THE FEDERAL CENTER OF TECHNOLOGICAL EDUCATION OF PARANÁ
Graduate School in Electrical Engineering and Industrial Computer Science
CEFET-PR/CPGEI

Mixed Integer Linear Programming and Constraint Logic Programming: Towards a Unified Modeling Framework

Doctoral Thesis

by

Leandro Magatão

Curitiba, Paraná, Brazil
May 12, 2005.

LEANDRO MAGATÃO

Mixed Integer Linear Programming and Constraint Logic Programming: Towards a Unified Modeling Framework

Doctoral Thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Science (DSc) at the Graduate School in Electrical Engineering and Industrial Computer Science (CPGEI), The Federal Center of Technological Education of Paraná (CEFET-PR), emphasis on Industrial Computer Science.

Adviser: Prof.ª Dra. Lúcia Valéria Ramos de Arruda

Curitiba, Paraná, Brazil
May 12, 2005.

Ficha catalográfica elaborada pela Biblioteca do CEFET-PR – Unidade Curitiba

M189m Magatão, Leandro
Mixed integer linear programming and constraint logic programming : towards a unified modeling framework / Leandro Magatão. – Curitiba : [s.n.], 2005.
xvii, 166 p. : il. ; 30 cm

Orientador : Profª Drª Lúcia Valéria Ramos de Arruda
Tese (Doutorado) – CEFET-PR. Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial. Curitiba, 2005.
Bibliografia : p. 155-66

1. Programação (Matemática). 2. Programação linear. 3. Programação inteira. 4. Programação lógica. 5. Otimização combinatória. 6. Agenda de execução (Administração). 7. Restrições (Inteligência artificial). 8. Oleodutos. 9. Simulação (Computadores). I. Arruda, Lúcia Valéria Ramos de, orient. II. Centro Federal de Educação Tecnológica do Paraná. Curso de Pós-Graduação em Engenharia Elétrica e Informática Industrial. III. Título.

CDD : 519.7
CDU : 65.012.122

Contents

List of Tables

List of Figures

List of Labels

AI	Artificial Intelligence
CLP	Constraint Logic Programming
CNF	Conjunctive Normal Form
CP	Constraint Programming
COPs	Combinatorial Optimization Problem(s)
CS	Computer Science
CSPs	Constraint Satisfaction Problem(s)
DNF	Disjunctive Normal Form
FD-store	Finite Domain Store
GDP	Generalized Disjunctive Programming
ILP	Integer Linear Programming
IP	Integer Programming
LCIF	Logical Constraint in the Implication Form
LCEF	Logical Constraint in the Equivalence Form
LFC$_k$	Linear Form Constraint k
LP	Linear Programming
LP-store	Linear Programming Store
LS	Local Search
MILP	Mixed Integer Linear Programming
MINLP	Mixed Integer Non-Linear Programming
MI(N)LP	Mixed Integer (Non-)Linear Programming
MIP	Mixed Integer Programming
MLLP	Mixed Logical Linear Programming
OPL	Optimization Programming Language
OR	Operational Research, Operations Research
SOS	Special Ordered Sets
TSP	Traveling Sales Person

Sponsorship

The author of this thesis acknowledges financial support from ANP and FINEP (PRH-ANP/MCT PRH10 CEFET-PR).

Acknowledgments

In a theoretical point of view, this work owes much to others and, fortunately, it is impossible to name them all. To a large degree it rests upon intellectual foundation laid by pioneers, who have skilled explored the connections between logic and mathematical programming. In a practical point of view, there are many individuals that have made it possible for me to pursue a doctorate, and it is also impossible to name them all. In particular, I would like to thank my adviser, Prof.ª Valéria, for the guidance, the free will on the research, and also for proofreading and commenting the entire manuscript(s) in depth. To Prof. Flávio, for his endless effort to coordinate the laboratory (financial) issues. His effort has been a sine qua non to my (and to others) research conduction. To the colleague Sérgio, for the research collaborations or discussions, advices, and comments, which gave me an invaluable help during all the ups and downs upon the research efforts. I am also specially grateful to the colleague Antônio, for his goodwill in always help me (and anyone else) in handling hardware and software computational issues. In addition, I would like to thank everyone in the "LASCA/CPGEI/CEFET-PR" research group for the many discussions that, in fact, hardly ever contributed to the research, but have made these years very enjoyable. I have never stopped learning from them. Last, but not least, I am thoroughly grateful to my family, for the unconditional support.

Abstract

The struggle to model and solve Combinatorial Optimization Problems (COPs) has challenged the development of new approaches to deal with COPs. In one of the front lines of such approaches, Operational Research (OR) and Constraint Programming (CP) optimization techniques are beginning to converge, despite their very different origins. More specifically, Mixed Integer Linear Programming (MILP) and Constraint Logic Programming (CLP) are at the confluence of the OR and the CP fields. This thesis summarizes and contrasts the essential characteristics of MILP and CLP, and the ways that they can be fruitfully combined. Chapters 1 to 3 sketch the intellectual background for recent efforts at integration and the main results achieved. In addition, these chapters highlight that CLP is known by its rich modeling framework, and the MILP modeling vocabulary is just based on inequalities, which makes the modeling process hard and error-prone. Therefore, a combined CLP-MILP approach suffers from this MILP inherited drawback. In chapter 4, this issue is addressed, and some "high-level" MILP modeling structures based on logical inference paradigms are proposed. These structures help the formulation of MILP models, and can be seen as a contribution towards a unifying modeling framework for a combined CLP-MILP approach. In addition, chapter 5 presents an MILP formulation addressing a combinatorial problem. This problem is focused on issues regarding the oil industry, more specifically, issues involving the scheduling of operational activities in a multi-product pipeline. Chapter 5 demonstrates the applicability of the high-level MILP modeling structures in a real-world scenario. Furthermore, chapter 6 presents a CLP-MILP formulation addressing the same scheduling problem previously exploited. This chapter demonstrates the applicability of the high-level MILP modeling structures in an integrated CLP-MILP modeling framework. The set of simulations conducted indicates that the combined CLP-MILP model was solved to optimality faster than either the MILP model or the CLP model. Thus, the CLP-MILP framework is a promising alternative to deal with the computational burden of this pipeline-scheduling problem. In essence, this thesis considers the integration of CLP and MILP in a modeling standpoint: it conveys the fundamentals of both techniques and the modeling features that help establish a combined CLP-MILP approach. Herein, the concentration is on the building of MILP and CLP-MILP models rather than on the solution process.

