sizes of production but not the sequence of the lots. Several items may be produced per period, that represents a big time slot, typically one week in the real world. The planning horizon is usually less than six months. Next we present the MIP model for this problem:

Data:

$j = 1, \ldots\ldots, J$	Number of products
$t = 1, \ldots\ldots, T$	Number of periods
C_t	Capacity (in time units) available in period t.
p_j	Time required to produce one unit of product j.
h_j	Non-negative holding costs of product j
s_j	Setup costs for product j
d_{jt}	Demand of product j in period t
I_{j0}	Initial inventory of product j at the beginning of the planning horizon.

Decision variables:

$I_{jt} \geq 0$	Inventory quantity of product j at the end of period t.
$q_{jt} \geq 0$	Production quantity of item j produced in period t.
$x_{jt} \in \{0,1\}$	1, if a setup for item j occurs in period t(= 0 otherwise).

$$\min \sum_{j=1}^{J} \sum_{t=1}^{T} (h_j I_{jt} + s_j x_{jt}) \tag{2.1}$$

Subject to:

$$I_{jt} = I_{j,t-1} + q_{jt} - d_{jt} \qquad \forall t, j \tag{2.2}$$

$$p_j q_{jt} \leq C_t x_{jt} \qquad \forall t, j \tag{2.3}$$

$$\sum_{j=1}^{J} p_j q_{jt} \leq C_t \qquad \forall t \tag{2.4}$$

Objective function (2.1) minimizes the sum of setup and holding costs. Since this model doesn't take into account sequence-dependent setups, it assumes that a fixed setup cost always occurs for a new product. Constraints (2.2) represent the inventory balances and, due to Constraints (2.3), production of an item can only take place if the machine is set up for that particular product. Finally Constraints (2.4) ensure that the production capacity is respected.

2.1.2 Small-bucket problems

Subdividing the macro-periods of the CLSP into several micro-periods, that become the new variable t, leads to the small-bucket models. Therefore, they only optimize the production plan for one macro-period. The name small-bucket comes from the fact that, in this models, at most one item can be produced per period, hence they usually correspond to small time slots such as hours or shifts.

Firstly it is presented the discrete lot-sizing and scheduling problem (DLSP), first mentioned by [8]. Here is introduced the 'all-or-nothing' assumption: only one item may be produced per period, and, if so, production has to use the full capacity. The advantage over the CLSP is that minimum lead times, such as transportation time or time for cooling, can easily be taken into account, having short time periods in mind.

The decision variables and the parameters for the DLSP are the same as for the CLPS presented before. Although since we consider short periods, the setups costs should only be incurred if the production of a new lot begins. To model this, we need a new parameter and a new decision variable:

Data:

$y_{j0} \in \{0,1\}$ 1, if the machine is set up for item j at the beginning of period 1(= 0 otherwise).

Decision variables (Model's output):

$y_{jt} \in \{0,1\}$ 1, if the machine is set up for item j in period t(= 0 otherwise).

Mathematically, the DLSP can now be specified as a mixed-integer programming model [6]:

$$\min \sum_{j=1}^{J} \sum_{t=1}^{T} (h_j I_{jt} + s_j x_{jt}) \tag{2.5}$$

Subject to:

(2.2)

$$p_j q_{jt} = C_t x_{jt} \qquad \forall t, j \tag{2.6}$$

$$\sum_{j=1}^{J} y_{jt} \leq 1 \qquad \forall t \tag{2.7}$$

$$x_{jt} \geq y_{jt} - y_{jt-1} \qquad \forall t, j \tag{2.8}$$

The objective function and inventory balance constraints are equal to the CLSP. The 'all-or-nothing' assumption comes in via Constraints (2.6), where in contrast to the CLSP the left and the right-hand side must be equal, so only the full capacity can be produced. Constraints (2.7) make sure that at most one item can be produced per period. The beginning of a new lot is spotted by the inequalities (2.8).

A step towards more realistic situations comes with the continuous setup lot-sizing problem (CSLP) proposed by [9]. In this model the 'all-or-nothing' assumption is given up and production quantities can now be of any continuous size. As a result, Constraints (2.7) are modified to allow production quantities smaller than the micro-period capacity:

$$p_j q_{jt} \leq C_t x_{jt} \qquad \forall t, j \tag{2.9}$$

A weakness of the CSLP model is that if the capacity is not used in full, the remaining is left unused. An attempt to avoid this is a variation of the model called the proportional lot-sizing and scheduling problem (PLSP) described by [10]. The basic idea of the PLSP is to use the remaining capacity for scheduling a second item. If it decides to have 2 items produced in a period, it must be clear in which order they are produced. To accomplish this, the setup state variables y_{jt} is now the state at the end of the period t. Production in a period may take place if the machine is properly set up for the product either at the beginning or at the end of the period, therefore only two items can be produced per period, expressed in Constraints (2.10).

$$p_j q_{jt} \leq C_t (y_{j,t-1} + y_{jt}) \qquad \forall t, j \tag{2.10}$$

Since more than one product can be produced in one micro-period, the capacity restriction (2.2) need to be replaced by:

$$\sum_j p_j q_{jt} \leq C_t x_{jt} \qquad \forall t, j \tag{2.11}$$

2.1.3 Integrated problem

[11] presents different approaches to model the integration of lot-sizing and scheduling decisions. Here we present the most standard model, called the general lot-sizing and scheduling problem (GLSP).

The GLSP combines macro-periods and an approach to sequence the lots using micro-periods. The term *General* is based on the fact that several well-known models for lot-sizing and scheduling integration differ from GLSP only by some additional constraints. Despite allowing for a very accurate modeling of the problem, this model is computationally hard to solve, whereas the big and small bucket models are much easier to tackle as pointed by [4]. The standard GLSP mixed-integer programming model [12] is shown next:

Data:

$j = 1, ..., J$	Number of products.
$t = 1, ..., T$	Number of macro-periods.
S_t	Set of micro-periods s belonging to macro-period t.
p_j	Production time of product j.
Cap_t	Production capacity (time units) available in macro-period t.
m_j	Minimum lot-size of product j.
M_j	Maximum lot-size of product j.
h_j	holding costs of product j.
sc_{ij}	Setup costs of changeover from product i to j.
st_{ij}	Setup times of changeover from product i to j.
d_{jt}	Demand of product j in macro-period t.
I_{j0}	Initial inventory of product j at the beginning of the planning horizon.
y_{j0}	equal 1, if the machine is set up for product j at the beginning of the planning horizon. (otherwise=0).

Variables (Model's output):

$I_{jt} \geq 0$	Inventory of product j at the end of macro-period t.
$x_{js} \geq 0$	Quantity of item j produced in micro-period s.
$y_{js} \in \{0, 1\}$	1, if the machine is setup for product j in micro-period s (= 0 otherwise).
$T_{ijs} \in \{0, 1\}$	1, if a changeover from product i to product j takes place at the beginning of micro-period s (otherwise=0).

$$\min \sum_{j=1}^{J}\sum_{t=1}^{T} h_j I_{jt} + \sum_{i=1}^{J}\sum_{j=1}^{J}\sum_{t=1}^{T}\sum_{s\in S_t} sc_{ij} T_{ijs} \tag{2.12}$$

Subject to:

$$I_{jt} = I_{j,t-1} + \sum_{s\in S_t} x_{js} - d_{jt} \qquad \forall t, j \tag{2.13}$$

$$\sum_{j=1}^{J}\sum_{s\in S_t} p_j x_{js} + \sum_{j=1}^{J}\sum_{i=1}^{J}\sum_{s\in S_t} st_{ij} T_{ijs} \le Cap_t \qquad \forall t \tag{2.14}$$

$$x_{js} \le M_{it} y_{js} \qquad \forall t, j, s \in S_t \tag{2.15}$$

$$\sum_{j=1}^{J} y_{js} = 1 \qquad \forall t, s \in S_t \tag{2.16}$$

$$x_{js} \ge m_j (y_{js} - y_{j,s-1}) \qquad \forall t, j, s \in S_t \tag{2.17}$$

$$T_{ijs} \ge y_{i,s-1} + y_{js} - 1 \qquad \forall t, j, i, s \in S_t \tag{2.18}$$

The objective function (2.12) minimizes holding and change-over costs. Constraints (2.13) give the inventory balances and ensure that the demand is met. Capacity constraints are given by (2.14), and (2.15) express that production can only take place if the machine is set up for the respective product. Constraints (2.16) assure that one and only one setup state is defined in each micro-period. Since the length of the micro-period s is determined by the time consumption of the production quantity x_{js} (where j is the product whose state indicator y_{js} is set to 1), micro-periods with zero length are allowed. Therefore, in Constraints (2.17), minimum lotsizes are introduced in order to avoid setup state changes without production. Constraints (2.18) establish the connection between setup state and changeover indicators indicating that a setup is not needed if the same product is to be produced in consecutive micro-periods.

[11] also present a small variation to the GLSP, a model designed by [13] (CC), in which decision variables y are dropped and the variables T are imposed to account for the changeovers and setup state in each micro-period. [11] show that the CC formulation is stronger than the original GLSP formulation. The CC model is then represented as the following:

$$\min \sum_{j=1}^{J}\sum_{t=1}^{T} h_j I_{jt} + \sum_{i=1}^{J}\sum_{j=1}^{J}\sum_{t=1}^{T}\sum_{s\in S_t} sc_{ij} T_{ijs} \tag{2.19}$$

Subject to:

(2.12) – (2.13)

$$\sum_{j=1}^{J}\sum_{i=1}^{J} T_{ji0} = 1 \tag{2.20}$$

$$x_{is} \leq M_j \sum_{j=1}^{J} T_{jis} \qquad \forall t, i, s \in S_t \tag{2.21}$$

$$\sum_{j=1}^{J} T_{ji,s-1} = \sum_{j=1}^{J} T_{ijs} \qquad \forall t, i, s \in S_t \tag{2.22}$$

$$x_{is} \geq m_j \sum_{j\neq i}^{J} T_{jis} \qquad \forall t, i, s \in S_t \tag{2.23}$$

In this formulation, the objective function is the same as the GLSP, as well as constraints (2.13) and (2.14) for inventory balances and capacity restrictions. Constraint (2.20) defines the initial setup state of the machine. (2.21) guarantee that production of a given product only occurs if the machine is set up at the micro-period. Flow constraints (2.22) keep track of changeovers and machine configuration state. Minimum lot-sizes Constraints (2.23) are again imposed.

2.1.4 Sequence-oriented

A different approach to model the sequencing of the production, proposed by [14], is to use a collection of predefined sequences that establish the items and sequences to be produced. Some extra parameters are then needed for a given sequence s:

S_t	Set of available sequences to schedule products on the machine in period t
$\widehat{sc}_s \geq 0$	setup cost incurred if sequence s is selected.
$\widehat{st}_s \geq 0$	setup time incurred if sequence s is selected.
$g_{is} \in \{0,1\}$	= 1 if product i is present in sequence s.
$f_{is} \in \{0,1\}$	= 1 if product i is first in sequence s.
$l_{is} \in \{0,1\}$	= 1 if product i is last in sequence s.

It is also necessary to have another decision variable to decide if sequence s is selected for production:

$W_s \in \{0,1\}$ = 1 if sequence s is selected for production.

The subproblem relative to the formulation of the predefined sequences can be solved with a metaheuristic method using local search strategies. Another solution method is proposed by [11], solving the subproblem as a price collecting traveling salesman [15]. A network is created that consist of a set of nodes, each representing a product, and arc sets representing the production sequence of this products. This method is described with more detail by [11] in Appendix B.

The model proposed by [14] is then represented as the following:

$$\min \sum_{j=1}^{J} \sum_{t=1}^{T} h_j I_{jt} + \sum_{s \in S_t} \widehat{sc}_s W_s \tag{2.24}$$

Subject to:

(2.13)

$$\sum_{j=1}^{J} p_j x_{jt} + \sum_{s \in S_t} \widehat{st}_s W_s \leq Cap_t \quad \forall t \tag{2.25}$$

$$\sum_{s \in S_t} W_s = 1 \quad \forall t \tag{2.26}$$

$$\sum_{s \in S_t} f_{js} W_s = \sum_{s \in S_t} l_{js} W_s \quad \forall j,t \tag{2.27}$$

$$x_{jt} \leq M_j \sum_{s \in S_t} g_{js} W_s \quad \forall j,t \tag{2.28}$$

Objective function (2.24) minimizes the total expenditure in holding costs and setup costs incurred from sequence selection. Constraints (2.13) represent the classical inventory balances. Capacity constraints are expressed in (2.25). The use of one sequence in each macro-period is ensured by (2.26). Constraints (2.27) guarantee setup carry-over by linking the first and last products of consecutive time periods. Lastly, Constraints (2.28) only allows production for products in the sequence selected in period t.

2.2 Heuristics and metaheuristics

In this section, solution methods based on heuristics and metaheuristics are reviewed. This methods can be applied in the integrated lot-sizing and scheduling problem. We present their functionality and how they can be applied.

2.2.1 Heuristics

Here firstly we describe two heuristics, Relax-and-fix (RF) with Fix-and-Optimize(FO), and how they can be employed to solve MIP models. Then it is presented the local search heuristic that works as base to some metaheuristics that will be shown next.

2.2.1.1 Relax-and-fix (RF)

The Relax-and-fix method is a construction heuristic used in the resolution of mixed-integer problems, which defines an initial solution by solving several small MIP models. Initially, all binary variables in the RF solution are relaxed which means they can take any value between 0 and 1. Then a set of variables X, according to a window size Ws defined, are forced to be integer, while the others are kept relaxed, and the resulting MIP is then solved. Next, the X variables are fixed with the results and another set of integer variables are optimized. This process is repeated until all variables are fixed [16].

This method was applied in a animal feed industry by [17], mentioned before, where the results are discussed and compared with the classic MIP solvers and with the current company's planning strategy.

2.2.1.2 Fix-and-optimize (FO)

The fix-and-optimize heuristic uses another approach to resolve MIP models. It also operates in an iterative fashion to solve a series of sub-problems that are derived from the main MIP model. In each iteration, most binary variables are set to a fix value and the resulting sub-problem is then solved by a MIP solver. A different set of binary variables are left "free" to optimize in every iteration until all the variables are optimized [18].

2.2.1.3 Local search

Local Search (LS) is one of the oldest and simplest heuristics method. It starts at a given initial solution and at each iteration the algorithm replaces the current solution by a neighbor solution. A neighbor is generated by the application of an operator that performs a small perturbation to the current solution. Three methods can be used to choose the next solution:

- **Best improvement (Steepest ascent)**: All the neighbors are calculated and the best solution is chosen to replace the current one.
- **First improvement:** Neighbors solutions are generated until one is better than the current solution that then replaces it.
- **Random Selection:** A random neighbor is selected from those improving the current selection.

This search stops when all candidate neighbors are worse than the current solution, so a local optimum is reached. In the case that the objective function is a minimizing one, LS may be seen as a descent walk in the graph representing the search space. Next it is presented the pseudo-code for the steepest ascent LS variant.

Algorithm 1: LS (steepest ascent) pseudo-code.

```
   Data: s = $s_0$; /* InitialSolution */
1  while (Termination Criteria no satisfied) do
2      Generate (N(s));         /*Generation of candidate neighbors*/
3      if (No better neighbor) then
4          Stop;
5      else
6          s = s';          /* Best neighbor s' */
7      end
8  end
9  Output: Final solution, local optima
```

In general, LS is an easy method to design and implement, but one of its main disadvantages is that it normally converges toward local optimal solutions. Therefore more complex methods, presented below, are needed to avoid that the algorithm gets trapped in local optima.

2.2.2 Metaheuristics

Metaheuristics are a higher-level procedure designed as strategies to guide a search process that may provide a sufficiently good solution to an optimization problem. Metaheuristics work with a set of solutions which is usually too large to be completely sampled.

An analysis on the various methods was made by [19], among the metaheuristics discussed are Tabu Search; Simulated Annealing; Genetic Algorithms; Memetic Algorithms that will be described below in more detail. The review of the metaheuristics in this section is based on [9].

2.2.2.1 Tabu search

Tabu search (TS) is a method to solve optimization problems, firstly proposed by [20]. It works as a steepest ascent LS algorithm but it uses a list to store past solutions, it also accepts non-improving solutions to escape from local optima when all the neighbors are worse than the current solution.

To avoid cycles, TS discards the neighbors that have been previously visited, managing a memory of the solutions recently applied, which is called *tabu list*. The tabu list may be too restrictive so an *aspiration criteria* is created in a way that tabu solutions can eventually be accepted.

Then the admissible solutions are the non-tabu ones or that hold the aspiration criteria. The TS pseudo-code is presented next:

Algorithm 2: TS pseudo-code.

```
  Data: s = s_0; /*InitialSolution*/
1 Initialize the tabu list;
2 while (Stopping criteria not satisfied) do
3     Find best admissible neighbor s';
4     s = s';
5     Update tabu list, aspiration conditions;
6 end
7 Output: Best solution found
```

2.2.2.2 Simulated annealing

The simulated annealing (SA) concept, applied to optimization problems, was first introduced by [21]. SA is based on the principles of statistical mechanics, where the annealing process requires heating and then slowly cooling to obtain a strong crystalline structure. This analogy is represented in the Table 2.1.