Keywords: Mixed Integer Linear Programming (MILP), Constraint Logic Programming (CLP), Combinatorial Optimization Problems (COPs), Scheduling, Pipeline.

Chapter 1

CLP–MILP: Introduction

This chapter briefly introduces constraint logic programming and mixed integer linear programming, discusses the opportunities for the integration of both techniques, and presents the thesis' outline.

1.1 Preliminary Overview

Many (real-life) problems have a combinatorial nature. Often, one has to find a solution to a situation by searching in a finite, but generally huge, collection of alternatives. In many cases, such a solution is chosen according to some well-defined criterion. Therefore, the set of solutions is evaluated and the best one is selected, that is, an optimized answer is chosen instead of just a feasible answer.

Decision-making problems that can be viewed as combinatorial (optimization) problems are ubiquitous in a variety of areas, such as: logistics, finance, transportation, configuration, etc. In a simplified standpoint, a combinatorial (optimization) problem is concerned with the efficient allocation of limited resources in order to meet desired objectives. Furthermore, the values of some (or all) of the involved variables are restricted to be integrals. Just to name a few examples of such problems, in production, the sequencing of jobs and assignment of resources, as well as packaging, design, and cutting stock problems have to be solved. In transportation, companies face decisions concerning the fleet management, timetables (flights, buses), configuration (e.g. trains), and intermediate storing. Personnel-related issues are crew scheduling (vital in the airline industry), the staffing of shifts (e.g. in health care), and timetables for courses. A typical managerial task is the planning of projects. An example in finance is optimal portfolio management. In telecommunications, routing and network layout decisions have to be taken (Heipcke, 1999). Therefore, the few examples previously named illustrate that Combinatorial Optimization Problems (COPs) are present in day-to-day situations, and their study is an important issue, even more because COPs have a negative characteristic: they are difficult to solve. In particular, the classical book of Garey and Johnson (1979) contains an explanation about the complexity of such class of problems.

The importance of solving combinatorial optimization problems has challenged the development of techniques to deal with these problems, and the Operations Research (OR) approach for solving COPs has been used for a long time. This approach was burst by advancements in continuous optimization, with its foundation in the rise of Linear Programming - LP (Dantzig, 1963). This development took place in the second half of the 20th century, and has defined the field known as Mixed Integer Linear Programming - MILP (Nemhauser and Wolsey, 1988; Wolsey, 1998). In a typical MILP, some variables (or all for Integer Programming - IP) are constrained to be integers, and linear equalities and inequalities link the variables. MILP technologies have been employed with tremendous success for solving combinatorial optimization problems. In recent years, however, a new programming paradigm has evolved within the Computer Science (CS) and Artificial Intelligence (AI) communities: Constraint Logic Programming (CLP).

According to Barták (2002), CLP can be understood as a framework for solving combinatorial (optimization) problems. The basic idea is to model the problem as a set of variables with domains (the valid values for each variable), and a set of constraints restricting the possible combinations for the values of variables. In a CLP framework, variables are linked by a set of constraints that can be mathematical or symbolical. Constraints embed constraint propagation algorithms that effectively reduce the search space by removing combinations of variable-value assignments that are proven infeasible[1]. The constraint logic programming methodology allows the natural expression of complex relationships between variables, including logical expressions. It is ideally suited for operational problems, which require fast and feasible answers. Users can also guide the search process based on their own problem knowledge. The industrial success of CLP has, little by little, caught the attention of many OR researchers for this technology (Focacci, 2000). CLP has been used for modeling and solving combinatorial optimization problems such as scheduling, planning, sequencing, and routing. A complete survey of CLP applications is given by Rossi (1999).

Currently, an increasing number of researchers, both from the OR and the AI communities, are investigating the possibility of integrating methodologies from the two fields for solving COPs. As stated by Hooker (2002), OR optimization techniques and CLP are beginning to converge, despite their very different origins. OR optimization is primarily associated with mathematics and engineering, while CLP developed much more recently in the CS and AI communities. The two fields evolved independently until a few years ago. Yet, they have much in common, and they are applied to many of the same problems. Both have enjoyed considerable commercial success. Most important for present purposes, they have complementary strengths, and the last few years have seen growing efforts to combine them. The recent interaction between OR optimization and CLP promises to change both fields. It is conceivable that portions of both will merge into a single problem solving technology for discrete and mixed discrete/continuous problems. The key feature in such integration is that OR optimization and CLP are similar enough to make their combination possible and yet different enough to make it profitable (Hooker, 2002).

1.2 Thesis' Outline

Situated at the confluence of the OR and the AI fields, the integration of Constraint Logic Programming (CLP) and Mixed Integer Linear Programming (MILP) is an emerging discipline that has been recognized as a suitable environment for

[1]Further details about constraint propagation are given in section 2.1.

achieving the best that both fields can contribute to solve COPs. This thesis summarizes and contrasts the characteristics of MILP and CLP, and the ways that they can be combined.

The benefits of an integrated CLP-MILP approach are explained in chapters 2 and 3. Chapter 2 conveys the fundamentals of CLP and MILP that guide the opportunities for integrating techniques. Chapter 3 surveys a cluster of articles that have contributed to different facets in this integration process. In particular, these chapters sketch the intellectual background for recent efforts at CLP-MILP integration and are not aimed at being a comprehensive review of neither CLP nor MILP, but rather an introductory discussion of opportunities for the integration of these techniques in a combined CLP-MILP approach.

Chapters 2 and 3 rise a quite large body of evidences that Mixed Integer Linear Programming and Constraint Logic Programming are merging. In addition, the literature surveyed in these chapters indicates that the combined CLP-MILP approach is a promising alternative to deal with combinatorial optimization problems. However, the integration process brings some technical difficulties, which are explained, for instance, by Heipcke (1999). One of these difficulties is that the MILP modeling vocabulary is based on inequalities, which makes the modeling process hard and error-prone. On the other hand, the CLP modeling vocabulary involves a series of different (logical) operators[2], besides inequalities. Thus, CLP is stronger than MILP in its expressive power (Williams and Wilson, 1998). Either in a traditional MILP framework or in a combined CLP-MILP approach, the MILP formulation is a difficult task. A combined CLP-MILP approach suffers from this MILP inherited drawback.