Table 2.1: Analogy between the physical system and the optimization problem.

Physical System	Optimization Problem
System state	Solution
Molecular positions	Decision variables
Energy	Objective function
Ground state	Global optimal solution
Metastable state	Local optimum
Rapid quenching	Local search
Temperature	Control parameter
Careful annealing	Simulated annealing

The algorithms works using an random LS strategy. At each iteration a random neighbor is generated, and moves that improve the actual solution are always accepted. However in SA, if the neighbor is not better it can be selected with a given probability (2.29) that depends on the current temperature (T) and the difference to the actual solution (ΔE). If the starting temperature (T_0) is very high the search will be like a random LS. Otherwise, if it is very low, it will behave like a first improvement variant LS. Hence, a balance is needed between these two extreme procedures.

$$P(\Delta E, T) = e^{-\frac{\Delta E}{T}} \tag{2.29}$$

Acceptance probability function: Main element of SA that enables non-improving neighbors to be selected.

The cooling schedule: Defines the temperature at each step of the algorithm. Essential to the efficiency and effectiveness of the algorithm.

The following algorithm describes the SA search process:

Algorithm 3: SA pseudo-code.

```
Data: s = $s_0$ /* Initial Solution*/
T = $T_{max}$ /* Start temperature*/
1  while (stopping criteria not satisfied ( $T < T_{min}$ )) do
2      while (Equilibrium condition not satisfied (fixed temperature)) do
3          Generate random neighbor s';
4          $\Delta E$ = f(s') - f(s);
5          if ($\Delta E < 0$) then
6              s = s' /*accept the neighbor solution*/
7          else
8              Accept s' with probability ($e^{-\frac{\Delta E}{T}}$);
9          end
10     end
11     T = g(T); /*Temperature update*/
12 end
13 Output: Best solution found
```

2.2.2.3 Genetic algorithms (GA) methods

Genetic algorithms are a family of computational methods inspired by the evolution theory. These algorithms encode a potential solution to a specific problem on a simple chromosome-like data structure, and apply recombination operators to these structures in order to optimize the solution.

Five main phases are considered in a standard genetic algorithm [22]:

Initial Population: The process begins with a set of individuals which is called a population. Each individual represents a solution to the problem and it is generated randomly.

Fitness score: The fitness function determines how good a solution is. It gives a fitness score to each individual. The probability that an individual will be selected for reproduction is based on its fitness score.

Selection: The idea of selection phase is to select the fittest individuals and let them pass their genes to the next generation. Two pairs of individuals (parents) are selected based on their fitness scores. Individuals with high fitness have more chance to be selected for the crossover.

Crossover: Crossover is the most significant phase in a genetic algorithm. For each pair of parents to be mated, a crossover point is chosen at random from within the genes. The individual is created by exchanging the genes of the parents among themselves using a crossover point. That new solution is then added to the population.

Mutation: In certain new individuals formed, some of their genes can be subjected to a mutation with a low probability. Mutation occurs to maintain diversity within the population and prevent premature convergence.

The pseudo-code algorithm with all this phases is presented below:

Algorithm 4: GA pseudo-code [23].

```
   Data: Set pop-size, max-gen, gen = 0, cross-rate, mutate-rate
1  initialize population;
2  while $max_gen \geq gen$ do
3      evaluate fitness;
4      for $i \leftarrow 1$ to pop-size by 1 do
5          select(parent1, parent2);
6          if (random(0,1) $\leq$ cross-rate) then
7              child = crossover(parent1, parent2);
8          end
9          if (random(0,1) $\leq$ mutate-rate) then
10             child = mutation();
11         end
12     end
13 end
14 Output: Best solution found.
```

A variant of the GA are biased random-key genetic algorithms (BRKGA), introduced by [2], where one of the parents used for mating is biased to be of higher fitness, called the elite individuals, a p_e fraction of the population with the best solutions. The transition process from the previous generation to next in the BRKGA method is represented in the Figure 2.1. BRKGA also uses an *parameterized uniform crossover*, in which for each gene has the probability (1 - p_e) of inheriting the value from the elite parent instead of the non-elite one. In this way, the offspring is more likely to inherit characteristics of the best parent.

A BRKGA heuristic was presented by [24] for the production scheduling problem. The methods shown good results getting the best-known solution for 73 % of the instances. The BRKGA's flowchart is represented in the Figure 2.2, which is very similar to the standard GA.

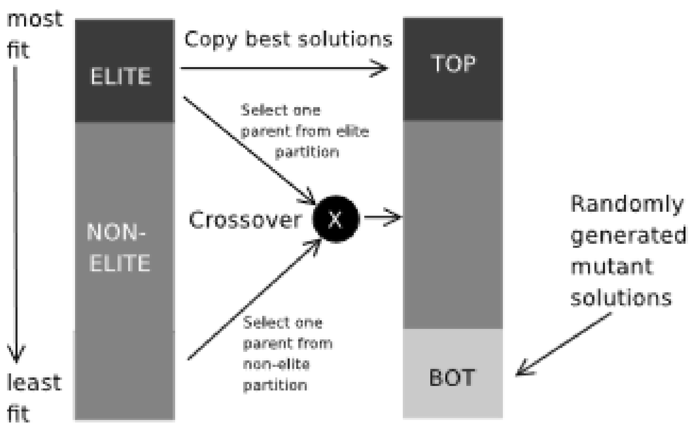

most fit
least fit
ELITE
NON-ELITE
Copy best solutions
Select one parent from elite partition
Crossover
X
Select one parent from non-elite partition
TOP
BOT
Randomly generated mutant solutions

Figure 2.1: Transition between generations in BRKGA [2].

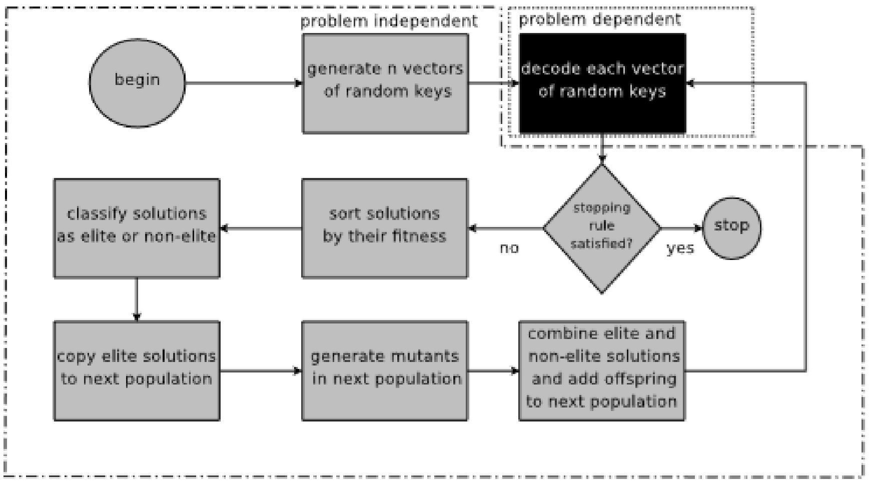

problem independent
problem dependent
begin
generate n vectors of random keys
decode each vector of random keys
stopping rule satisfied?
yes
no
stop
sort solutions by their fitness
classify solutions as elite or non-elite
copy elite solutions to next population
generate mutants in next population
combine elite and non-elite solutions and add offspring to next population

Figure 2.2: BRKGA's flowchart [2].

2.2.2.4 Memetic algorithm

The term 'memetic algorithms'(MAs) was introduced in the late 80s to define a family of meta-heuristics that have as central theme the hybridization of different algorithmic approaches for a given problem. The memetic algorithms can be viewed as a merge between a population-based global algorithm and a local search made by each of the individuals. They are a special kind of genetic algorithms with a local hill climbing.

In a memetic algorithm the population is initialized at random or using a constructive heuristic. Then, each individual makes a local search to improve its fitness. Like generic GA's, individuals with higher fitness are more likely to be selected to pass their genes to the next generation. The role of the local search in memetic algorithms is to locate the local optimum more efficiently then genetic algorithms, using less randomness.

Algorithm 5: Memetic algorithm pseudo-code [23].

```
  Data: Set pop-size, max-gen, gen = 0, cross-rate, mutate-rate
1 initialize population; while max-gen > gen do
2     apply GA;
3     apply local search;
4     gen = gen + 1;
5 end
6 apply final local search to best chromosome;
```

2.3 Lot-sizing and scheduling in the animal feed industry

This work also addresses the production planning of an animal feed company, in the studied case, the optimization problem occurs in planning the schedule for a mixer's use.

Product changes are frequent in this industry, typically about 30–40 per week, and can be grouped into several families. Products within the same family do not contaminate each other and have negligible changeover times and identical processing times. A complicating feature of the animal feed industry is that some product families can contaminate others if produced in successive batches so the production line must be cleaned, resulting in substantial setup time. Thus, the production scheduling in this industry, has the objective of minimizing the amount of cleanings necessary in the production plan.

2.3.1 Non-triangular setup times

Triangular sequence-dependent setup times occurs when it is always faster to perform the sequence from product p to r directly than via a third product q. However, in the animal feed and other industries, typically such cleanings can sometimes be avoided by introducing a single lot of an intermediate product from other family, between the two problematic products, since the triangular inequality does not hold. which is called non-triangular setup times [3]. This concept is represented in the Figure 2.3.

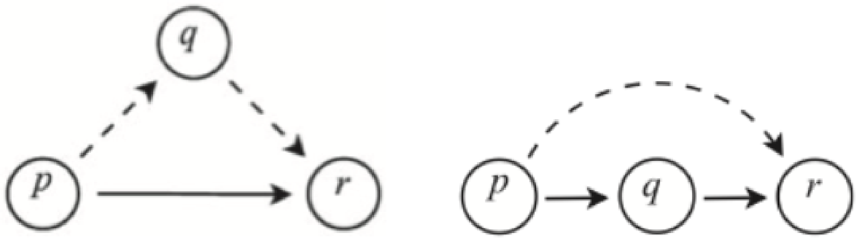


Figure 2.3: Triangular and non-triangular setup [3].

2.3.2 Integration problem in animal feed industries

[17] conducted a research on the production planning of a Brazilian animal feed compound company, that works with one mixer. To decide lot sizes and sequences in each period, two MIP models were designed based on the GLSP. The first for independent sequences, where a cleaning is made during non-productive time between periods, therefore not requiring initial setup in the beginning of the period. The second for depedent sequences where the production is active 24h a day, eliminating the non-productive time between periods. This last one being more complex since the sequence has to be optimized over multiple periods rather than over a single period.

Another approach was made by [25], where the production system studied was constituted by one mixer in the first stage, a pelletizer, an extruder and a bulk machine, with each one having one or more silos. The production planning sequences the batches on the mixer and also assigns each product to a silo. Here a MIP model is formulated as an extension of the GLSP.

A more extensive study on the integrated lot sizing and scheduling problem in the animal feed compound industry is presented by [26]. Using a case study in a company of the sector, two approaches are proposed to model and solve the problem. The first is based on GLSP with sequence dependent setup times. The other consists of modeling the lot sequencing problem as an asymetric travelling salesman problem (ATSP). For each method is proposed two company strategies related to the cleaning of the production line already mentioned before, the first for *Independent Sequences*, and the second with *Dependent Sequences* (setup carryover). The instances presented in the appendix of [26] will be used in a computational experiment later and their respective results compared to the method proposed.

Chapter 3

Problem Definition

The objective of this chapter is to provide a clear definition on the problem that is addressed by this thesis. First, all the parameters needed for the lot-sizing and scheduling integration problem are presented, focusing then on the specific case of an animal feed industry.

3.1 Integrated production planning

The integrated production planning considered in this work consists of a set of N products, and a planning horizon of T macro-periods, each one containing N micro-periods. Every product has a production time that, alongside the setup times calculated, have to respect the capacity(time) of each macro-period. The main decision of the planning is to decide the production quantities X_{js} for each micro-period s, in order to answer the demand that is represented with a matrix N×T that has the needs d_{jt} for every product j in macro-period t. The setup costs and time values, between each product, are represented by a matrix N×N.

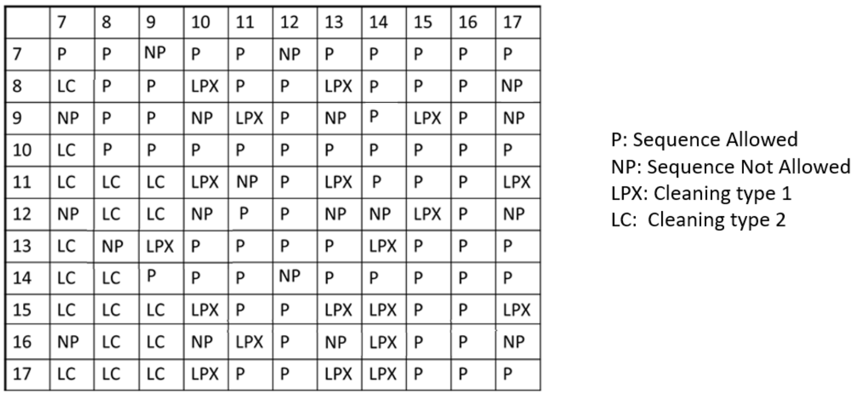
Table 3.1 presents an example of a production plan solution for a timespan of 5 days, considering 5 distinct families (Fam) of products and showing the quantity (Quant) to be produced for each product.

Table 3.1: Example of a Production Plan.

Order	Period 1		Period 2		Period 3		Period 4		Period 5	
	Fam	Quant	Fam	Quant	Fam	Quant	Fam	Quant	Fam	Quant
1	2	10	4	8	1	4	4	5	5	6
2	5	3	2	4	3	12	2	8	4	1
3	3	8	5	6	2	5	1	5	2	1
4	1	5	3	2	5	1	3	1	3	10
5	4	7	1	9	4	2	5	8	1	7

3.1.1 Animal feed industry problem

As mentioned before, in the case of an animal feed industry the setups are constant since they always represent a cleaning that is necessary if two products produced consecutively can contaminate each other. The following figure represents this setups costs in a framework with 11 different family products:



	7	8	9	10	11	12	13	14	15	16	17
7	P	P	NP	P	P	NP	P	P	P	P	P
8	LC	P	P	LPX	P	P	LPX	P	P	P	NP
9	NP	P	P	NP	LPX	P	NP	P	LPX	P	NP
10	LC	P	P	P	P	P	P	P	P	P	P
11	LC	LC	LC	LPX	NP	P	LPX	P	P	P	LPX
12	NP	LC	LC	NP	P	P	NP	NP	LPX	P	NP
13	LC	NP	LPX	P	P	P	P	LPX	P	P	P
14	LC	LC	P	P	P	NP	P	P	P	P	P
15	LC	LC	LC	LPX	P	P	LPX	LPX	P	P	LPX
16	NP	LC	LC	NP	LPX	P	NP	LPX	P	P	NP
17	LC	LC	LC	LPX	P	P	LPX	LPX	P	P	P

P: Sequence Allowed
NP: Sequence Not Allowed
LPX: Cleaning type 1
LC: Cleaning type 2

Figure 3.1: Animal feed setups matrix.

The animal feed industry contacted during the development of this thesis, works with an additional parameter where each product, in addition to being part of a family with possible setup costs with others, also has a set of 9 attributes. This attributes can lead to cleanings as well, according to a matrix 9×9. So both setups sources need to be combined to produce the full setup costs matrix for all the production products.

3.2 Model proposed for the case studied

The model proposed to represent the problem addressed in this study is based on the GLSP. However, it brings additional features not considered in the standard GLSP.

In the scenario studied the following additional costs are take into account, that are not presented in the GLSP models presented before:

- Backlog costs: possibility of having backlog, if the demand could only be answered after the deadline date.
- Overtime costs: the production capacity can be exceeded with additional costs associated

When the demand surpasses the production capacity the model has the objecive to find the right balance between overtime and backlog costs.

Adding this new properties to the original GLSP, a new version of the model was constructed, presented below. This will serve as a reference to test the data used and compare the results with the algorithms that were developed in this thesis.

Data:

$Sets:$	
$i,j = 1,.....,P$	Number of products
$t = 1,.....,T$	Number of macro-periods
$r = 1,.....,R$	Number of raw-materials
S_t	Set of micro-periods s belonging to macro-period t.
$InitialState:$	
$I0_j$	Initial inventory of product j
$B0_j$	Initial backlog of product j
$Time:$	
C_t	Production capacity of period t
P_j	Production time of product j
s_{ij}	Setup time in a changeover from product i to product j
o_t	Overtime limit in period t
$Quantity:$	
m_j	Minimum lot size of product j
M_{jt}	Maximum lot size of product j in period t
d_{jt}	Demand of product j in period t
$Costs:$	
sc_{ij}	Costs of setup change from product i to product j
h_j	Holding costs of product j
bc_j	Backlog costs of product j
oc_t	Overtime costs on period t

Decision variables:

$x_{js} \geq 0$	Production of product j in micro-period s
$O_t \geq 0$	Overtime used in period t
$I_{jt} \geq 0$	Inventory of product j in the final of period t
$B_{jt} \geq 0$	Backlog of product j in the final of period t
$z_{ijs} \in \{0,1\}$	Changeover from product i to product j in micro-period s
$y_{js} \in \{0,1\}$	Setup ready for product j in micro-period s

$$\min \sum_{j=1}^{J}\sum_{t=1}^{T} h_j I_{jt} + \sum_{j=1}^{J}\sum_{t=1}^{T} B_{jt} bc_j + \sum_{t=1}^{T} O_t oc_t + \sum_{i=1}^{J}\sum_{j=1}^{J}\sum_{t=1}^{T}\sum_{s=1}^{S_t} sc_{ij} z_{ijs} \tag{3.1}$$

Subject to:

$$I_{jt-1} + B_{jt} + \sum_{s=1}^{S_t} x_{js} = I_{jt} + B_{jt-1} + d_{jt} \qquad \forall j,t \tag{3.2}$$

$$B_{j0} = B0_j \qquad \forall j \tag{3.3}$$

$$I_{j0} = I0_j \qquad \forall j \tag{3.4}$$

$$\sum_{j=1}^{J}\sum_{s=1}^{S_t} x_{js} P_j + \sum_{i=1}^{J}\sum_{j=1}^{J}\sum_{s=1}^{S_t} z_{ijs} s_{ij} \leq C_t + O_t \qquad \forall t \tag{3.5}$$

$$O_t \leq o_t \qquad \forall t \tag{3.6}$$

$$x_{js} \leq M_{jt} y_{js} \qquad \forall t,s,j \tag{3.7}$$

$$\sum_{j=1}^{J} y_{js} = 1 \qquad \forall t, s \in S_t \tag{3.8}$$

$$\sum_{j=1}^{J} z_{ijs} = y_{is-1} \qquad \forall t, s \in S_t, j \tag{3.9}$$

$$\sum_{i=1}^{J} z_{ijs} = y_{js} \qquad \forall t, s \in S_t, j \tag{3.10}$$

$$x_{js} \geq m_j (y_{js} - y_{js-1}) \qquad \forall t, s \in S_t, j \tag{3.11}$$

The objective function (3.1) consists in minimizing the holding costs of the products, the backlog of the demands, the overtime and the setup costs for the changeover between products. Inventory balances are made in Constraints (3.2). Initial inventory and backlog are set in Constraints (3.3) and (3.4) respectively. Production capacity constraints are represented in Constraints (3.5). The limit of overtime are defined in Constraints (3.6) and in (3.7) it is ensured that the setup is in the right state to produce a product. Constraints (3.8) establishes that the mixer can only have one setup state. Constraints (3.9) and (3.10) are needed establish changeovers between micro-periods. Lastly, Constraints (3.11) force that something is produced when a change of products is made.

Chapter 4

Iterative method

In this chapter a novel iterative solution method is detailed. Firstly, we show why the GLSP model is intractable to solve large scale instances. Then the iterative algorithm conceived is detailed. It consists of two phases, the tactical level and the operational level. In each iteration, feedback based on the past solutions is used from the previous iterations to construct new constraints. These new constraints are added to the tactical level model and are based on capacity used and setup costs, later in the chapter these restrictions formulation is detailed.

4.1 Computational intractability of the GLSP model

The GLSP variant model presented in Chapter 3 is able to solve integrated lot-sizing and scheduling problem, however if it has a significant number of macro-periods and different families of products, the amount of constraints and variables of the MIP model increases exponentially. As pointed by [6], this problem is NP-hard, since it cannot be solved in polynomial time. The amount of binary variables z_{ijs} necessary to represent the changeover from one product to another in a micro-period, is equal to $N \times T \times N \times N$, leading to $2^{N \times T \times N \times N}$ possible solutions, hence the number of solutions increases exponentially with the number of different products and periods.

To portray this issue we present three different examples with low, medium and high number of parameters (N products and T macro-periods) and the respective number of integer values (V) and constraints(C) for the standard GLSP represented in Section 6.2. The resulting running time for each example is then showed, using the same computational power conditions.

- N = 5, T = 3 $\rightarrow$ Constraints = 558 Variables = 540 $\Rightarrow$ Running time = 0.66 sec
- N = 10, T = 3 $\rightarrow$ Constraints = 3663 Variables = 3630 $\Rightarrow$ Running time = 111.87 sec
- N = 15, T = 5 $\rightarrow$ Constraints = 19280 Variables = 19200 $\Rightarrow$ Running time = 1091.48 sec

The exponential number of constraints and variables of the GLSP model, makes it highly intractable, which becomes impractical for the companies to use this model for cases with several products and long planning horizons.

This comes from the fact that integrating the scheduling with the lot-sizing problem substantial increases the complexity of the problem. It is easy to see that if you only take into account the lot-sizing problem, the running time and the number of binary variables decrease substantially as showed below:

- N = 5, T = 3 $\rightarrow$ Constraints = 33 Variables = 45 $\Rightarrow$ Running time = 0.05 sec
- N = 10, T = 3 $\rightarrow$ Constraints = 63 Variables = 90 $\Rightarrow$ Running time = 0.18 sec
- N = 15, T = 5 $\rightarrow$ Constraints = 155 Variables = 225 $\Rightarrow$ Running time = 0.58 sec

This will be the base for the idea of the iterative algorithm.

4.2 Introduction to the ILSP

The solution method proposed to solve the integrated lot-sizing and scheduling problem consists on an iterative lot-sizing and scheduling planning (ILSP). This approach is based on the iterative strategy, represented in Figure 4.1, already discussed in Section 1.1.2.

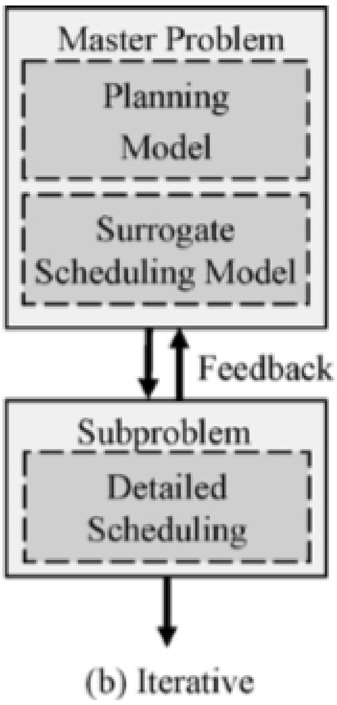


Figure 4.1: Iterative strategy for production planning and scheduling [1].

We consider that this intuitive phased algorithm would be easy to implement in large production company's with significant sequence-dependent setups costs.

The method consists on solving both lot-sizing and scheduling separately and, with the additional information of the solution found, start a new iteration. This is repeated until the solution converges or can no longer be improved.

Following the conclusions made before, only the tactical level is solved first, since it requires far less computational time, using a big-bucket linear programming model. The lot-sizes to be produced are then transmitted to the operational level, where a scheduling optimization is executed, for this, phase local search strategies are used, since, having the products to be produced and their respective setup costs defined, it becomes a simple optimization problem. The solution found will result in feedback for the next iteration of the tactical level.

This feedback is made using two parameters. Adding the setup times, the new capacity used is analyzed and, if it now surpasses the company's limit, that information is saved for the next tactical level iteration. If expensive setup costs are inevitable after the operational level, a new cost to the products creating them are added to next tactical level iteration.

This cycle was defined to be stopped when 5 consecutive iterations fail to improve the current best solution.

The ILSP algorithm is represented in Figure 4.2:

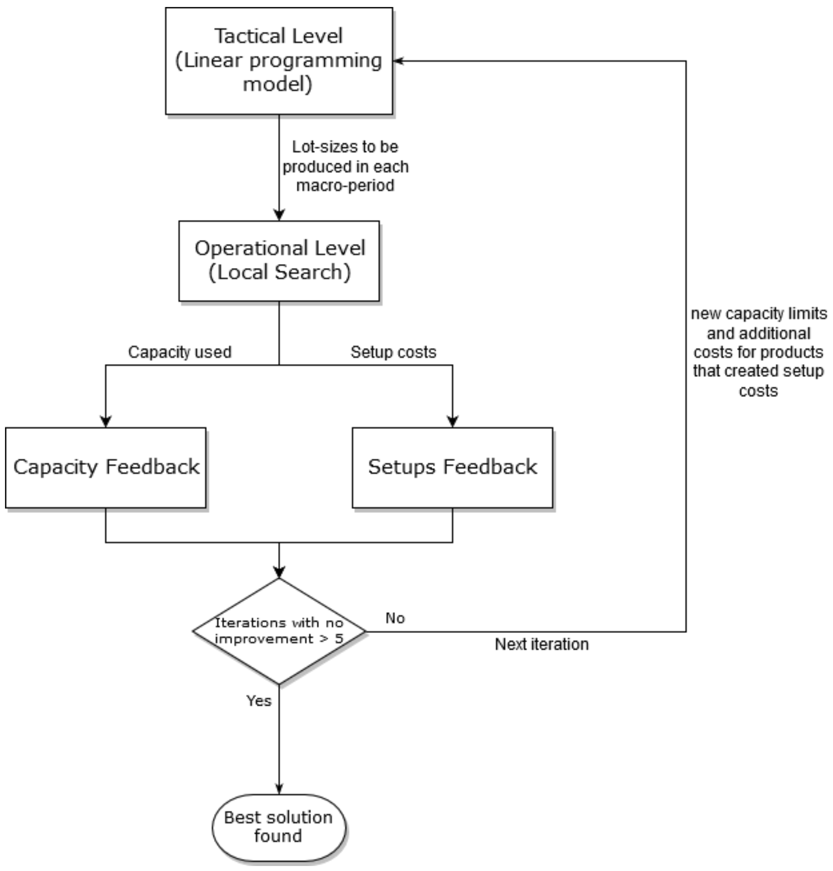

Tactical Level (Linear programming model)
Lot-sizes to be produced in each macro-period
Operational Level (Local Search)
Capacity used
Setup costs
Capacity Feedback
Setups Feedback
Iterations with no improvement > 5
No
Next iteration
Yes
Best solution found
new capacity limits and additional costs for products that created setup costs

Figure 4.2: ILSP method.

4.3 Tactical level

Firstly, the ILSP uses a simple linear programming model, represented below, to solve the tactical planning sub-problem. This tactical model (ILSP-Tactical) is based on the CLSP model previously mentioned in section 4.3. In the model it is considered the possibility of having backlog and overtime.

Data:

$j = 1,.....,J$	Number of products
$t = 1,.....,T$	Number of periods
C_t	Capacity (time) available in period t.
O_t	Overtime (time) available in period t.
p_j	Capacity consumption (time) needed to produce one unit of product j.
h_j	Non-negative holding costs of product j.
oc_t	Overtime costs for product family j.
bc_j	Backlog costs for product family j.
d_{jt}	Demand of product j in period t.
I_{j0}	Initial inventory of product j at the beginning of the planning horizon.

Decision variables (Model's output):

$I_{jt} \geq 0$	Inventory of product j at the end of period t.
$B_{jt} \geq 0$	Backlog of product j at the end of period t.
$Ot_t \geq 0$	Overtime used in period t.
$q_{jt} \geq 0$	Production quantity of item j produced in period t.

$$\min \sum_{j=1}^{J} \sum_{t=1}^{T} (I_{jt} h_j + B_{jt} bc_j) + \sum_{t=1}^{T} Ot_t oc_t \tag{4.1}$$

Subject to:

$$I_{jt-1} + B_{jt} + q_{js} = I_{jt} + B_{jt-1} + d_{jt} \qquad \forall j,t \tag{4.2}$$

$$\sum_{j=1}^{J} p_j q_{jt} \leq C_t + Ot_t \qquad \forall t \tag{4.3}$$

$$Ot_t \leq O_t \qquad \forall t \tag{4.4}$$

For the ILSP-Tactical, the objective function (4.1) takes into account the three possible costs in the production studied: holding Inventory, backlog and overtime. Constraints (4.2) represent the typical inventory balances combined with the backlog possibility. Capacity of each macro-period is controlled by constraints (4.3) and the maximum overtime is limited in Constraints (4.4).

For a simple production planning of 5 families of products and 3 macro-periods, a tactical level solution can be represented as:

Table 4.1: Example of tactical level solution.

Period 1		Period 2		Period 3	
Fam	Quant	Fam	Quant	Fam	Quant
1	10	1	8	1	4
2	3	2	4	2	12
3	8	3	6	3	5
4	5	4	2	4	1
5	7	5	9	5	2

4.4 Operational level

After obtaining an tactical solution, the next step of the ILSP is to optimize the production sequence. First, a good initial solution is generated and then local search techniques are used to improve the solution. This process is represented in Figure 4.3.

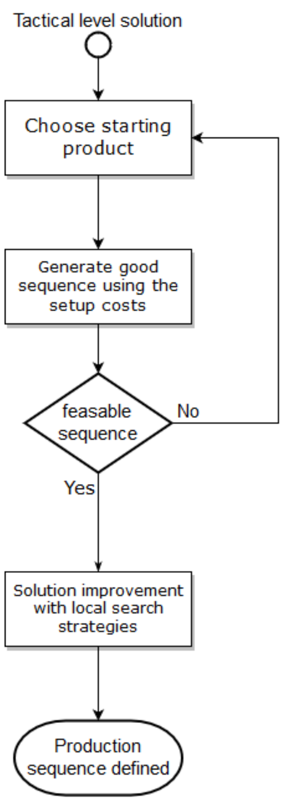

Tactical level solution
Choose starting product
Generate good sequence using the setup costs
feasable sequence
No
Yes
Solution improvement with local search strategies
Production sequence defined

Figure 4.3: Operational level algorithm.

4.4.1 Generate initial solution

Firstly, the algorithm 6 generates initial solutions for each macro period, considering a new parameter, $setups2_j$, that represents the sum of the setup costs of a product when it is the second in every setup pair. This parameter will be important to choose the products in the making of an initial good sequence solution.

The concept of the algorithm is that the product from the first period in the tactical level solution that requires the most setups, $setups2_j$, is chosen as the initial one, then the next product to be placed in the sequence is always the one that creates the less setup costs. This is repeated until the total production sequence is defined. Since in the cases addressed it is possible that two products can not ever be produced consecutively, a new restriction is necessary where, in the case that the last products in a period have this condition, the sequence creation is stopped. Then a different product from the tactical solution is chosen as the initial one to again start the algorithm.

This strategy was chosen in order to reduce the running time in the local search phase since it represents an already good starting solution to it and requires low running time.

Algorithm 6: Generate an operational solution pseudo-code.

```
Data:
1  $sc_{ij}$; /*Matrix of setup costs from product $i$ to $j$*/
2  $setups2_j$ = sum($sc_{ij}$); /* Sum of setup cost for product $j$*/
3  $plan_t$; /*Tactical plan with products for each period $t$*/
4  $op_{t,s}$; /*Operational plan with the sequence of the products for each period $t$ */
5  $p_{tried}$; /*list of products already tried as the initial product*/
6  while $(size(p_{tried}) <$ total number of products$)$ do
7      $op_{0,0} = j \in$plan$_t$ with max($setups2_j$) if $j$ not in $p_{tried}$; /* Choose the product that
         needs the more setups for the initial one in the first macro-period, that has not been
         tried yet*/
8      for $t \leftarrow 0$ to $T$ by $1$ do
9          for $s \leftarrow 0$ to $length(plan_t)$ by $1$ do
10             if (Every product left can't be produced after $op_{s-1}$) then
11                 $p_{tried}$.add($op_{0,0}$);
12                 break; /* Go the next iteration of the while cycle*/
13             else
14                 $op_{t,s} = j$ in min($sc_{p_{s-1}j}$) if $j \in$plan$_t$ with $plan_t$ and not in the solution
                     yet; /* the next product to be chosen is the one that creates the less
                     amount of setup costs for the previous product*/
15             end
16         end
17     end
18     break; /* The whole plan has been defined*/
19 end
20 Output: $op$ /* Sequence of production*/
```

For the special case of an animal feed industry addressed in this work, one part of the algorithm is modified. As mentioned before, in this production sector the setup costs represent cleanings, that are needed in the mixer if a product is produced after another specific type that contaminates it.

Therefore in line 14, instead of choosing the minimum setup cost, we choose one product that doesn't require cleaning if possible. However there may be various situations where there are several products still to place in the sequence that does not force a cleaning in the mixer. In order to better optimize the sequence, a strategy around the setup costs was chosen to decide which one of the products is to be placed. This is done using another parameter, $setups1_j$, that now represents the sum of the setup costs of a product when it is the first in every setup pair. The product chosen is the one that creates the necessity of cleaning in the less amount of products still to produce, min($setups1_j$). The goal is that more problematic types of products are placed at the end of the sequence, so less cleanings are needed and those products become easier to identify later to create the feedback information. Also being dependent sequences with setup carryover for the next period, the last mixer state, for a problematic product, can be passed by to the next period and avoid a cleaning, if that same product is to be produced in that period.

So line 14, used to decide the next product after a product *i*, is changed to:

$$op_{t,s} = j \in plan_t \text{ ,} with\ min(setups1_j) \qquad \forall j\ that\ sc_{op_{t,s-1},j} = 0$$

4.4.2 Local search heuristic

After that, to optimize the initial solution, a local search (LS) heuristic is used for each macro-period with the following characteristics:

- **Neighborhood:** For each iteration the neighbors of the current solution are evaluated. To create each one of the neighbors, two products positions in the sequence are swapped, demonstrated in Figure 4.4.
- **First improvement:** In each iteration the first neighbor that improves the best solution is chose as the new solution. This first improving technique was chosen, instead of the steepest descent one, since for extensive production problems, creating all the neighborhood for each iteration quickly becomes intractable.
- **Stopping criteria:** The local search ends when all the neighborhood solutions are worst than the current one so a local optimum was reached.