In chapter 4, the MILP modeling issue is addressed in details. The difficulties of an LP/MILP modeling framework are stretched, and some "high-level" MILP modeling structures based on logical inference paradigms are proposed to soften such modeling difficulties. These structures help the formulation of MILP models, and can be seen as a contribution towards a unifying modeling framework for a combined

[2]Further details about CLP operators are given in chapter 2.

approach between CLP and MILP. Therefore, chapter 4 provides an approach to narrow the gap between CLP and MILP modeling devices.

Chapter 5 presents the development of an MILP formulation addressing a combinatorial problem. This model is focused on issues regarding the oil industry, more specifically, issues involving the scheduling of operational activities in a multi-product pipeline. The main goal is to demonstrate the applicability of the high-level MILP modeling structures developed in chapter 4 rather than to fully exploit the problem-scheduling details. The modeling and optimization tool Extended LINGO/PC Release 8.0 (LINDO, 2002) is used to implement and solve the MILP formulation.

Chapter 6 presents a CLP and a combined CLP-MILP formulation addressing the same problem exploited in chapter 5. The main goal is to demonstrate the applicability of the high-level MILP modeling structures developed in chapter 4 in an integrated CLP-MILP modeling framework. The functionalities of the modeling and optimization tool ILOG OPL Studio 3.6.1 (ILOG, 2002b) are used to implement the root MILP[3] and CLP formulations and also the combined CLP-MILP approach. The computational results presented by the MILP, the CLP and the CLP-MILP approaches are compared, demonstrating that the combined approach is a promising alternative to deal with this pipeline-scheduling problem.

Last but not least, chapter 7 presents the main thesis' conclusions/contributions and the directions for future research. The interested reader is invited to examine the following chapters, which focus on modeling and how a rethinking on modeling[4] traditions can aid the formulation of either MILP or CLP-MILP approaches.

[3]The MILP formulation, which is previously presented/implemented in chapter 5, is (re)implemented in the ILOG OPL Studio 3.6.1.

[4]The development of algorithms to improve the communication between CLP and MILP devices is not addressed in this work.

Chapter 2

CLP–MILP: Overview of Main Characteristics

This chapter introduces the fundamental principles of CLP and MILP (search, relaxation, and inference) and overviews the main characteristics of both techniques.

2.1 Introduction to CLP

Constraint Logic Programming (CLP) is a multidisciplinary research area that can be located between AI, OR, and programming languages. CLP has to do with modeling, solving, and programming problems, which can be described as a set of statements (the constraints) that pose some relationship amongst the problem's variables (Rossi, 1999).

In last few years, CLP has attracted high attention among experts from many areas due to its potential for solving real-life problems. Not only it is based on a strong theoretical foundation but it is also attracting widespread commercial interest, in particular, in areas of modeling optimization problems. Not surprisingly, it has been identified by the Association for Computing Machinery (ACM) as one of the strategic directions in computer research (Barták, 1999).

The seminal work of computational systems based on constraints, or Constraint

Programming (CP) systems, can be traced back in the late 70s (Laurière, 1978). However, the success of constraint programming technology was sprung due to the union of CP systems with Logic Programming inference methods (Colmerauer, 1987; Hentenryck, 1989). CLP is a programming paradigm that combines constraint solving techniques with logic programming devices (Jaffar and Maher, 1994).

According to Thorsteinsson (2001), CLP programs are traditionally aimed at solving feasibility problems rather than optimization problems. In feasibility problems the goal is to find a solution to a situation satisfying a set of requirements; in optimization problems, the goal is to find the best solution given some criteria on what makes one solution better than another. Classical examples of feasibility programs are the n-queens problem and the map coloring problem. In the former, n queens should be placed on an n by n chessboard in arrangements that no queen can capture another queen. In the later, the goal is to color different regions on a map with a limited number of colors, such that adjacent regions have different colors. These problems are more accurately classified as Constraint Satisfaction Problems (CSPs). However, an algorithm that finds a feasible solution can be turned into an optimization algorithm by imposing a bound on the objective function. Anytime a better feasible solution is found, the bound is updated, and the solution is stored. When no more feasible solutions are found, the last stored solution is the optimal one.

Informally speaking, an instance of a CSP is described by a set of variables, a set of possible values for each variable, and a set of constraints among variables. The set of possible values of a variable is called the variable's *domain*. A constraint among variables expresses the combinations of values that are allowed for the variables. The question to be answered for a CSP instance is whether there exists an assignment of values to variables, such that all constraints are satisfied. Such an assignment is called a *solution* of the CSP (Smith, 1995). One of the key ideas of CLP is that constraints can be "actively" used to reduce the computational effort needed to solve combinatorial problems. Constraints are, thus, not only used to test the validity of a solution, as in conventional mathematical programming, but also in an active mode: they are used to remove values from the domains, deduce new constraints, and detect inconsistencies. This process of actively using constraints to come to certain deductions is called *constraint propagation*, which decisively aids to prune the search space . The specific deductions that result in the removal of values from the domains are called *domain reductions*. The set of values in the domain of a variable that are not invalidated by constraint propagation is called the *current domain* (*indomain*) of that variable (Baptiste et al., 2001). Modeling languages (constraint-based languages) and efficient constraint propagation algorithms have been developed to address classes of constraints, such as boolean constraints, finite domain constraints, interval constraints, and linear constraints (Rossi, 1999).

Therefore, the core of CLP is based on *constraint propagation* and *domain reduction* techniques. Considering a simple example, where x and y denote integer variables, $x < y$ and $x > 3$ are imposed constraints. Thus, it can be inferred that $y \geq 4$. If later the constraint $y \leq 3$ is added, a contradiction can be immediately detect. Thus, each constraint has assigned a filtering algorithm that can reduce domains of variables involved in constraints. The filtering algorithm removes the values that cannot take part in any feasible solution. This algorithm is evoked every time a variable's domain is changed, and this change is propagated to domains of other variables (Barták, 1999). Several techniques have been developed to propagate constraints and reduce domains. Among these, the most well-known is the *arc-consistency*, which is exploited by Brailsford et al. (1999).

Figure 2.1, which is adapted from Barták (2002), illustrates the constraint propagation and the domain reduction mechanisms. In this figure, it is initially considered that the domains of variables x, y, and z are $D_x = D_y = D_z = \{1, 2, 3, 4, 5\}$. As some hypothetical model constraints are imposed ($x < y$ and $z < x - 2$), it immediately leads to a significant reduction in the domain of all variables. In this simple example, the domains of x, y, and z become singleton ($D_x = \{4\}$, $D_y = \{5\}$, $D_z = \{1\}$), and the values for x, y, and z are promptly determined.