In Figure 4.4, it is demonstrated how the neighbours are defined. Each arrow represents a switch between the two products that originates a different scheduling solution. Using a simple example of a sequence with four products, the first one is swapped with every subsequent product after it, then the second is placed in every position ahead of it and so on consecutively. Therefore there is no duplicated position swaps in the neighborhood.

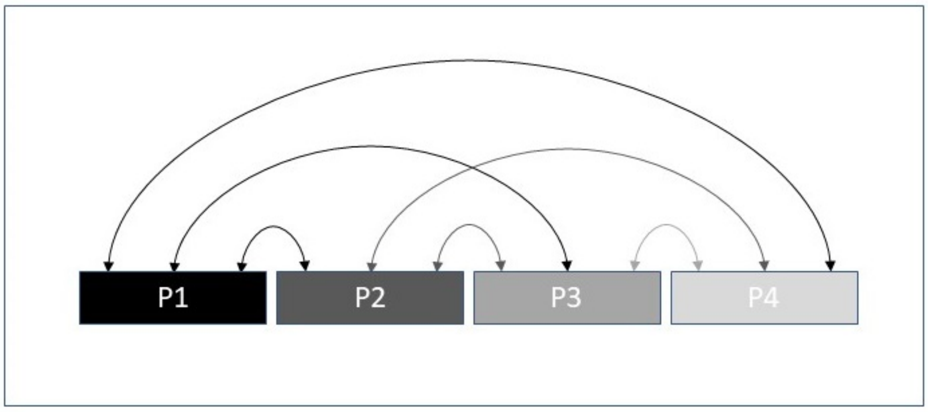


Figure 4.4: Neighborhood creation strategy in Local Search.

4.4.2.1 Non-triangular instances

In the case of non-triangular instances, an additional type of neighbor solutions are formulated in the local search heuristic. For each product, a single lot with the minimum production quantity allowed is placed in a different position in order to test if it is able to reduce the setup costs. This is possible in the case of non-triangular production setups, since high setup costs can be avoided by producing an intermediate product. This process is represented in Figure 4.5.

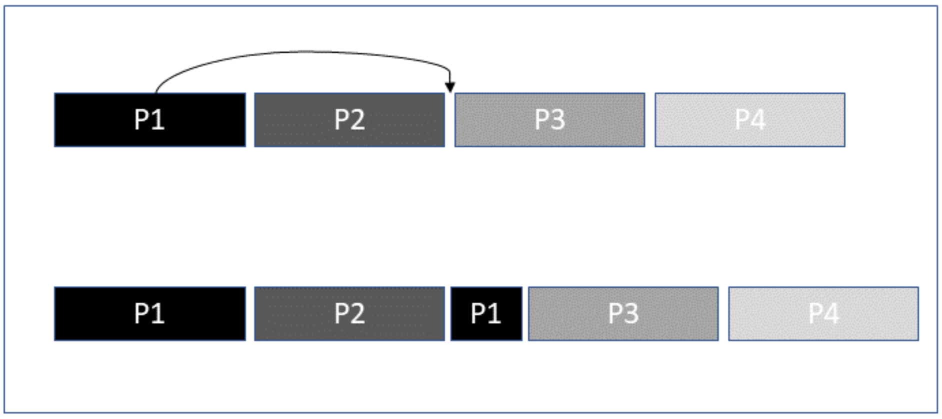


Figure 4.5: Additional neighbor creation strategy for non-triangular instances.

Having the example in Table 4.1 as input for operational level, the output of the algorithm is represented in Table 4.2.

Table 4.2: Example of operational level result.

Order	Period 1		Period 2		Period 3	
	Fam	Quant	Fam	Quant	Fam	Quant
1	2	10	4	8	1	4
2	5	3	2	4	3	12
3	3	8	5	6	2	5
4	1	5	3	2	5	1
5	4	7	1	9	4	2

4.5 Feedback from operational level to tactical

As mentioned before, the integration of both levels of the ILSP comes with a feedback given from the operational phase to the next iteration of the tactical phase. After the optimization of the sequence it is registered the capacity used and where in the plan still are found significant setup costs. This information are incorporated in the tactical level of the next iteration.

The operational level, at the end of the scheduling optimization, sends two type of feedback back:

- **Capacity:** Adding the setup times in the operational level, the amount of capacity time used above the limit or free time in each period is sent to the next tactical level iteration.
- **Setup costs:** The products that created significant setup costs, unavoidable during the operational level, have an additional cost in the next tactical level iteration.

The formulation of this tactical level inputs are described in detail in the next subsections.

4.5.1 Capacity feedback

Although the ILSP-tactical model presented in section 4.3 takes into account the capacity limits presented in constraints 4.3, it does not acknowledge the setup times needed since the sequence is not being defined. Therefore it is common that the model estimates that the capacity is respected but after the sequence is optimized, adding the setup times, that can no longer be true.

The capacity feedback is accomplished by changing the cap_t and ot_t limits in the ILSP-tactical model. Having the optimized sequence, initially the capacity time used for each period is calculated with the following formulation:

$$used_t = \sum_j p_j q_{jt} + \sum_i \sum_j st_{ij} z_{ijt} \qquad \forall t \tag{4.5}$$

Then it is registered where $used_t$ is in the capacity limits. There are 2 possible situations that could take place in each macro-period t:

- $used_t < cap_t$**:**

  If the capacity used is less than the limit in macro-period t, the new capacity in the next iteration of the tactical level is increased and the overtime boundary is decreased by the same amount.

  new $cap_t = cap_t + (cap_t - used_t)$
  new $ot_t = ot_t - (cap_t - used_t)$

- $used_t > cap_t$**:**

  If the capacity used exceeds the limit and requires overtime, the inverse of the previous case happens, with the new capacity and overtime being adjusted.

  new $cap_t = cap_t - (used_t - cap_t)$
  new $ot_t = ot_t + (used_t - cap_t)$

The goal of this approach is to adjust the capacity and overtime limits according to the results of the scheduling phase. The total time available in the tactical level doesn't change but as the portions of the standard capacity and the overtime vary, the model tends to put more products in periods with less overtime in order to avoid overtime costs.

4.5.2 Setup costs feedback

For the setup costs feedback two different approaches were developed. The first, $ILSP_{nR}$, allows the tactical phase to choose when to produce the products that were resulting in high setup costs. The second, $ILSP_R$ determines the macro-periods for the problematic products that reduce the setup costs and forces them to be produced at these periods in the tactical level.

4.5.2.1 $ILSP_{nR}$ - Feedback from setup costs without resolution

As mentioned before, at the end of the sequence optimization in the operational level it is possible setup costs could not be avoided. The Table 4.3 shows an example of an optimized sequence that still presents a cleaning between product 8 and 1.

Table 4.3: Example of a production sequence with a cleaning.

Day	Order										
2	4	7	1	9	4	2	5	8	cleaning	1	7

For this first proposed method, $ILSP_{nR}$, it is only analyzed in which periods t an inevitable expensive setup costs will occur for a product j. In productions where all changeovers require a setup cost, to determine this expensive setup costs after the scheduling optimization, all the setup costs are sorted and the biggest *hsc* are considered, where *hsc* is a small portion of the total number of setup costs. In productions with constant setup costs, as it is in the animal feed case analyzed with the cleanings process, all the setups costs after the operational level are considered.

To incorporate this information in the tactical level, a new production cost pc_{jt} is added to the ILSP-Tactical model for products j that present the considered *hsc* setups or cleaning in period t. This new parameter is updated at the end of every iteration if this big setup costs are found:

$$pc_{jt} = sc_{ij} \tag{4.6}$$

An additional variable y_{jt} is created that determines if product j is being produced in period t and the objective function is also updated to add this new costs:

$y_{jt} \in \{0,1\}$ 1, if product j is produced in period t(= 0 otherwise).

$$\min \sum_j \sum_t (I_{jt}h_j + B_{jt}bc_j) + \sum_t Ot_t oc_t + \sum_t \sum_j y_{jt} pc_{jt} \tag{4.7}$$

Therefore the ILSP-Tactical model can still place the problematic product in the same period if putting it in any one of the others creates more costs than the setup ones. But this strategy enables the model to acknowledge inevitable setup costs.

4.5.2.2 *ILSP$_R$* - Feedback from setup costs with resolution

The previous method only determines where the inevitable *hsc* setups are and gets that information to the Tactical Level in order for the model to have that costs into consideration.

One step forward for this strategy is the second method proposed, *ILSP$_R$*. It locates in which different macro-period the problematic product can be placed that will create the less costs and then forces the tactical level to follow that solution.

This is accomplished by using the local search based algorithm 7. The idea is that for every *hsc* setup or cleaning considered, the products that are causing the setup cost are moved to every possible position in other macro-periods sequences, creating new solutions. The evaluation of the solution also takes into account the capacity, holding inventory and backlog costs, since the products are being moved to new macro-periods and the ones that improve the initial sequence is stored in a list. Then a steepest ascent approach is employed, in which the best solution of the list replaces the initial solution, and the parameters of the setup avoided: product type, old macro-period and new macro-period are stored. This process is repeated for all the remaining *hsc* setup costs, considering the new best solution as reference, until all of them are resolved or no better solution is found.

Algorithm 7: Find solutions for high setup costs pseudo-code.

```
   Data:
1  $Nhsc \leftarrow$ number of $hsc$ setup costs found from $i$ to $j$ in macro-period $t$;
2  $sc$ setup cost with the parameters: $t$ period, $i$ first product and $j$ second product;
3  $osolution_{t,s}$; /*optimized sequence solution for each macro-period $t$*/
4  $nsolution_{t,s}$; /*new solution for each macro-period $t$*/
5  Lsolutions $\leftarrow$ empty list of new solutions, with the product and position changed;
6  $S_t$ Production sequence for macro-period t;
7  while $Nhsc > 0$ do
8      for $sc \leftarrow 0$ to $Nhsc$ by 1 do
9          for $t \leftarrow 0 \neq sc_t$ to $T$ by 1 do
10             for $s \leftarrow 0$ to $S_t$ by 1 do
11                 $nsolution = osolution$
12                 delete $nsolution_{sc_t,sc_s}$ /* Take out product $j$ from the period $sc_t$ where he is
                     creating a $hsc$ setup cost */
13                 $nsolution_{t,s} = sc_j$; /* Put product $j$ from period $sc_t$ in period $t$ and position $s$
                     */
14                 if $(nsolution.fitness < osolution.fitness)$ then
15                     Add [$nsolution,j,sc_t,t$] to Lsolutions;
16             end
17         end
18     end
19     if $lenght(Lsolutions) = 0$ then
20         break; /*None of the products position movements improved the solution */
21     Store $j$,$sc_t$ and $t$ for the best solution in Lsolutions;
22     $osolution$ = best solution in Lsolutions; /* the new solution where the remaining $hsc$
         setup costs are going to be analysed is the best from the previous iteration*/
23 end
24 Output: Setup costs resolved.
```

In the end of the algorithm, the information stored for each product position moved is passed on to the tactical level. For this it is introduced two new variables in the ILSP-Tactical model:

$maxq_{jt}$	maximum quantity of product j that can be produced in period t.
$minq_{jt}$	minimum quantity of product j that needs to be produced in period t.

And an additional constraint that controls the production quantities:

$$minq_{jt} \leq q_{jt} \leq maxq_{jt} \quad \forall j,t \tag{4.8}$$

The first variable, $maxq_{jt}$, is used to prevent the ILSP-Tactical model to produce the product j in period t, by setting it to 0. Therefore if algorithm 7 determines that the plan can be improved by moving product j, that is creating a hsc setup cost in period sc_t, to another period, the assignment in 4.9 and 4.10 is made, in which min_{jt} is also set to 0 to assure feasibility if previous assignments where made for the same product.

$$maxq_{j,bs_t} = 0 \tag{4.9}$$

$$min_{j,bs_t} = 0 \tag{4.10}$$

The new variable min_{jt} forces the tactical model to produce a minimum quantity of product j defined by algorithm 7. To ensure that this product j is produced in period t, the quantities of that product from periods sc_t and t have to be taken into account. Again to ensure feasibility, the $maxq_{j,bs_t}$ is set to the maximum lot-size M_j. Thus the next formulation is used:

$$min_{j,bs_t} = q_{jt} + q_{jbs_t} \tag{4.11}$$

$$maxq_{j,bs_t} = M_j \tag{4.12}$$

It is important to notice that in this case, unlike method $ILSP_{nR}$, it is possible to force the model to produce in certain periods, since the algorithm 7 already considered the additional costs caused from moving the product to an other macro-period and determined that it improves the solution. This method can obtain a good solution faster but it reduces the solution space of the ILSP-Tactical model, which may lead to a local optimum solution.

Chapter 5

Computational Experiment

In this chapter it is presented the results of the computational experiment, necessary to evaluate and validate the performance of the method proposed. It also serves as study on the various methods chosen to solve the integration of lot-sing and scheduling. This methods are presented in the first section, among them are a GA adapted to this problem and a hierarchical approach method that were developed to serve as a reference to the model proposed. Then in the second section a demonstration and validation of the ILSP method is featured, using an example dataset. Lastly the three main computational studies are detailed, first using instances found in the literature that were decided as references to this research and then using data from a real animal feed company, who was approached in the making of this work.

All the solution method's algorithms were implemented using the programming language **Python 3.6**. To solve the mixed integer programming models we used IBM ILOG CPLEX 12.6 Python API, that solves mathematical models using a algorithm based on branch cut. Every computational experiment were performed on a Intel i7 @ 2.70 GHz processing unit with 16 GB of RAM.

5.1 Alternative methods

In this section we detail the methods that will be used in the computational experiment in comparison with the iterative method. First a GA developed for the lot-sizing and scheduling integration problem is presented. Then a hierarchical strategy algorithm is explained.

5.1.1 Genetic algorithm

To represent the meta-heuristics solution methods, a GA was developed. This method was chosen based on the approaches found in the literature, in which several authors successfully applied genetic algorithms for the integrated lot-sizing and scheduling problem.

5.1.1.1 Solution representation

To represent the individuals of the GA population, a similar representation made in the ILSP was adopted. Each gene represents the amount of production decided for a certain product so it has two integer variables:

- Product family: Type of product to be produced
- Quantity: Number of lots to be produced

Therefore for a production with N products and T macro-periods, an individual consists of a list of T sequences representing the production scheduling for each period t. In Table 5.1 it is presented an example of an individual solution:

Table 5.1: GA individual.

	Fam	Quant	Fam	Quant	Fam	Quant	Fam	Quant	Fam	Quant
Period 1	1	4	3	3	4	5				
Period 2	4	5	3	1	5	2	1	4	2	6
Period 3	2	3	4	1	1	5	5	6		

To determine the *fitness* of each individual, the *decoder* designed uses a similar equation to the objective function (3.1). According to the production quantities of each product, and the sequence decided for each period, the respective costs of backlog, holding inventory, setup changes, overtime and raw material, are calculated.

5.1.1.2 First generation

To initialize the genetic algorithm is necessary to generate a first generation of individuals. In this problem using completely random generated solutions leads to entirely infeasible solutions that would cause the GA to take a long time to evolve to near optimal solutions. To avoid this and have a good starting point, the ILSP-Tactical model, previously mentioned in Chapter 4, is used to provide good estimates for the quantities that each product requires.

Afterwards, to create each individual, those production quantities are randomly sequenced in each period.

5.1.1.3 Selection

The first step of each new generation is to select the best individuals according to their fitness determined by the *decoder* mentioned before.

The GA developed uses an *elitist* strategy, therefore the population with size P is divided into two groups of individuals: a smaller group of p_e elite individuals, the ones with a best fitness, and the remaining P - p_e non-elite individuals. In this strategy all of the elite individuals are copied unchanged to the next generation, this way the best solution of each generation never gets worst than the one from the previous generation, creating a monotonically improving heuristic.

5.1.1.4 Mutation

Mutation is an essential part of GA's evolution, used to enable the algorithm to escape from local optimums. Mutation is implemented by introducing mutant solutions in the next generation. Each one of the previous generation solutions can produce a mutant solution with a predefined probability. Two types of mutation were designed that can occur with the same probability.

The first consists on swapping two random lots from the whole production sequence as represented in the next figure:

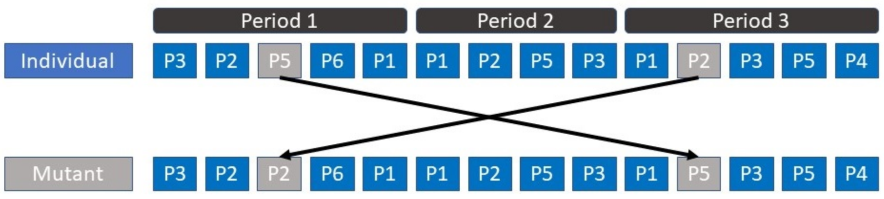

Period 1
Period 2
Period 3
Individual
P3 P2 P5 P6 P1 P1 P2 P5 P3 P1 P2 P3 P5 P4
Mutant
P3 P2 P2 P6 P1 P1 P2 P5 P3 P1 P5 P3 P5 P4

Figure 5.1: GA mutation 1 process example.

The second is more simple where one random gene gets its quantity increased or decreased by 1. Represented in the Table 5.2.

Table 5.2: GA mutation 2 process example.

	Fam	Quant
Individual	1	4
Mutant	1	5

5.1.1.5 Crossover

With only the p_e elite and p_m mutant individuals in the next generation, it is necessary to produce additional new individuals to complete the population size. This is accomplished by mating the solutions from the previous generation.

To select the parents for mating it is used a technique from the BRKGA presented in Chapter 2. Each new individual is generated combining one randomly selected element from the elite portion and one from the non-elite portion. Repetition in the selection process is allowed and therefore an individual can serve as a parent multiple times. To select the parents, both from the elitist and non-elitist, each individual x is assigned with a probability of being selected, using its fitness and the sum of all the individuals fitness ($fsum$) according to the equation: $\frac{x.fitness}{fsum}$. Hence the best solutions are more likely to be chosen and pass their characteristics to future generations.

Having both parents selected, the crossover is done by choosing one cut point from the whole production plan of both parents as represented in Figure 5.2:

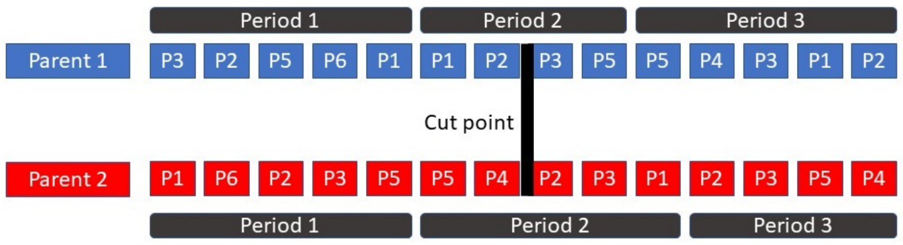

Period 1
Period 2
Period 3
Parent 1
P3 P2 P5 P6 P1 P1 P2 P3 P5 P5 P4 P3 P1 P2
Cut point
Parent 2
P1 P6 P2 P3 P5 P5 P4 P2 P3 P1 P2 P3 P5 P4
Period 1
Period 2
Period 3

Figure 5.2: Cut point for crossover phase of the GA.

Using this cut point as a reference, two offspring solutions are created, by combining the two sequences into one. For the first offspring, the beginning of the sequence is the same as the first parent, and the rest of the products to be produced are placed in the same order in which they are in the sequence of the second parent. For the second offspring the inverse happens. This crossover process is represented in Figure 5.3, assuming the parents in Figure 5.2.

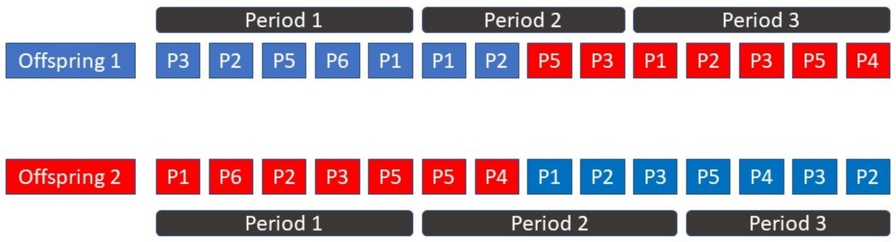

Period 1
Period 2
Period 3
Offspring 1
P3 P2 P5 P6 P1 P1 P2 P5 P3 P1 P2 P3 P5 P4
Offspring 2
P1 P6 P2 P3 P5 P5 P4 P1 P2 P3 P5 P4 P3 P2
Period 1
Period 2
Period 3

Figure 5.3: New solutions generated by the crossover.

5.1.2 Hierarchical strategy

Hierarchical production planning is one of the most common strategy used by companies to link tactical production planning with operational scheduling. In this method tactical production planning is defined first and then production schedule is determined for each macro-period. This method, presented in Figure 5.4, is the third type of solution strategy for integrating product planning and scheduling presented in Figure 1.4.

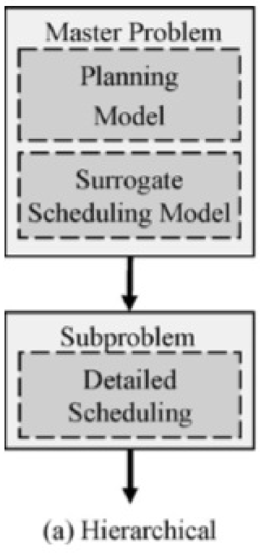


Figure 5.4: Hierarchical strategy framework.

To implement this strategy, the ILSP algorithm is applied but only with one iteration is considered. In order to estimate setup times spent in the tactical level, a reduced production capacity is considered in the tactical level. Therefore, the new capacity is calculated as follows:

$$newcap_t = limiter \times cap_t \qquad \forall t$$

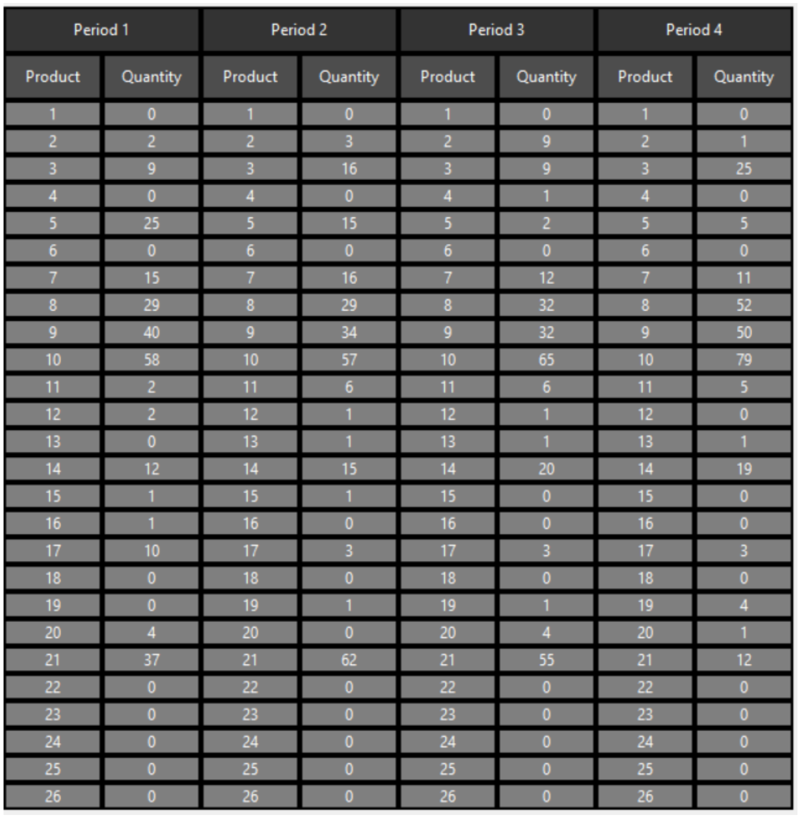
This *limiter* parameter is calculated for each specific production case.

5.2 Illustrative example of ILSP

To demonstrate how the ILSP method works, the evolution of a solution throughout the algorithm iterations is presented. For this example, we consider the instance "month A" presented in [26] and later in the appendix A. It consists of a production planning of 4 periods, of 26 different types of products. The setup times are structured as a sparse matrix with two possible values (X or 0), meaning that setup times and costs are constant so a pair of products either requires a setup or not.

5.2.1 First iteration

As explained in Chapter 4, in the first iteration a simple big-bucket model, ILSP-Tactical, is used to estimate the productions quantities necessary for each product in every period, without considering sequence setups. Figure 5.5 shows the production quantities for the tactical level after the first iteration.

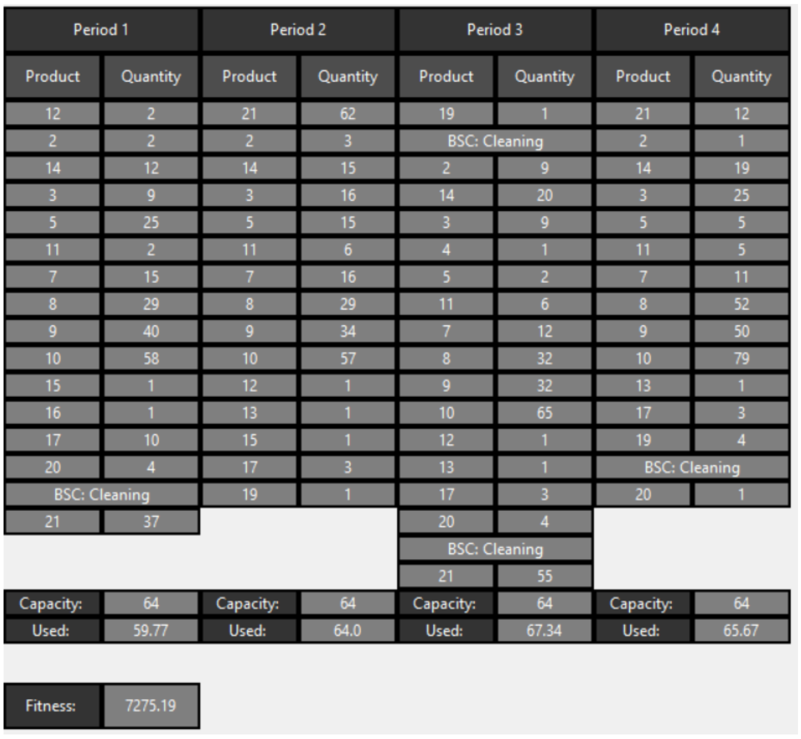


Period 1		Period 2		Period 3		Period 4	
Product	Quantity	Product	Quantity	Product	Quantity	Product	Quantity
1	0	1	0	1	0	1	0
2	2	2	3	2	9	2	1
3	9	3	16	3	9	3	25
4	0	4	0	4	1	4	0
5	25	5	15	5	2	5	5
6	0	6	0	6	0	6	0
7	15	7	16	7	12	7	11
8	29	8	29	8	32	8	52
9	40	9	34	9	32	9	50
10	58	10	57	10	65	10	79
11	2	11	6	11	6	11	5
12	2	12	1	12	1	12	0
13	0	13	1	13	1	13	1
14	12	14	15	14	20	14	19
15	1	15	1	15	0	15	0
16	1	16	0	16	0	16	0
17	10	17	3	17	3	17	3
18	0	18	0	18	0	18	0
19	0	19	1	19	1	19	4
20	4	20	0	20	4	20	1
21	37	21	62	21	55	21	12
22	0	22	0	22	0	22	0
23	0	23	0	23	0	23	0
24	0	24	0	24	0	24	0
25	0	25	0	25	0	25	0
26	0	26	0	26	0	26	0

Figure 5.5: ILSP-Tactical production planning after first iteration.

The production quantities obtained by solving the tactical level are used in the operational level to obtain the best production sequence. Figure 5.6 presents the solution structure for the operational level considering cleaning setups, production capacity and its fitness.

The next table represent its output for this first iteration. It is also possible to see where cleanings are necessary, the capacity limit and the one used in each period. Lastly it is also presented the fitness of the solution, the total costs produced by the represented production plan.



Period 1		Period 2		Period 3		Period 4	
Product	Quantity	Product	Quantity	Product	Quantity	Product	Quantity
12	2	21	62	19	1	21	12
2	2	2	3	BSC: Cleaning		2	1
14	12	14	15	2	9	14	19
3	9	3	16	14	20	3	25
5	25	5	15	3	9	5	5
11	2	11	6	4	1	11	5
7	15	7	16	5	2	7	11
8	29	8	29	11	6	8	52
9	40	9	34	7	12	9	50
10	58	10	57	8	32	10	79
15	1	12	1	9	32	13	1
16	1	13	1	10	65	17	3
17	10	15	1	12	1	19	4
20	4	17	3	13	1	BSC: Cleaning	
BSC: Cleaning		19	1	17	3	20	1
21	37			20	4		
				BSC: Cleaning			
				21	55		
Capacity:	64	Capacity:	64	Capacity:	64	Capacity:	64
Used:	59.77	Used:	64.0	Used:	67.34	Used:	65.67

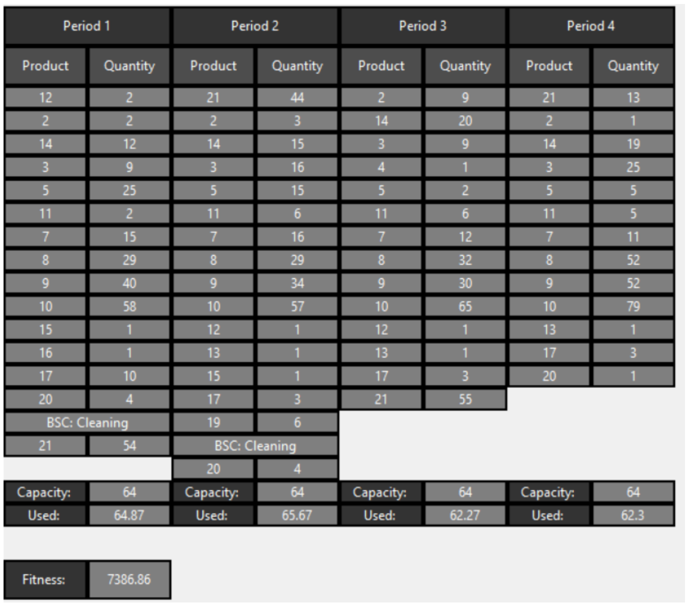
Fitness:	7275.19

Figure 5.6: Solution structure after the first iteration.

5.2.2 Feedback results

At the end of each iteration, feedback from the final solution of operational level is used in the next iteration. As described in Chapter 4, this feedback has two type of information. The first is related to capacity, so in the example shown in Figure 5.6, the feedback will have information that both periods 3 and 4 after the scheduling optimization exceeded the capacity limit and required overtime, and, on the other hand, period 1 had a lot of free time. The second feedback are on the products that caused high setup costs in the previous iteration. The first method proposed, $ILSP_{nR}$, gives freedom to the tactical level to decide where to move those products and $ILSP_R$ forces these problematic products to be produced in the best period found in the operational level.

The results of each method are shown next, for the sequence in Figure 5.6. The solution of $ILSP_{nR}$ in Figure 5.7 shows that the capacity used in the periods have changed from the previous iteration, and also that product 19 and 20 were moved from periods 3 and 4, where they were causing cleanings, to period 2 since the production in these periods now had an associated setup cost.

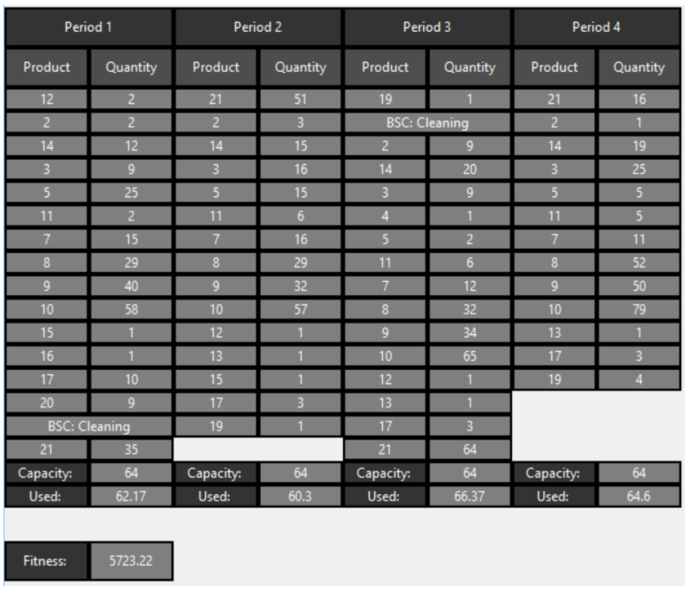


Period 1		Period 2		Period 3		Period 4	
Product	Quantity	Product	Quantity	Product	Quantity	Product	Quantity
12	2	21	44	2	9	21	13
2	2	2	3	14	20	2	1
14	12	14	15	3	9	14	19
3	9	3	16	4	1	3	25
5	25	5	15	5	2	5	5
11	2	11	6	11	6	11	5
7	15	7	16	7	12	7	11
8	29	8	29	8	32	8	52
9	40	9	34	9	30	9	52
10	58	10	57	10	65	10	79
15	1	12	1	12	1	13	1
16	1	13	1	13	1	17	3
17	10	15	1	17	3	20	1
20	4	17	3	21	55		
BSC: Cleaning		19	6				
21	54	BSC: Cleaning					
		20	4				
Capacity:	64	Capacity:	64	Capacity:	64	Capacity:	64
Used:	64.87	Used:	65.67	Used:	62.27	Used:	62.3

Fitness:	7386.86

Figure 5.7: Solution structure after the second iteration of the $ILSP_{nR}$.

For the $ILSP_R$, a new strategy is employed to address problematic products in a sequence. This strategy applies local search to find the best period to place a problematic product. For instance, Figure 5.8 shows that product 20 was creating setup cleanings in both periods 3 and 4 and the search algorithm concluded that the plan could be improved if that product is placed in period 1.