The mechanisms of constraint propagation and domain reduction can be applied to reduce the search space of variables. However, while this may determine whether a model is infeasible, it does not necessarily find solutions to the model. In

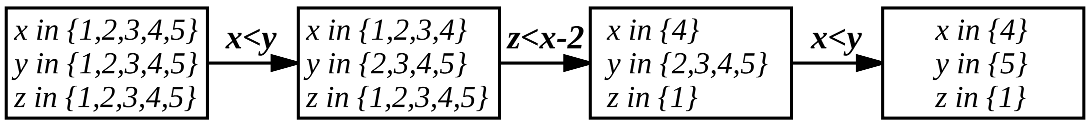


Figure 2.1: Constraint Propagation and Domain Reduction: A Simple Example.

order to do this, one must program a *search strategy*. Traditionally, the search facilities provided by a constraint programming system have been based on tree search (see section 2.3) with depth-first strategy for node selection. Depth-first has been used because, in the context of computer programming, the issues regarding memory management are dramatically simplified. Nevertheless, there are other alternative strategies for node selection such as, best-first search, limited discrepancy search, depth bounded discrepancy search, and interleaved depth-first search (Lustig and Puget, 2001). In Smith (1995), a n-queens example is considered in detail in order to show the effect of different search strategies.

Variables in CLP include types such as integer, boolean, symbolic, rational numbers, real numbers. Thus, the set is richer than MILP, which would only include integer, boolean, and real number types. In particular, the CLP field denominated Finite Domain Constraint Logic Programming (FD-CLP) has developed specialized algorithms to deal with variables that present finite-valued domains (Brailsford et al., 1999).

Constraints in CLP can be of several types: mathematical constraints ($a+b = c$), conjunctive/disjunctive constraints (tasks A and B occur at different times), relational constraints (at most two jobs should be allocated to machine number three), explicit constraints (the pair (x, y) must be $(1, 2)$ or $(2, 3)$ or $(4, 5)$), unary constraints (z is an integer between five and ten). Again, the set of types of constraints is essentially richer than the MILP set. In MILP all constraints must, by convention, be linear. Within CLP, constraint operators such as $=$, $\geq$, $\leq$, $>$, $<$, $\times$, $\div$, *subset*, *union* ($\cup$), *intersection* ($\cap$), boolean *or* ($\vee$), boolean *and* ($\wedge$), boolean *xor* ($\otimes$), negation ($\neg$), equivalence ($\leftrightarrow$), implication ($\rightarrow$) are all permissible. In general, CLP constraints can be translated into MILP constraints involving integer and linear variables, with varying degrees of difficulty. However, CLP admits the types of listed operators in their immediate form, without translation. Thus, CLP is stronger than MILP in its expressive power (Williams and Wilson, 1998).

Another important expressiveness feature of CLP relies on the development of *global constraint* structures. Global constraints capture interesting substructures of a problem, serving to the model builder as building blocks of problem statements. Operationally, global constraints are software components inside the solver. They encapsulate dedicated inference algorithms based on feasibility reasoning. Global constraints infer which values are infeasible for a variable with respect to the constraint-structure they represent, and provide information to the search process on the most viable course. The most classical example of a global constraint is the *alldifferent* predicate, which is largely used in scheduling problems (Williams and Wilson, 1998). As it can be inferred from its name, the *alldifferent* global constraint states that each variable takes a different value chosen from its domain (e.g. the constraint $alldifferent(x, y)$ states that $x \neq y$). According to Williams and Wilson (1998) this global constraint is awkward to express in MILP. In the special case of a disequality constraint involving two variables (e.g. $x \neq y$) the usual MILP formulation requires the introduction of a "small quantity" denoted by ε ($\varepsilon > 0$), so that if $x \geq y + \varepsilon$ or $x \leq y - \varepsilon$ then $x \neq y$.

One reason for introducing global constraints is that they allow the model builder to represent a problem in a more "natural" and compact manner. The availability of non-linear and high-level constraints gives a great scope to express models, extending the potentialities of the modeling language. If practitioners can write their models with as many global constraints as possible, this means that they can change their modeling habits. However, the more important feature is that global constraints open up the possibility of including structure specific propagation into a general solver, which is extremely important for the solver computational efficiency (Ottosson et al., 2001). Thus, the CLP modeling environment allows incorporating some problem-specific features, which are used in dedicated solver algorithms. There are different global constraints, such as *atleast*, *atmost*, *exactly*, *cumulative*, *element*, *circuit*, *sequence*, etc, which are described, for instance, by

Beldicenau and Contejean (1994) and by Hooker (2000).

The classical example of the Traveling Sales Person (TSP) can be used to evidence the modeling benefits of using global constraints. Supposing that a salesperson wants to visit a set E of cities, every city exactly once, such that the cost of the tour is minimized. Let c_{ij} be the cost of going directly from city i to city j, and let $x_{ij} = 1$ if the tour goes directly from i to j, 0 otherwise. Then the problem can be written in the classical MILP approach, as illustrated in formulation 2.1. In this formulation, the objective function is the first expression, which states the cost minimization. The second and third equations ensure that exactly one edge of the tour enters, and exactly one leaves each node (city). The fourth inequality is the subtour elimination, ensuring that selected edges represent a Hamiltonian tour[1], and the last statement defines x as a binary variable (Thorsteinsson, 2001).

$$\begin{aligned}
min \quad & \sum_{i,j} c_{ij}x_{ij} \\
s.t. \quad & \sum_{i} x_{ij} = 1 \quad \forall j \\
& \sum_{j} x_{ij} = 1 \quad \forall i \\
& \sum_{i,j \in S} x_{ij} \leq |S| - 1 \quad \forall S \subsetneq E \\
& x_{ij} \in \{0, 1\} \quad \forall i, j
\end{aligned} \tag{2.1}$$

In a CLP modeling framework, variables can appear in subscripts, which can be of great value to the model builder. Thus, the factor $c_{ij}x_{ij}$ that appeared in formulation 2.1 can be replaced by the factor c_{jy_j}, using the variable subscription technique. In this case, y_j is a general integer variable, which represents the city in the tour after city j. The look up for coefficients (e.g. y_j in c_{jy_j}) is handled by a global constraint denominated *element* (Bockmayr and Kasper, 1997). The TSP formulation can be also dramatically simplified by the use of the global constraint *circuit*, which states that y_1, y_2, ..., y_E describe a Hamiltonian tour. The final TSP formulation, within a typical CLP framework, can be seen in 2.2. Hooker (2003)

[1]A path through a graph that starts and ends at the same node and includes every other node exactly once (ILOG, 2002a).

makes a detailed algorithmic explanation about formulation 2.2. The TSP remains an NP-complete problem. The difference is that the MILP approach uses a complex model and a generic solution algorithm; in the CLP approach, the model is simpler, but the design of the propagation algorithm is more complicated. The work is shifted from designing the model to designing the algorithm (Thorsteinsson, 2001).

$$\begin{aligned} min \quad & \sum_j c_{jy_j} \\ s.t. \quad & circuit\{y_1, y_2, \ldots, y_E\} \end{aligned} \tag{2.2}$$

CLP has been the basis for many commercially successful systems, which model and solve real–life problems by using constraint (logic) programming techniques. In particular, CHIP was the first CLP language to employ constraint propagation (Hentenryck, 1989). Other examples of CLP-related systems are the constraint handling libraries of ILOG (2001a) and COSYTEC (1995), and the CLP languages Prolog III (Colmerauer, 1987), CLP(R) (Jaffar and Maher, 1994), ECL^{i}PS^e (Wallace et al., 1997), CIAO (Hermenegildo, 1997), and clp(fd) (Codognet and Diaz, 1996).

Many hard applications have been successfully tackled using constraint related technologies. Rossi (1999) provides a description of CLP applications in areas such as network management, scheduling, transport problems, personnel assignment, controlling electro-mechanical systems, constraint-based spreadsheets, interactive problem solving, graphical interfaces, over-constrained problems, circuit verification, resource allocation, timetabling, and many other combinatorial problems.

2.2 Introduction to MILP

Combinatorial Optimization Problems (COPs) are solved by choosing an optimal member (according to some well-defined criterion) of a finite, but often huge, collection of alternatives. The MILP formulation of COPs is based on algebraic specification of a set of feasible alternatives, as well as the objective criterion for comparing alternatives. This is achieved by (Chandru and Rao, 1996):

(i) Introducing discrete-value decision variables (often 0-1 variables);

(ii) Expressing the criterion as a linear function of variables;

(iii) Representing the set of feasible alternatives as the solutions to a conjunction of linear equations and inequalities on variables.

Therefore, MILP provides a general framework for modeling a large variety of problems such as planning, scheduling, packing, network design, supply chain management, etc. However, MILP models remain challenging from a computational standpoint: they are NP-complete or worse (see Garey and Johnson, 1979), and it is widely believed that no general and efficient algorithm exists for solving them (Chandru and Rao, 1996).

Expression 2.3 defines a general MILP formulation. In this expression, the c_j's and the c_i's are referred to as cost coefficients, the a_{kj}'s are referred to as constraint coefficients on continuous variables x_j's, the a_{ki}'s are referred to as constraint coefficients on integer variables y_i's, the b_k's are referred to as requirements[2], J is the set of continuous variables, I is the set of integer variables, and K is the set of constraints. The symbol ρ denotes mathematical relations, such as, $\leq$, $\geq$, and $=$. A maximization model can be written as a minimization model by multiplying the objective by (-1) and minimizing it.

$$\begin{aligned} min \quad & \sum_{j \in J} c_j x_j + \sum_{i \in I} c_i y_i \\ s.t. \quad & \sum_{j \in J} a_{kj} x_j + \sum_{i \in I} a_{ki} y_i \; \{\rho\} \; b_k \quad \forall k \in K \\ & x_j \geq 0 \quad \forall j \in J \\ & y_i \in Z_+ \quad \forall i \in I \end{aligned} \tag{2.3}$$

MILP has been the subject of many books (e.g. Nemhauser and Wolsey, 1988; Wolsey, 1998; Williams, 1999) and survey articles (e.g. Chandru and Rao, 1996; Sherali and Driscoll, 2000; Bixby et al., 2000), which deeply exploit theoretical topics such as simplex and interior point methods, column generation, branch-and-bound, branch-and-cut, relaxations, polyhedron theory, etc. A practical and concise point of view can be obtained in Magatão (2001). The author comments the resolution

[2]It is customary to gather all the constants to the right-hand side of an MILP constraint.

of MILP models, and he highlights aspects concerning MILP modeling techniques (number of binary variables, number of constraints, Big-M formulation, logical conditions, etc). The author also develops detailed MILP formulations of a problem-specific environment, demonstrating the potential and functionality of this OR modeling/solving technique.

MILP is built on top of Linear Programming (LP) by adding the option of integrality requirements of some variables (or all for ILP models). Most commonly, 0-1 variables represent no-yes, off-on, or false-true choices (decision variables), or they are linked to some of the continuous variables (for details see Big-M formulations in Schrage, 2000). A classical example of a 0-1 program is the Traveling Salesperson Problem (TSP), previously illustrated in formulation 2.1 on page 12. The underlying linear form is a natural part of the solution process of MILP models. The linear programming relaxation is obtained by "ignoring" the integrality restrictions on integer variables. Therefore, the art of MILP rests on useful interplays between search methodologies and linear programming relaxation (Chandru and Rao, 1996). The effectiveness of MILP methods depends on both: (i) the size of LP subproblems; (ii) the *integrality gap*[3]. Thus, at the core of any MILP off-the-shelf software, a(n) (efficient) linear programming code is indispensable.

MILP modeling techniques allow the expression of some logical conditions between 0-1 variables (e.g. *or*, *and*, *not*, and *xor*) by means of linear constraints (LINDO, 2002). This restricted form of problem representation has aided the formalization and modeling of problems. However, the lack of high-level modeling constructs for disjunctions and combinatorial constraints makes the process of modeling hard and error-prone. Thus, MILP programmers are becoming more aware of inequality-based modeling limitations.

There exist, however, one example of a "global constraint" in MILP: the Special Ordered Sets (SOS), which were primarily introduced by Beale and Tomlin (1970). This constraint comes in two different forms: SOS-1, which is a set of variables within

[3] $|(z_r - z_o)/z_o|$, where z_r is the initial linear programming relaxed solution and z_o is the optimal solution. The smaller the relaxation gap, the faster the model tends to be solved. For an in-depth explanation see, for instance, Wolsey (1998).

exactly one variable must be non-zero, and SOS-2, which is a set of variables within two variables can be non-zero, but they must be adjacent in the ordering. These conditions can be modeled using linear inequalities, but there is great computational advantage to be gained from treating these restrictions algorithmically (Nemhauser and Wolsey, 1988; Williams, 1999). Examples where the SOS conditions might be useful include depot siting, where a depot can be built at only one of a number of possible positions (the SOS-1 condition), and in piecewise linear optimization, where each piece can be modeled as the convex combination of its endpoints (the SOS-2 condition). The SOS is, generally, a built-in feature in the most popular MILP solvers.