Period 1		Period 2		Period 3		Period 4	
Product	Quantity	Product	Quantity	Product	Quantity	Product	Quantity
12	2	21	51	19	1	21	16
2	2	2	3	BSC: Cleaning		2	1
14	12	14	15	2	9	14	19
3	9	3	16	14	20	3	25
5	25	5	15	3	9	5	5
11	2	11	6	4	1	11	5
7	15	7	16	5	2	7	11
8	29	8	29	11	6	8	52
9	40	9	32	7	12	9	50
10	58	10	57	8	32	10	79
15	1	12	1	9	34	13	1
16	1	13	1	10	65	17	3
17	10	15	1	12	1	19	4
20	9	17	3	13	1		
BSC: Cleaning		19	1	17	3		
21	35			21	64		
Capacity:	64	Capacity:	64	Capacity:	64	Capacity:	64
Used:	62.17	Used:	60.3	Used:	66.37	Used:	64.6

Fitness:	5723.22

Figure 5.8: Solution structure after the second iteration of the $ILSP_R$.

5.2.3 Solution convergence

Considering that the tactical level does not account for setup costs and times, its first solution can be considered as a lower bound for the integrated problem. With the feedbacks from the operational phase, new information related to production scheduling (capacity and setup costs) are added to the tactical level at the end of each iteration. The new information aim at evaluating the real costs of the "relaxed" tactical production planning and it is expected that at one point both tactical and operations solutions converge.

The solution convergence of both $ILSP_{nR}$ and $ILSP_{R}$ for the instance "month A" is depicted in Figures 5.9 and 5.10.

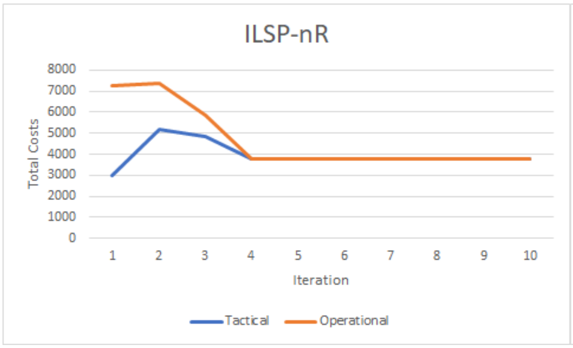


Figure 5.9: $ILSP_{nR}$ convergence.

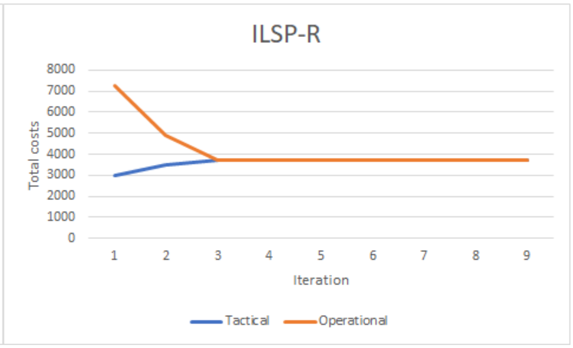
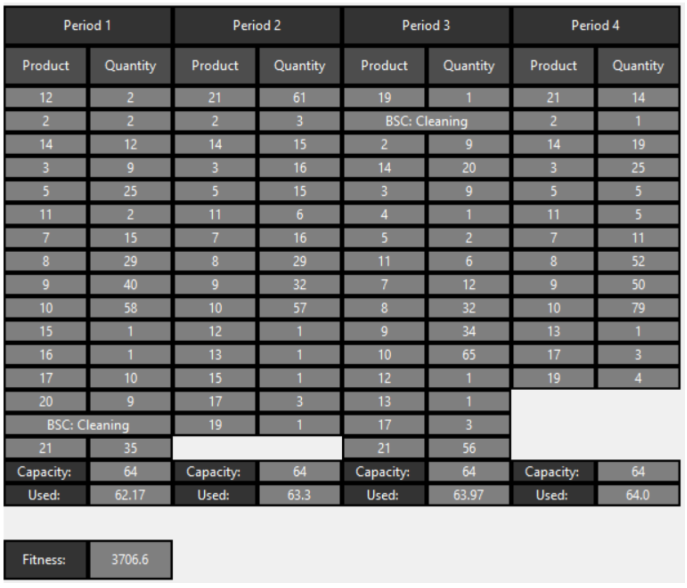


Figure 5.10: $ILSP_{R}$ convergence.

Both models reach similar solutions, but the $ILSP_{nR}$ converges slower than the $ILSP_{R}$ since it evaluates more possible solutions to avoid setup costs.

5.2.4 Final solution

For the instance presented, the best solution found by the *ILSP*$_R$ method is depicted in Figure 5.11



Period 1		Period 2		Period 3		Period 4	
Product	Quantity	Product	Quantity	Product	Quantity	Product	Quantity
12	2	21	61	19	1	21	14
2	2	2	3	BSC: Cleaning		2	1
14	12	14	15	2	9	14	19
3	9	3	16	14	20	3	25
5	25	5	15	3	9	5	5
11	2	11	6	4	1	11	5
7	15	7	16	5	2	7	11
8	29	8	29	11	6	8	52
9	40	9	32	7	12	9	50
10	58	10	57	8	32	10	79
15	1	12	1	9	34	13	1
16	1	13	1	10	65	17	3
17	10	15	1	12	1	19	4
20	9	17	3	13	1		
BSC: Cleaning		19	1	17	3		
21	35			21	56		
Capacity:	64	Capacity:	64	Capacity:	64	Capacity:	64
Used:	62.17	Used:	63.3	Used:	63.97	Used:	64.0

Fitness:	3706.6

Figure 5.11: Final solution obtained by the *ILSP*$_R$ for instance Month A.

5.3 Classical instances

For the first computational experiment, the instances presented in [11] were used.

For all the instances, the parameters values are generated randomly using an uniform distribution. The parameters presented next are created in the same way for each instance, using this limits:

- Demand: U[40-59] units
- Holding costs: U[2-9] units
- Setup times: U[5-10] units
- Processing time: 1 time unit

The setup costs are made proportional to setup times by using a cost factor θ. To define the machine capacity, two parameters *Cut* and *CutVar* were created. *Cut* establishes the target utilization over the entire planning horizon and *CutVar* the maximum deviation from the target capacity utilization in each period, this last one was defined as 0.5. It is ensured that the cumulative capacity utilization in any period does not exceed *Cut* to unsure problem feasibility.

The instances are generated considering the following values for each parameter:

- $N = 15$

- $T \in \{5, 10, 15\}$
- $Cut \in \{0.6, 0.8\}$
- $\theta \in \{50, 100\}$

Combining this parameters, 12 different problem types were formulated with the following designation: $N - T - Cut - \theta$.

The solutions methods compared in this computational experiment are:

- **Full-space models:** GLSP
- **Hierarchical method:** HIER
- **Metaheuristic:** GA
- **Iterative Methods:** $ILSP_{nR}$ and $ILSP_R$

Table 5.3 presents the results of the computational experiment for each one of the methods, the production costs for the best solution found and the respective running time(s). The GLSP model used is a variant of the one presented in Section 3.2, where neither backlog or overtime is considered. For the capacity limit in the HIER method, the capacity reduction is calculated considering the average setup time AVG and the number of products N, so the new capacity is calculated as follows: $cap_t - AVG - stime \times N$. For all the methods, the running time was limited to 1 hour.

Table 5.3: Results for the classic instances

Instances	ILSP-nR		ILSP-R		HIER		GA		GLSP			Literature
	Cost	Time	Cost	Time	Cost	Time	Cost	Time	Cost	Time	Gap	solution
15-5-0.6-50	15874	8	15687	16	15874	8	15399	83	**14.605**	3600	21%	17839
15-5-0.6-100	28192	77	28052	34	31872	13	27511	80	**24586**	3600	29%	29517
15-5-0.8-50	16024	28	15347	30	15926	11	17012	82	**15.318**	3600	19%	17814
15-5-0.8-100	27477	34	**26734**	54	29521	18	28823	80	24.210	3600	31%	30635
15-10-0.6-50	**30534**	199	31758	203	32625	53	33844	167	32.545	3600	41%	35270
15-10-0.6-100	60036	156	60739	156	63479	34	61993	148	**52229**	3600	52%	58268
15-10-0.8-50	30103	164	**29968**	244	30808	51	33510	143	32.002	3600	52%	35475
15-10-0.8-100	60856	97	54121	237	62657	21	61224	143	**54.107**	3600	63%	59197
15-15-0.6-50	48245	433	**47420**	670	48419	57	55427	202	53525	3600	60%	53061
15-15-0.6-100	83954	1096	86579	1604	93405	72	96774	210	**76564**	3600	58%	87509
15-15-0.8-50	46358	175	**45299**	559	46331	66	54881	217	47.946	3600	48%	53393
15-15-0.8-100	83644	703	79904	2275	91871	64	100486,1	212	**79.625**	3600	60%	88468

For the smaller instances, with less periods, the GLSP model reaches better solutions but requires higher computational times. However, once the instances get more extensive, with more products and periods, the solutions of the GLSP models get worst for the limited time, as demonstrated by the gap evolution throughout the instances presented in Figure 5.12.

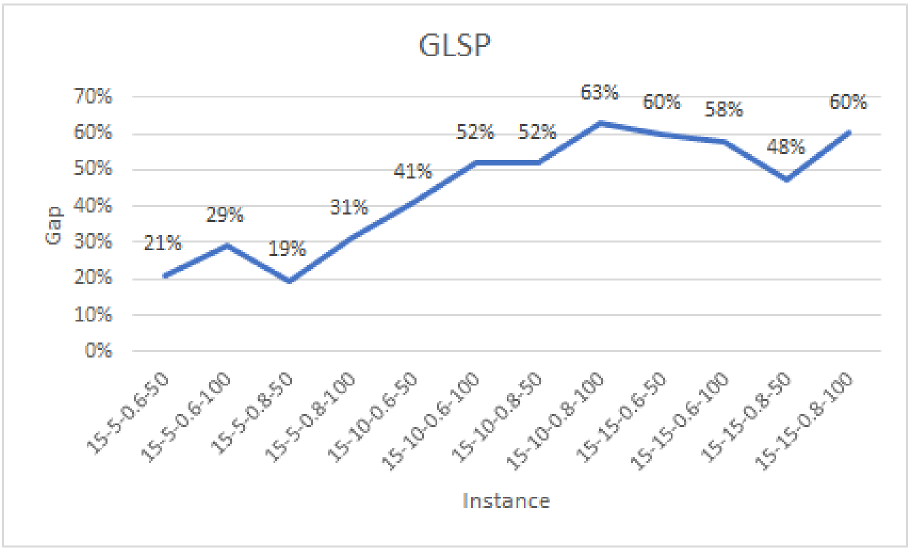


Figure 5.12: Gap evolution for the GLSP model.

Comparing the solutions of $ILSP_R$ and $ILSP_{nR}$, the predicted results are confirmed, with the first reaching better solutions but taking more computational time. This comparison is represented in Figure 5.13.

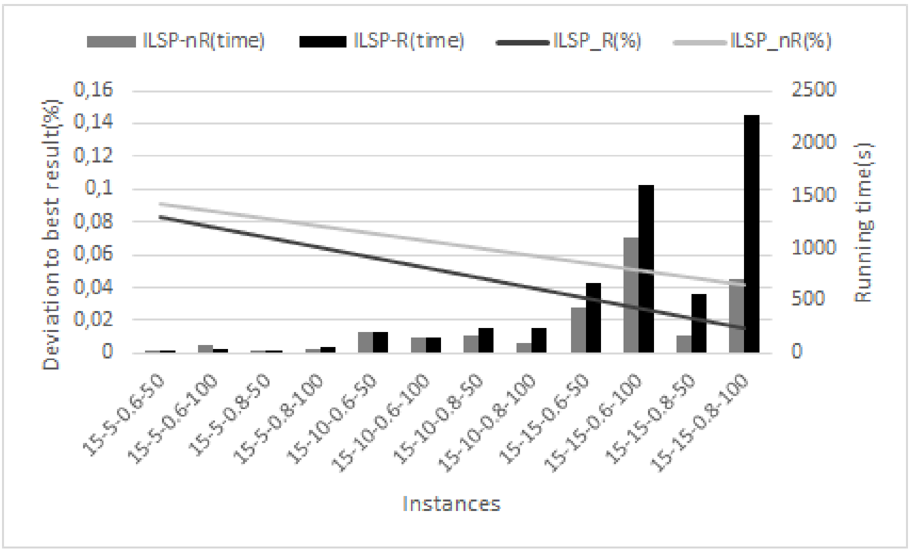


Figure 5.13: Performance comparison between $ILSP_R$ and $ILSP_nR$.

Figure 5.14 depicts the performance evolution across all instances for the $ILSP_R$, GLSP, and the GA, using the deviation to the best solution found in all the methods for each one. The respective trend line is then shown in Figure 5.15. The results show that the iterative method solutions get better for the instances with more products and periods.

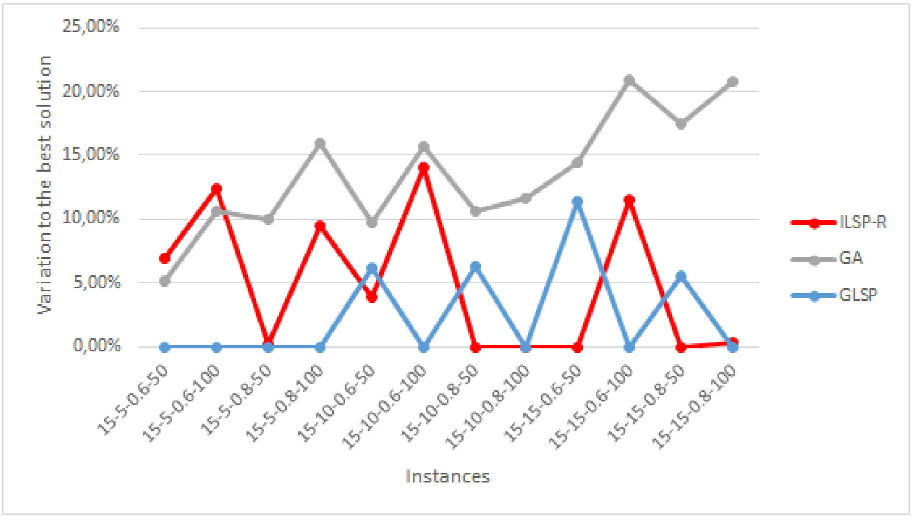


Figure 5.14: Performance comparison for the GLSP, $ILSP_R$and GA.

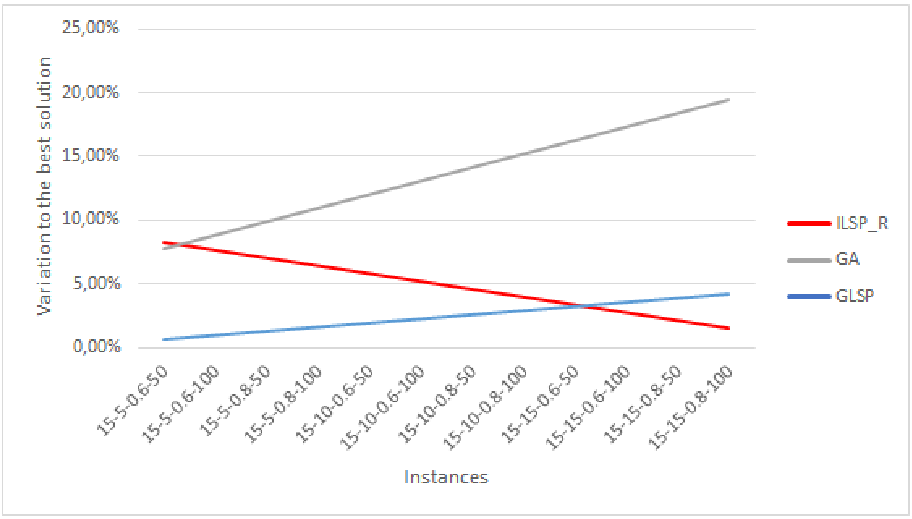


Figure 5.15: Trendline comparison for comparison for the GLSP, $ILSP_R$ and GA.

Figure 5.16 presents the average deviation to the best solution found for all the methods tested, and the average running time.

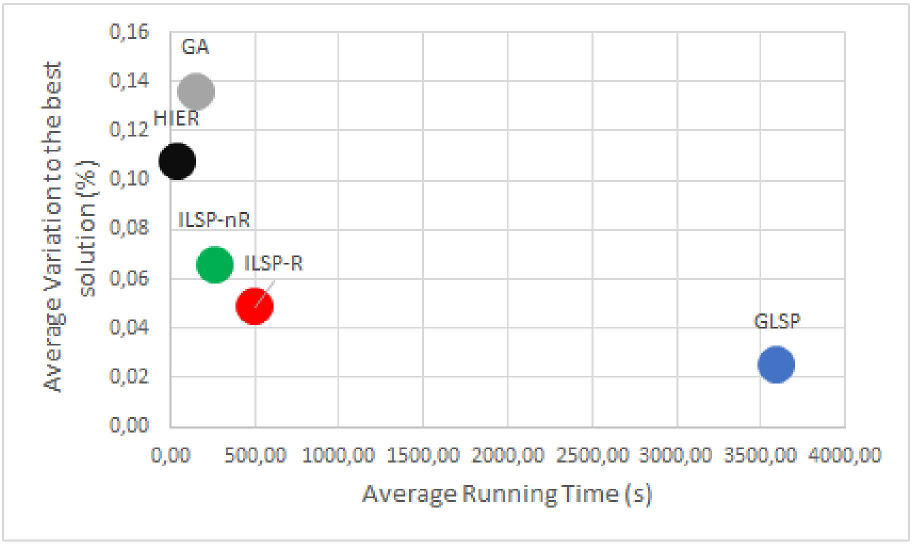


Figure 5.16: Average results for all methods in the variable setup costs case

5.4 Animal feed instances

For the second experimental test, instances from [26] and from a real company case were used. The instances represent problems of an animal feed company. In these problems, setup times are calculated based on cleaning necessity, which consists of two values: cleaning required (setup time = 1.67) and no cleaning required (setup time = 0).