MILP modeling takes place within fully declarative modeling languages, such as AMPL (Fourer et al., 1990), GAMS (Brooke and Meeraus, 1982), LINGO (LINDO, 2002), XPRESS-MP (Guéret et al., 2002), and MPL (Maximal, 2005). The model builder can state the model without describing the solution procedure, and the model can be passed to different solvers by means of standard formats (for details see MPS matrix in Schrage, 2000). High-level modeling languages often provide global constructs or simplified syntax to aid in modeling process. These constructions include sum, set membership, set cover, vector operators, etc. In the end, however, the high-level model is passed to a matrix generator and only the matrix is passed to the solver. Thus, at the time the model is compiled, all the high-level constructs must be determined by the matrix generator, since an MILP solver cannot process them during the tree search. Thus, much of the problem structure is lost before the problem reaches the solver algorithms. Furthermore, the black-box structure of MILP solvers does not allow the users to directly interfere in the search process[4].

In spite of the "disadvantages" of the black-box structure of MILP solvers, the improvement of MILP codes (see Bixby, 2002) has contributed to widespread the use of this OR optimization technique in real-world applications, with notorious results. Kalvelagen (2003), for instance, reports the resolution of a large-scale MILP progressive party problem. Examples of MILP models for difficult scheduling problems

[4]Indeed, the user can manage some solver settings that, in theory, may influence the search process conduction.

are given by Pekny and Reklaitis (1998), Shah (1998), and Pinto and Grossmann (1998). Nevertheless, there is still a great number of hard problems (Garey and Johnson, 1979), whose resolution would bring great benefits in day-to-day situations. Goldbarg and Luna (2000) describe some of these situations. In particular, the integration of deeply studied MILP techniques with CLP technology seems to offer a better chance of success in solving some hard COPs than the state-of-the-art of either CLP or MILP techniques in separate frameworks (Heipcke, 1999). Therefore, the integration CLP-MILP is a promising field[5]. Section 2.3 conveys the fundamental principles of CLP and MILP that guide the main opportunities for integrating the techniques.

2.3 Search, Relaxation, and Inference

Although developed in different communities, with different perspectives, Constraint Logic Programming (CLP) and Mixed Integer Linear Programming (MILP) share the same fundamental principles: search, relaxation, and inference . These principles guide the main opportunities for integrating the techniques and, thus, are herewith explained.

Search is the process of systematic exploration of a set (or space) of possible solutions. In a simplified point of view, the search goal is to assign values to variables to satisfy given constraints. A search algorithm is said to be exact if it finds a solution (given enough time), in case of the solution exists; otherwise, the algorithm reports the lack of solutions. For combinatorial optimization, most exact algorithms are based on tree search, that is, they implicitly define a tree, where internal nodes correspond to partial solutions, branches are choices (partitioning the search space), and leaf nodes are complete solutions (Ottosson and Carlsson, 1997).

The branch-and-bound algorithm (Land and Doig, 1960) is an exact tree-based search used in both CLP and MILP, although in slightly different shapes. In CLP, the branching is combined with inference, which is aimed at reducing the amount

[5]Chapter 3 further investigates the computational advantages of the hybrid approach CLP-MILP in comparison with the root techniques.

of choices to be exploited. In MILP, the branching is intertwined with a relaxation analysis. This procedure eliminates the exploitation of nodes for which the relaxation value is worse than the best solution found so far.

Within the class of tree-based search algorithms there exist numerous variations. Classically, one identifies three choices to make in each algorithm step:

(i) What node to continue exploiting;

(ii) What variable to branch on; and,

(iii) What (set of) values to restrict it to.

In addition to tree search, there are various schemes of approximate search, such as genetic algorithms and neighborhood search. These iterative procedures maintain one or more complete solutions, which are gradually improved. These algorithms usually scale well on hard problems, but they can stuck in local optima and, therefore, not guarantee to ever find the globally best solution (Glover and Laguna, 1993).

A **relaxation** of a model, in a simplified point of view, is a formulation that includes the set of feasible solutions to the model. The relaxation provides a lower bound (assuming minimization), since the model optimal solution cannot be better than the relaxation "best" solution. The relaxation also provides a point in space around which the search can be centered and, in case of a good relaxation, this point is close to the true solution. This fact is exploited in traditional MILP branch-and-bound search, which branches on fractional values that violate the integrality requirements. The relaxation should (to be useful) satisfy two criteria (Thorsteinsson, 2001):

(i) It should be faster to solve to optimality than the original formulation; and,

(ii) Its "solution structure" should resemble the original model as closely as possible (in order to provide strong bounds).

According to Hooker (2002), a clear distinction between MILP and CLP arises in the relaxation feature. In general, MILP packages are built on the top of efficient linear programming (LP) codes, which provide the "global relaxation" of a model[6]. On the other hand, CLP lacks such global relaxation because constraint propagation and domain reduction mechanisms operate "locally" in each constraint. Thus, when looking for an optimal solution instead of just any feasible solution, the power of LP plays a significant role.

An **inference** method attempts to derive a desired implication from a set of constraints. Various forms of inference can occur in algorithms for combinatorial optimization. In general, inference strengthens the model by adding valid constraints, thus, reducing the search space. Inference acts in CLP by reducing domains (the possible values of variables); in MILP, inference eliminates fractional regions of the space solution by adding cutting planes. A cutting plane, in a simplified point of view, is a logical implication of a set of inequalities (this issue is deeply addressed by Wolsey, 1998). Inference in CLP has traditionally been focused on feasibility; constraint propagation removes only infeasible elements from domains, and not suboptimal ones. On the contrary, inference in MILP has been optimization oriented (e.g. cutting planes). Moreover, MILP preprocessing algorithms, which eliminate variables or strengthen constraints, can be regarded as inference methods (Thorsteinsson, 2001).

Inference in CLP and MILP takes different forms due to the difference between the underlying solution techniques: constraint propagation versus linear programming. Hooker (2002) highlights that inference can be very ineffective in CLP when an equation contains many variables. Cost and profit functions tend to contain many variables, representing the many activities that influence the cost or contribute to profit. This fact often makes optimization problems hard to CLP. The OR community escapes from this impasse by using continuous relaxations.

[6]Such "global relaxation" is obtained by dropping integrality requirements and solving an LP resulting model.

2.4 Remarks on Chapter 2

Sections 2.1, 2.2, and 2.3 have established a background knowledge about the fundamentals of CLP and MILP, and described the underlying principles of both techniques (search, relaxation, and inference). The next chapter (chapter 3) assumes this background knowledge, and surveys a literature that is focused on the integration of constraint programming devices and (mixed) integer programming techniques.

Chapter 3

CLP–MILP: Introduction to a Cluster of Articles

This chapter aims at presenting a cluster of articles that have contributed to the integration of CLP and MILP.

3.1 Comparisons between CLP and MILP

Several papers compare CLP and MILP and propose schemes for integrating the underlying fundamental principles of both techniques (search, relaxation, and inference). Sections 3.1 and 3.2 try to present the main contributions to this integration process[1].

Darby-Dowman et al. (1997) apply CP and IP to a set of real-world manufacturing assignment problems. Models for both CP and IP are introduced and discussed. The CP formulation is found to be closer to the problem structure, and facilitates the application of heuristic searches and redundant constraints. Furthermore, the mechanism to control the search is more powerful in the CP solver, which turns out to be valuable. For these problems, CP outperforms IP in both: CPU

[1]The reader should be aware that some articles herein clustered discuss the merge between CP and IP rather than CLP and MILP. However, this chapter does not make distinctions between CP-IP or CLP-MILP combinations. On the contrary, the integration of mathematical programming techniques with constraint programming devices is broadly relevant, and, therefore, articles conveying this subject are of interest.

time and robustness. The IP solver demonstrates difficulties to prove the solution optimality. The paper also discusses and illustrates how CP could be integrated with IP to increase the performance of the IP solver, but this integration is not deeply exploited.

Darby-Dowman and Little (1998) expand the investigations made by Darby-Dowman et al. (1997) and add three more problems for study. The new problems are a golf scheduling problem, a crew scheduling problem, and a flow aggregation problem. The results confirm some of the previous acclaimed properties of the techniques. The golf scheduling problem is highly constrained, with an awkward IP formulation with many variables and a weak linear relaxation, which makes the problem hard to be solved with IP. In contrast, the CP formulation is more compact, and CP solves the problem in a few seconds using a customized search procedure. The opposite is true for the crew scheduling problem, which is easily solved using IP, and where CP cannot find any good solution. The flow aggregation problem is also easily solved with IP, while CP only produces suboptimal solutions in the same time.

The article of Proll and Smith (1998) describes the use of IP to solve a practical problem arising in the printing industry. Solution of the model proves elusive. The problem is then modeled using CP in order to assess whether certain deficiencies of the IP approach can be remedied. The article demonstrates some strengths of the CP approach, and makes suggestion for generalizations of the ideas.

Yunes et al. (2000) present a crew rostering problem and solve it by means of IP and CP approaches. Both models are discussed in detail. Lower bounds obtained with LP relaxation are used to evaluate the quality of solutions. Furthermore, the article presents a hybrid column generation approach that combines IP and CP. Experiments are conducted upon real data sets, and the computational results of the three solution methods are compared.

The article of Brailsford et al. (1996) contrasts IP and CP for the solution of a specific discrete optimization problem. This problem arose in the course of organizing the social program at a yacht rally. The problem proved to be difficult, although it does not belong to a known category of combinatorial optimization problems (it poses some similarities with school timetabling). The problem proved intractable to IP alone, and CP was unable to find good (near optimal) solutions. However, a combination of LP with CP provided a very fast method to find optimal solutions of different problem instances.

Another well-studied example is the Progressive Party Problem (Smith et al., 1996). This problem is a quite clear case when the linearity requirements of IP force a very large and weak relaxation. In comparison, CP has a more compact model, where variables have a direct mapping to the original problem decisions. The latter allows better search strategies to be implemented. This fact, in combination with the smaller model, makes CP perform better on this problem. Hooker and Osorio (1999) have also studied this problem, and a model using global constraints is given by Kay (1997).

The article of Brailsford et al. (1999) is a comprehensive review of CSPs, their typical applications and solution methods. Moreover, this article also evaluates CP as a technique for solving CSPs and compares it with IP. The criteria used for evaluation include facility of implementation, flexibility to handle a variety of constraints that occur in practical problems, computational time, and solution quality. The article concluded that CP compares favorably in terms of ease of implementation and flexibility to add new constraints. Its performance with respect to solution quality tends to be problem/data dependent.

3.2 Modeling and Solving in a Combined CLP-MILP Approach

According to Focacci (2000), at least two main and complementary streams for the integration of CP and OR modeling/solving techniques can be considered. The first one is *an algorithmic approach.* Several studies follow this stream on different directions. An incomplete list of directions is the following: OR methods can be used to remove uninteresting (infeasible or suboptimal) values from the domain of variables within a CP framework; logical methods can be used to deduce valid inequalities within a branch-and-cut framework; local search algorithms can be used within a CP framework to speed up the convergence towards good solutions; consistency algorithms can be used to some variables in an Integer Linear Program (ILP) during a preprocessing step. The second stream is called *an engineering approach.* New optimization languages such as OPL (ILOG, 2001b) propose a clear separation between problem modeling and problem solving. In these languages, the model does not imply a solving technology, but it is just a problem description. In general, software engineering practice can be applied to OR approaches leading to more flexible and modifiable code. In this sense, CP can be understood as software engineering applied to OR (Puget, 1994).

In CLP, modeling has traditionally been done within a programming language. In MILP, the problem is specified by the use of algebraic modeling languages, and then translated to a matrix form. Barth and Bockmayr (1995) show that the basic functionality of algebraic modeling languages can be realized in a CLP language. Fourer (1998) proposes an extension of the algebraic modeling language AMPL that incorporates modeling devices from constraint programming, but the author does not address the issue of how solvers might cooperate to handle this extension.

COSYTEC, a French optimization company, has recently announced the integration of CHIP V5 (COSYTEC, 2005) constraint programming technology with the leading edge linear programming engine from Dash Optimization company, the XPRESS-MP (Guéret et al., 2002). The companies argue that the headway gained by global constraints and the linear relaxation is made available in an intertwined kernel, which is called XPRESS-CP.