5.4.1 Literature instances

The instances considered in this section, consist of 26 product types, and a planning horizon of 4 weeks. with a less filled demand, where some products do not have requests in various macro-periods.

The same methods were used in this experiment, apart from the full space MIP model GLSP. Instead, the results from the GLSP model proposed by [26] are presented. For this case, the capacity reduction for the HIER method is calculated as follows: the capacity time limit is decreased by 2 cleanings, $capt - 2 \times stime$, in the Tactical Level.

Table 5.4: Results of each method for the animal feed instances.

Instances	ILSP-nR		ILSP-R		HIER		GA		GLSP	
	Result	Time	Result	Time	Result	Time	Result	Time	Result	Time
Month A	3521	10	3707	7	6276	**2**	5255	89	**3519**	360
Month B	17288	6	16870	17	18669	**1**	16875	79	**16616**	157

Figure 5.17 presents the average results and running time of each solution method for the animal feed instances.

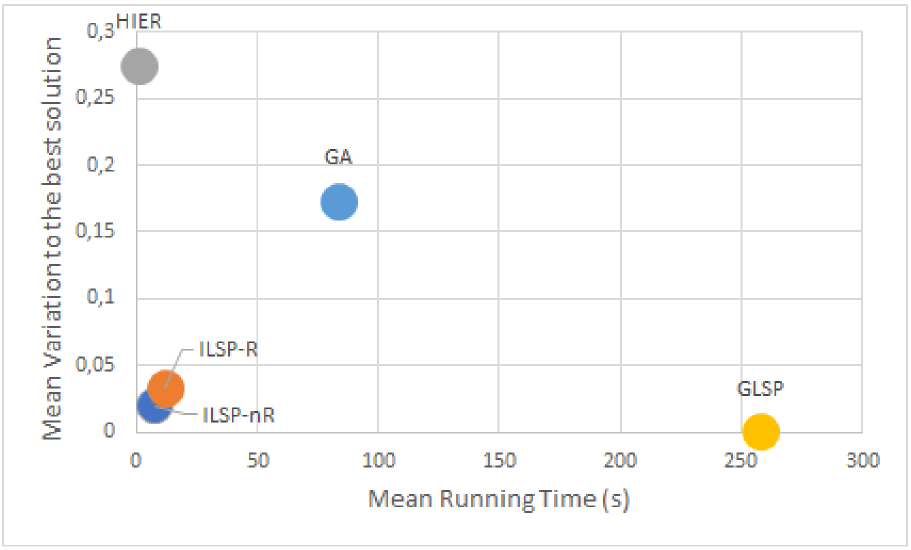


Figure 5.17: Average results for all methods considering the animal feed instances.

The GLSP model is able to converge and find better solutions than the iterative method but it is exponentially slower. For these instances, both $ILSP_R$ and $ILSP_{nR}$ produced similar solutions, with an significant improvement to the HIER and GA methods.

5.4.2 Real animal feed company instance

A real animal feed company was contacted in order to understand how this production planning problems are solved in the industry. The company is based in Brazil, and has a extensive variety of orders, since they produce to both large costumers, that request large quantities of products, and local customers with smaller orders.

The company provided the data related to demand for one week, production and setup time/costs, production capacity, minimum and maximum lot sizes and inventory costs. The demand consists of 51 different product types, each one belonging to a group from a set of 8, each product is also characterized by 9 attributes. Both the groups and the attributes have a matrix of cleaning necessity presented in Tables 5.5 and 5.6, respectively, that are used to combine each pair of products and create a new matrix 51 $\times$ 51 of setup costs and time for each pair of products. Some products can not be produced consecutively even with a cleaning between them so their setup cost is represented by 999999.

Table 5.5: Group cleanings necessity.

Groups	1	2	3	4	5	6	7	8
1	0	0	1	0	0	1	1	0
2	1	0	999999	1	0	999999	999999	1
3	1	1	0	0	0	0	0	0
4	1	1	1	0	0	1	1	0
5	1	1	999999	1	0	999999	999999	1
6	1	1	1	0	0	0	1	0
7	1	1	0	0	0	0	0	0
8	1	1	1	0	0	1	1	0

Table 5.6: Atributtes cleanings necessity.

Atributtes	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1	9999999
2	0	0	0	0	0	0	1	0	0
3	0	0	0	0	0	0	1	0	0
4	1	1	1	1	1	1	1	1	1
5	0	0	0	0	0	0	1	0	0
6	0	0	0	0	0	0	1	0	0
7	0	0	0	0	0	0	1	0	0
8	1	1	1	1	1	1	1	1	1
9	0	0	0	0	0	0	1	0	0

In addition to those data, the company also provided its production planning for that one week period. Through a discussion with the company, the production planning process was concluded to be similar to a hierarchical strategy.

In this experiment only the iterative method and the GA was used to compare the solutions, since for the GLSP model, having 51 different product types, only for the changeover Z_{ijs}, originates $4 \times 51 \times 51 \times 51$ binary variables, that leads to an unreasonable running time. In Figure 5.18 we compare the company's production plan to the one obtained by the $ILSP_R$, presenting the respective productions costs for setups and holding inventory.

Both the $ILSP_R$ and the GA methods present better results than the company's plan for the data considered. The three method solutions are presented in this work's appendix.

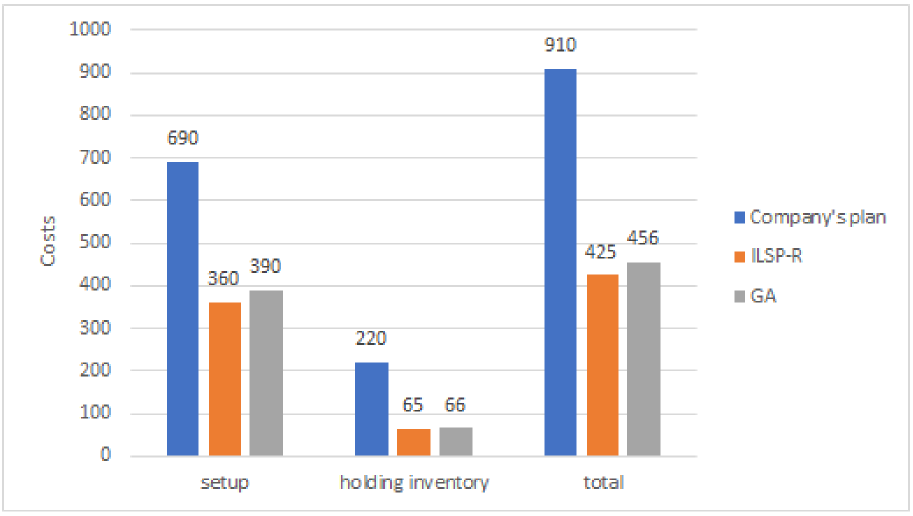


Figure 5.18: Production costs comparison between $ILSP_R$ and the company's plan.

Chapter 6

Conclusion and future work

In this work a novel iterative solution *ILSP* method with two phases was proposed for the lot-sizing and scheduling integration problem. The first is called the tactical level, in which a MIP model is used to calculate the production lot sizes for each macro-period of the planning horizon. Then, in the operational level, local search heuristic is applied to sequence the production lots obtained in the tactical level.

These two phases are executed in consecutive iterations. At the end of each iteration, a feedback to the tactical level is formulated using the best solution found in the operational level. This feedback consists on analyzing the capacity used for each macro-period after the scheduling optimization and balancing the next iteration capacity according to it. An additional feedback about significant setup costs found at the end of the operational level. Two approaches were designed for the formulation of this feedback information. The first, $ILSP_{nR}$, only adds costs, for the next tactical level, to the products in the period where they are causing the high setup costs, so it can decide in which new period to place it. The other approach, $ILSP_{R}$, involves an algorithm that uses a search heuristic to find if the solution can be improved by moving this problematic products. This switches are taken into account in the next iteration tactical level.

We consider that this intuitive phased algorithm would be easy to implement in large production company's with significant sequence-dependent setups costs.

6.1 Conclusions

Several computational experiments were performed to compare the performance of the iterative methods proposed with other solutions methods found in the literature. Among them are a full-space strategy using an MIP model called the GLSP, an genetic algorithm and an hierarchical strategy algorithm (HIER). Firstly a set of classic instances were used, with different production planning dataset ranges. For the small instances, with the less amount of products and macro-periods, the GLSP was able to find better solutions than the other methods, but taking more computational time. For the large instances the solutions found by the GLSP in a feasible time start to get worst than the others methods, since a lot of computational power is needed to solve the MIP model. For these more extensive instances, the iterative methods are able to find better solutions than the GA, but taking more computational time, with the HIER being the faster to solve but getting the worst solutions. Comparing the iterative methods, the $ILSP_R$ is able to find better results than the $ILSP_{nR}$ but with more time required, resulting from the search algorithm.

6.2 Future work

Although it can be concluded that the iterative method provided good solutions for large instances of the lot-sizing and scheduling integration problem, we believe that the algorithm can still be improved.

The local search algorithm used in the operational phase can still be largely improved in order to reduce the computational time in larger problems. Other strategies can also be employed in the operational level to improve the solutions and the computational time. Predefined sequences methods like the one mentioned in Section 2.1.4 may improve the results compared with the local search strategies used. Another possible approach is the use of a second MIP model for the operational level. Therefore, the iterative method method would be composed of two MIP models, where feedback from the second one would change the next iteration tactical level results. This approach may reduce the running time needed in a full-space MIP model.

The simple genetic algorithm developed in this work, presented in Section 5.1.1, showed good results for the integration problem addressed. We believe that the algorithm could however be vastly upgraded, using more complex mutation and crossover phases, that could lead to an improvement in the solutions found. The GLSP method could also be improved, using different models, such as the CC model presented in that could reduce the computational power needed.

The computational experiment carried with the animal feed company only used the data from one week of production. A more extensive comparison, with more insight from the production management department and a larger planning horizon should be conducted to validate the potential gains from the iterative method to the current planning procedures used by the company.

Appendix A

Literature animal-feed instance

Table A.1: Capacity, overtime limit and it's respective costs

Period	Capacity	Overtime limit	Overtime costs
1	64	16	859,2
2	64	16	859,2
3	64	16	859,2
4	64	16	859,2

Table A.2: Processing time and holding inventory cost for each product

Product	Holding cost	Processing time
fam1	0,4	660
fam2	0,4	170
fam3	0,4	85,1
fam4	0,4	151,2
fam5	0,2	103,4
fam6	0,4	110
fam7	0,2	42,1
fam8	0,2	44,3
fam9	0,2	39,2
fam10	0,2	48,8
fam11	0,2	77,5
fam12	0,2	59,1
fam13	0,3	84,9
fam14	0,3	92,2
fam15	0,2	31,2
fam16	0,2	43,2
fam17	0,2	62,1
fam18	0,4	59,2
fam19	0,6	137,1
fam20	0,6	102,6
fam21	0,3	44,6
fam22	0,2	44,3
fam23	0,2	50,1
fam24	0,2	52,5
fam25	0,3	98,6
fam26	0,2	69,2

Table A.3: Demand for each product

Product	Month A				Month B			
	t = 1	t = 2	t = 3	t = 4	t = 1	t = 2	t = 3	t = 4
fam1	0	0	0	0	0	0	0	0
fam2	2	3	9	1	1	6	7	5
fam3	9	16	9	25	12	41	20	16
fam4	0	0	1	0	0	0	0	0
fam5	25	15	2	5	6	6	12	10
fam6	0	0	0	0	0	0	0	0
fam7	15	16	12	11	43	44	52	51
fam8	29	29	32	52	32	32	48	32
fam9	40	32	32	52	32	40	48	32
fam10	58	57	65	79	56	31	57	58
fam11	2	6	6	5	0	0	0	0
fam12	2	1	1	0	4	4	0	0
fam13	0	1	1	1	0	0	0	0
fam14	12	15	20	19	8	8	9	6
fam15	1	1	0	0	0	0	0	0
fam16	1	0	0	0	0	0	0	0
fam17	10	3	3	3	11	11	4	0
fam18	0	0	0	0	0	0	0	0
fam19	0	1	1	4	0	0	0	0
fam20	4	0	4	1	1	1	2	1
fam21	35	38	46	47	56	38	73	36
fam22	0	0	0	0	0	0	0	0
fam23	0	0	0	0	0	0	0	0
fam24	0	0	0	0	0	0	0	0
fam25	0	0	0	0	0	0	0	0
fam26	0	0	0	0	0	0	0	0

Appendix B

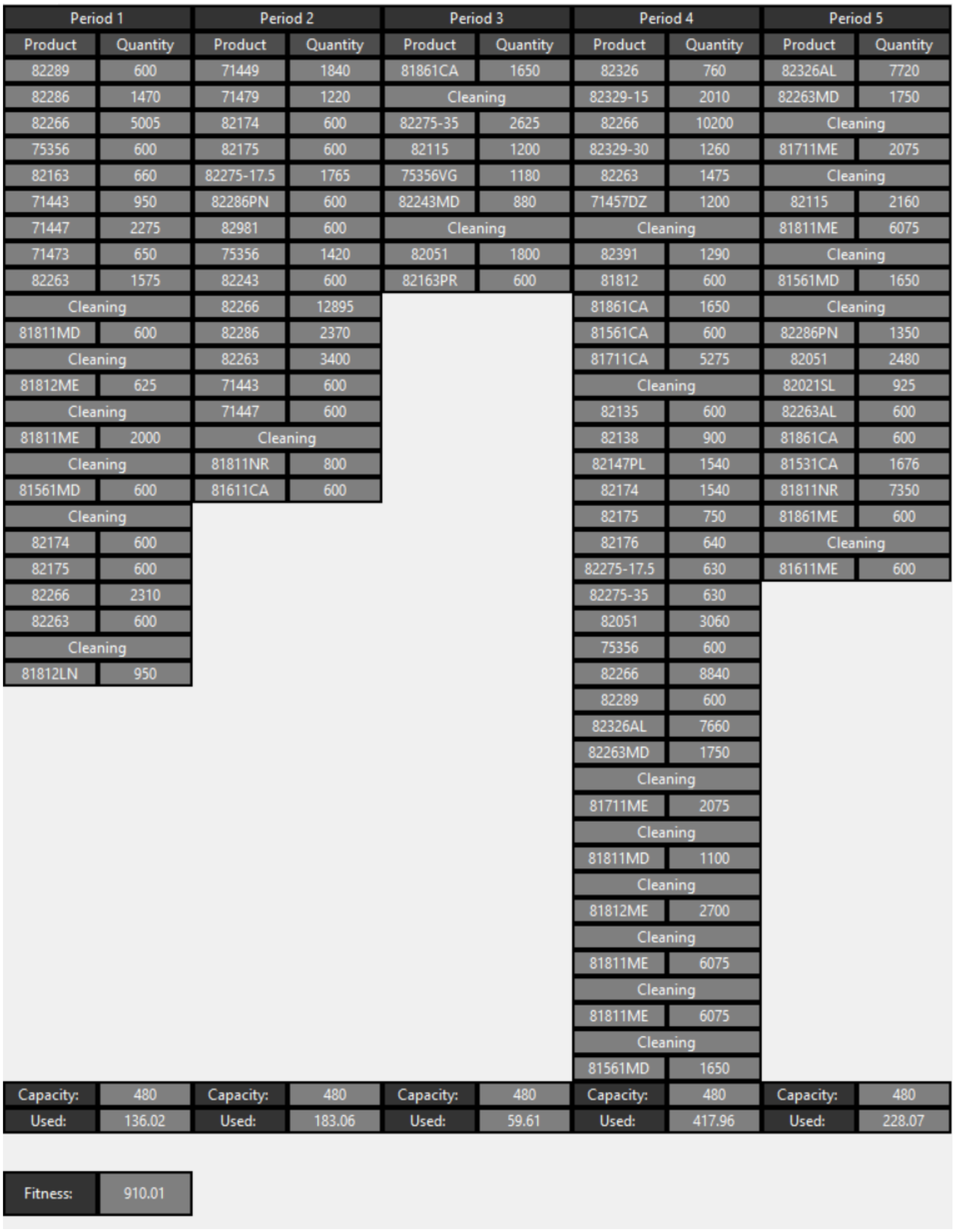
Real animal feed company instance

Table B.1: Groups and attributes for each product

Product	Group	Atributes								
		1	2	3	4	5	6	7	8	9
82289	11	0	0	3	0	0	0	0	0	9
82286	11	0	0	3	0	0	0	0	0	0
82266	11	0	0	3	0	0	0	0	0	9
75356	11	0	0	3	0	0	0	0	0	0
82163	11	0	0	3	0	0	0	0	0	0
71443	12	0	0	3	0	0	0	0	0	0
71447	12	0	0	3	0	0	0	0	0	0
71473	12	0	0	3	0	0	0	0	0	0
82263	12	0	0	3	0	0	0	0	0	0
81811MD	15	1	0	3	0	0	6	0	0	0
81812ME	15	1	0	3	0	0	6	0	0	0
81811ME	15	1	0	3	0	0	6	0	0	0
81561MD	15	1	0	3	0	0	6	0	0	0
82174	10	0	0	0	0	0	0	0	0	0
82175	10	0	0	0	0	0	0	0	0	0
81812LN	15	0	0	3	0	0	0	0	0	9
71449	10	0	0	3	0	0	0	0	0	9
71479	10	0	0	3	0	0	0	0	0	9
82275-35	10	0	0	3	0	0	0	0	0	0
82275-17.5	10	0	0	3	0	0	0	0	0	0
82286PN	10	0	0	3	0	0	0	0	0	0
82981	14	0	0	3	0	0	6	0	0	0
82243	11	0	0	3	0	0	0	0	0	0
81811NR	15	0	0	3	0	0	6	0	0	0
81611CA	15	0	0	3	0	0	0	0	0	0
81861CA	15	0	0	3	0	0	6	0	0	0
82115	10	0	0	3	0	0	0	0	0	0
75356VG	10	0	0	0	0	0	0	0	0	0
82243MD	11	1	0	3	0	0	0	0	0	0
82051	14	0	0	0	0	0	6	0	0	0
82163PR	11	0	0	3	0	0	0	0	0	0
82326	11	0	0	3	0	0	0	0	0	0
82329-15	11	0	0	3	0	0	0	0	0	9
82329-30	11	0	0	3	0	0	0	0	0	9
71457DZ	12	0	0	0	0	0	0	0	0	0
82391	8	0	0	0	0	0	0	0	0	0
81812	15	0	0	3	0	0	6	0	0	0
81561CA	15	0	0	3	0	0	6	0	0	0
81711CA	15	0	0	3	0	0	6	0	0	0
82135	10	0	0	3	0	0	0	0	0	0
82138	10	0	0	3	0	0	0	0	0	0
82147PL	10	0	0	0	0	0	0	0	0	0
82176	10	0	0	0	0	0	0	0	0	0
82326AL	11	0	0	3	0	0	0	0	0	0
82263MD	12	1	0	3	0	0	0	0	0	0
81711ME	15	1	0	3	0	0	6	0	0	0
82021SL	14	0	0	0	0	0	0	0	0	0
82263AL	11	0	0	3	0	0	0	0	0	0
81531CA	15	0	0	3	0	0	6	0	0	0
81861ME	15	1	0	3	0	0	6	0	0	0
81611ME	15	1	0	3	0	0	6	0	0	0

Table B.2: Demand for each product

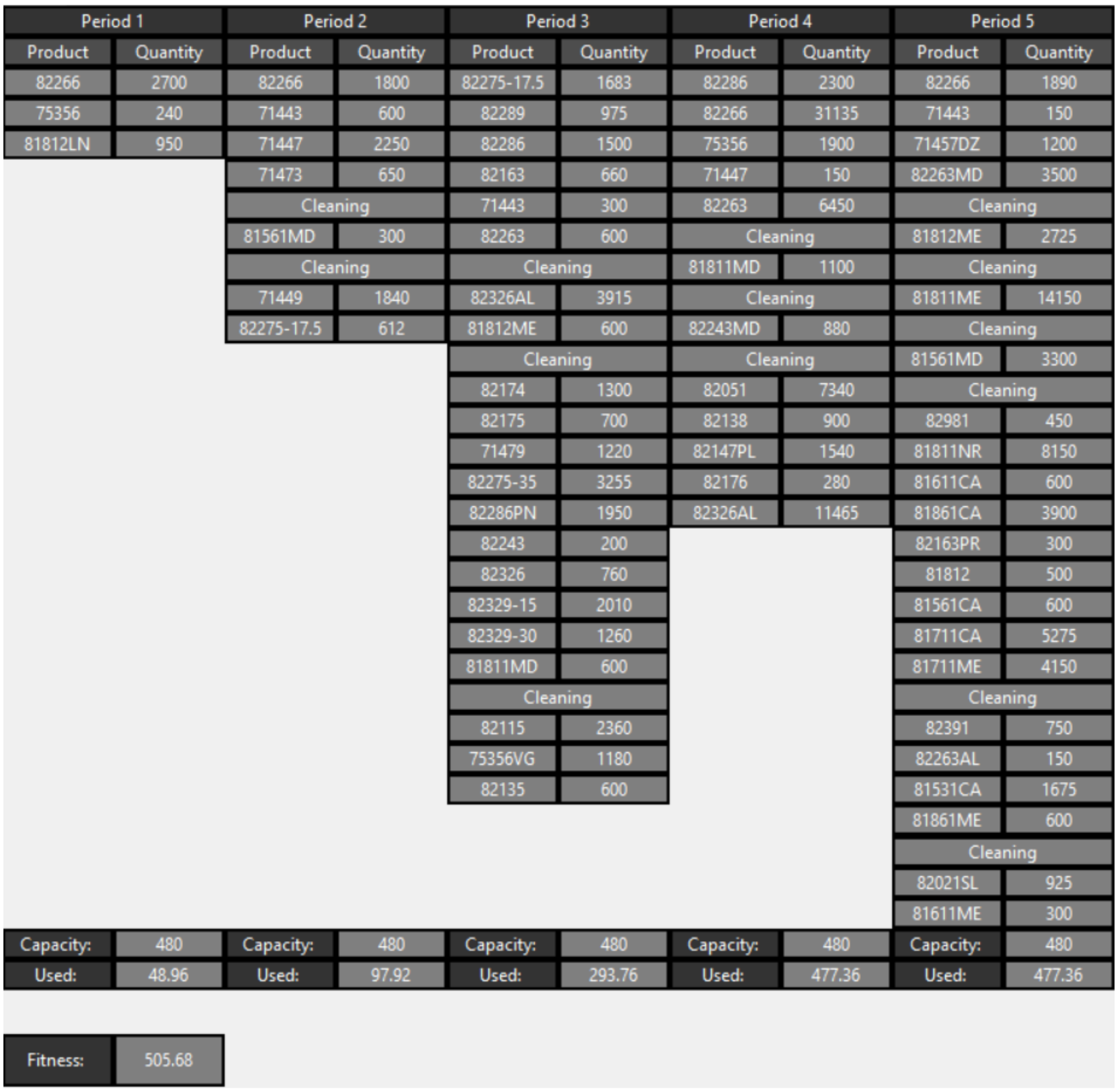
Product	Day 1	Day 2	Day 3	Day 4	Day 5
82289	0	0	500	0	475
82286	0	0	1500	1500	800
82266	2700	1800	0	0	33025
75356	240	0	0	200	1700
82163	0	0	660	0	0
71443	0	600	300	0	150
71447	0	2250	0	150	0
71473	0	650	0	0	0
82263	0	0	600	0	6450
81811MD	0	0	600	0	1100
81812ME	0	0	600	0	2725
81811ME	0	0	0	0	14150
81561MD	0	0	300	0	3300
82174	0	0	0	0	1300
82175	0	0	0	0	700
81812LN	950	0	0	0	0
71449	0	1840	0	0	0
71479	0	0	0	0	1220
82275-35	0	0	0	0	3255
82275-17.5	0	612	875	0	808
82286PN	0	0	0	0	1950
82981	0	0	0	0	450
82243	0	0	0	0	200
81811NR	0	0	0	0	8150
81611CA	0	0	0	0	600
81861CA	0	0	0	0	3900
82115	0	0	0	0	2360
75356VG	0	0	0	0	1180
82243MD	0	0	0	0	880
82051	0	0	0	0	7340
82163PR	0	0	0	0	300
82326	0	0	0	0	760
82329-15	0	0	0	0	2010
82329-30	0	0	0	0	1260
71457DZ	0	0	0	0	1200
82391	0	0	0	0	750
81812	0	0	0	0	500
81561CA	0	0	0	0	600
81711CA	0	0	0	0	5275
82135	0	0	0	0	600
82138	0	0	0	0	900
82147PL	0	0	0	1540	0
82176	0	0	0	0	280
82289	0	0	0	0	15380
82326AL	0	0	0	0	3500
82263MD	0	0	0	0	4150
81711ME	0	0	0	0	925
82021SL	0	0	0	0	150
82263AL	0	0	0	0	1675
81531CA	0	0	0	0	600



Period 1		Period 2		Period 3		Period 4		Period 5	
Product	Quantity	Product	Quantity	Product	Quantity	Product	Quantity	Product	Quantity
82289	600	71449	1840	81861CA	1650	82326	760	82326AL	7720
82286	1470	71479	1220	Cleaning		82329-15	2010	82263MD	1750
82266	5005	82174	600	82275-35	2625	82266	10200	Cleaning	
75356	600	82175	600	82115	1200	82329-30	1260	81711ME	2075
82163	660	82275-17.5	1765	75356VG	1180	82263	1475	Cleaning	
71443	950	82286PN	600	82243MD	880	71457DZ	1200	82115	2160
71447	2275	82981	600	Cleaning		Cleaning		81811ME	6075
71473	650	75356	1420	82051	1800	82391	1290	Cleaning	
82263	1575	82243	600	82163PR	600	81812	600	81561MD	1650
Cleaning		82266	12895			81861CA	1650	Cleaning	
81811MD	600	82286	2370			81561CA	600	82286PN	1350
Cleaning		82263	3400			81711CA	5275	82051	2480
81812ME	625	71443	600			Cleaning		82021SL	925
Cleaning		71447	600			82135	600	82263AL	600
81811ME	2000	Cleaning				82138	900	81861CA	600
Cleaning		81811NR	800			82147PL	1540	81531CA	1676
81561MD	600	81611CA	600			82174	1540	81811NR	7350
Cleaning						82175	750	81861ME	600
82174	600					82176	640	Cleaning	
82175	600					82275-17.5	630	81611ME	600
82266	2310					82275-35	630		
82263	600					82051	3060		
Cleaning						75356	600		
81812LN	950					82266	8840		
						82289	600		
						82326AL	7660		
						82263MD	1750		
						Cleaning			
						81711ME	2075		
						Cleaning			
						81811MD	1100		
						Cleaning			
						81812ME	2700		
						Cleaning			
						81811ME	6075		
						Cleaning			
						81811ME	6075		
						Cleaning			
						81561MD	1650		
Capacity:	480	Capacity:	480	Capacity:	480	Capacity:	480	Capacity:	480
Used:	136.02	Used:	183.06	Used:	59.61	Used:	417.96	Used:	228.07

Fitness:	910.01

Figure B.1: Company’s production plan



Period 1		Period 2		Period 3		Period 4		Period 5	
Product	Quantity	Product	Quantity	Product	Quantity	Product	Quantity	Product	Quantity
82266	2700	82266	1800	82275-17.5	1683	82286	2300	82266	1890
75356	240	71443	600	82289	975	82266	31135	71443	150
81812LN	950	71447	2250	82286	1500	75356	1900	71457DZ	1200
		71473	650	82163	660	71447	150	82263MD	3500
		Cleaning		71443	300	82263	6450	Cleaning	
		81561MD	300	82263	600	Cleaning		81812ME	2725
		Cleaning		Cleaning		81811MD	1100	Cleaning	
		71449	1840	82326AL	3915	Cleaning		81811ME	14150
		82275-17.5	612	81812ME	600	82243MD	880	Cleaning	
				Cleaning		Cleaning		81561MD	3300
				82174	1300	82051	7340	Cleaning	
				82175	700	82138	900	82981	450
				71479	1220	82147PL	1540	81811NR	8150
				82275-35	3255	82176	280	81611CA	600
				82286PN	1950	82326AL	11465	81861CA	3900
				82243	200			82163PR	300
				82326	760			81812	500
				82329-15	2010			81561CA	600
				82329-30	1260			81711CA	5275
				81811MD	600			81711ME	4150
				Cleaning				Cleaning	
				82115	2360			82391	750
				75356VG	1180			82263AL	150
				82135	600			81531CA	1675
								81861ME	600
								Cleaning	
								82021SL	925
								81611ME	300
Capacity:	480	Capacity:	480	Capacity:	480	Capacity:	480	Capacity:	480
Used:	48.96	Used:	97.92	Used:	293.76	Used:	477.36	Used:	477.36

Fitness:	505.68

Figure B.2: $ILSP_{nR}$ solution

References

[1] Christos T Maravelias and Charles Sung. Integration of production planning and scheduling: Overview, challenges and opportunities. *Computers & Chemical Engineering*, 33(12):1919–1930, 2009.

[2] José Fernando Gonçalves and Mauricio GC Resende. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, 17(5):487–525, 2011.

[3] Alistair Clark, Masoumeh Mahdieh, and Socorro Rangel. Production lot sizing and scheduling with non-triangular sequence-dependent setup times. *International Journal of Production Research*, 52(8):2490–2503, 2014.

[4] António Aroso Menezes, Alistair Clark, and Bernardo Almada-Lobo. Capacitated lot-sizing and scheduling with sequence-dependent, period-overlapping and non-triangular setups. *Journal of Scheduling*, 14(2):209–219, 2011.

[5] Dileep R Sule. *Production planning and industrial scheduling: examples, case studies and applications*. CRC press, 2007.

[6] Andreas Drexl and Alf Kimms. Lot sizing and scheduling—survey and extensions. *European Journal of operational research*, 99(2):221–235, 1997.

[7] Karina Copil, Martin Wörbelauer, Herbert Meyr, and Horst Tempelmeier. Simultaneous lotsizing and scheduling problems: a classification and review of models. *OR spectrum*, 39(1):1–64, 2017.

[8] Bernhard Fleischmann. The vehicle routing problem with multiple use of vehicles. *Forschungsbericht Fachbereich Wirtschaftswissenschaften, Universität Hamburg*, 1990.

[9] Uday S Karmarkar and Linus Schrage. The deterministic dynamic product cycling problem. *Operations Research*, 33(2):326–345, 1985.

[10] Andreas Drexl and Knut Haase. Proportional lotsizing and scheduling. *International Journal of Production Economics*, 40(1):73–87, 1995.

[11] Luis Guimarães, Diego Klabjan, and Bernardo Almada-Lobo. Modeling lotsizing and scheduling problems with sequence dependent setups. *European Journal of Operational Research*, 239(3):644–662, 2014.

[12] Bernhard Fleischmann and Herbert Meyr. The general lotsizing and scheduling problem. *Operations-Research-Spektrum*, 19(1):11–21, 1997.

[13] Alistair R Clark and Simon J Clark. Rolling-horizon lot-sizing when set-up times are sequence-dependent. *International Journal of Production Research*, 38(10):2287–2307, 2000.

[14] Knut Haase and Alf Kimms. Lot sizing and scheduling with sequence-dependent setup costs and times and efficient rescheduling opportunities. *International Journal of Production Economics*, 66(2):159–169, 2000.

[15] Egon Balas. The prize collecting traveling salesman problem. *Networks*, 19(6):621–636, 1989.

[16] Claudio Fabiano Motta Toledo, Márcio da Silva Arantes, Marcelo Yukio Bressan Hossomi, Paulo Morelato França, and Kerem Akartunalı. A relax-and-fix with fix-and-optimize heuristic applied to multi-level lot-sizing problems. *Journal of heuristics*, 21(5):687–717, 2015.

[17] Eli AV Toso, Reinaldo Morabito, and Alistair R Clark. Lot sizing and sequencing optimisation at an animal-feed plant. *Computers & Industrial Engineering*, 57(3):813–821, 2009.

[18] Stefan Helber and Florian Sahling. A fix-and-optimize approach for the multi-level capacitated lot sizing problem. *International Journal of Production Economics*, 123(2):247–256, 2010.

[19] Andrea T Staggemeier and Alistair R Clark. A survey of lot-sizing and scheduling models. In *23rd annual symposium of the Brazilian operational research society (SOBRAPO)*, pages 938–947. Citeseer, 2001.

[20] Fred Glover. Tabu search—part i. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[21] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[22] Vijini Mallawaarachchi. Introduction to genetic algorithms - including example code, Jul 2017.

[23] Poonam Garg. A comparison between memetic algorithm and genetic algorithm for the cryptanalysis of simplified data encryption standard algorithm. *arXiv preprint arXiv:1004.0574*, 2010.

[24] Yuping Wang et al. A new hybrid genetic algorithm for job shop scheduling problem. *Computers & Operations Research*, 39(10):2291–2299, 2012.

[25] Ana Sofia Figueirdo, Eduardo Curcio, and Pedro Amorim. Lot-sizing and scheduling optimization in animal feed industry. Technical report, Faculty of Engineering, University of Porto, 2018.

[26] Eli Angela Vitor Toso et al. Dimensionamento e seqüenciamento de lotes de produção na indústria de suplementos para nutrição animal. 2008.