Other approach is to combine CLP and MILP techniques by means of user-written interface programs. Rodošek et al. (1999) combine the finite domain constraint solver ECL^{i}PS^e (Wallace et al., 1997) with the well-known mathematical programming solver CPLEX (ILOG, 2001d). Heipcke (1999) used the mathematical programming technology of the XPRESS-MP associated with the SchedEns (Colombani and Heipcke, 1997), a constraint programming system for solving numerical constraints over unions of integer intervals.

A recent algebraic modeling system called Optimization Programming Language (OPL), invokes both linear programming (ILOG Planner/CPLEX) and constraint programming (ILOG Solver) solvers. OPL is the core of ILOG OPL Studio, an interactive modeling environment, which lets the user state optimization models without the low-level complexities of ordinary programming languages. OPL allows using both CLP constraints and a linear relaxation, although the interface allows only limited forms of interaction. Links between CLP and LP are one-to-one mappings of variables. Model-specific search can be defined, overriding the default. A review of the ILOG Optimization Suite can be obtained in ILOG (2001c).

Some research effort has been aimed at incorporating better support for symbolic constraints and logic in IP. Hajian (1996) shows how disequalities can be handled (more) efficiently in IP solvers. Furthermore, the author gives a linear modeling of the *alldifferent* constraint by means of introducing non-zero variables. This restriction is not expressed as constraints in the model, but rather handled implicitly through an extension to the branch-and-bound algorithm. This is similar to how SOS variables are usually treated[2]. The expression of logical relationships in 0-1 (in)equalities is a well-studied subject, such as presented by McKinnon and Williams (1989) and Williams (1995). Mitra et al. (1994) present a tool for expressing these (in)equalities. Logical relationships expressed in (in)equalities can be solved in an LP solver and used for complementing logical constraints in a CLP framework. Many schemes for logic-based branching in IP (e.g. Raman and Grossmann, 1993; Raman and Grossmann, 1994) can be seen as a step towards CLP, whether they were the source of inspiration or not (Hooker, 2002).

Bockmayr and Kasper (1997) develop a branch-and-infer framework to unify solution algorithms. The authors suggest ways in which algorithms for both types of approaches may be combined. They extended the classes of constraints that the algorithms can handle, either in relaxed or exact form. They propose a framework for combining finite domain CP and IP in which several approaches to integration or

[2]Special Ordered Sets (SOS) are explained in section 2.2.

synergy are possible. They investigate how symbolic constraints can be incorporated into IP, much as cutting planes are. The authors also show how a linear system of inequalities can be used in CP by incorporating it as a symbolic constraint. Furthermore, they discuss a closer integration in which linear inequalities and domains appear in the same constraint store.

Rodošek et al. (1999) use CLP along with LP relaxations in a single tree search. This integration tightens bounds rather than adds constraints during the search. The integrated system combines components of the CLP system ECL^{i}PS^e (Wallace et al., 1997) and the MILP system CPLEX (ILOG, 2001d). The model builder may annotate constraints to indicate which solver should handle them (CLP, MILP, or both). The hybrid algorithm reduces the solution space by calling finite domain propagation of ECL^{i}PS^e, as well as dual simplex of CPLEX. A node can fail by two different causes: (i) propagation produces an empty domain; (ii) the LP relaxation is infeasible or has a value that is worse than the "temporary optimal" solution value. Moreover, the linear relaxation can be strengthened by adding cutting planes. The results of such integration are illustrated by efficiently solving difficult optimization problems (the progressive party problem and the hoist scheduling problem), while neither the CLP nor the MILP solvers were able to solve them in reasonable time.

Carlsson and Ottosson (1999) compare CP, IP, and a hybrid algorithm for a configuration problem. Linear relaxations, branch-and-bound search, cutting planes, and preprocessing are experimented in both, pure and mixed variants. Computational experiments show that, for this problem, linear relaxations and cutting planes are the most important factors for computational efficiency.

There are also approaches where the constraint programming and mathematical programming models are not merged. Heipcke (1999) proposes a scheme where two different models, which are formulated according to finite domain CP and MILP techniques, are separately solved in two synchronized search trees. The models are linked by variables, which provide information exchange between solvers. This approach is called *double modeling*: a problem is formulated in both systems, and communication modules connect corresponding decision variables. During the search, each software handles its own tree search, and the communication modules coordinate the two-part status. The reasoning behind this approach is that each software is implemented in a way that best suits the specific needs of its genuine techniques. In the finite domain CP framework, the constraint propagation mechanism dictates the system architecture, whereas in MILP it is necessary to handle the matrices in an efficient way.

Jain and Grossmann (1999) present a scheme where the problem is decomposed into two sub-parts, one handled by IP and one by CP. This is exemplified with a multi-machine scheduling problem, where the assignment of tasks to machines is formalized as an IP, and the sequencing of tasks on the assigned machines is handled with CP. The implemented search scheme is an iterative procedure, where first an assignment model is solved to optimality (identifying which machine to use for each task), and then a feasibility model is solved by CP, trying to sequence tasks according to the previous assignment. If the sequencing fails, cutting planes are added to the IP formulation, and the process is iterated.

Harjunkoski et al. (2000) combine mathematical programming and CLP in the area of job-shop scheduling and trim-loss problems. They propose two hybrid approaches: first, they consider a decomposition strategy in which they iterate between an MILP master model and a feasibility CLP model; second, they reformulate the solution of a bilinear MINLP (Mixed Integer Non-Linear Programming) model as the solution of an MILP, followed by the iterative solution of a feasibility CLP submodel. The authors compare the two hybrid approaches, and the numerical results are encouraging.

Harjunkoski and Grossmann (2002) present two strategies to reduce the combinatorial complexity of solving single stage and multistage optimization scheduling problems that involve cost minimization, due dates, and sequence independent setup times. The problems are decomposed into assignment and sequencing subproblems. The proposed strategies rely on either combining MILP to model the assignment part and CLP for modeling the sequencing part, or combining MILP models for both parts. Results are presented for both single and multistage systems. The authors stated that MILP evaluates all constraints simultaneously, while CLP evaluates the effect of constraints sequentially, by communication through the domain of variables. Consequently, it is generally difficult to obtain the optimal solution for loosely constrained CLP models. On the other hand, MILP methods require all constraints to be linear equalities or inequalities due to the LP based model. This restriction does not apply for CLP formulations.

In some models constraint programming solvers can be more efficient for finding a satisfying solution, but not perform as well as MILP for finding and proving optimality. Hajian et al. (1995) describe how the ECL^{i}PS^e (finite domain) constraint system is used to find a good feasible initial solution to a fleet assignment problem. The initial solution is then used to "warm-start" a traditional branch-and-bound MILP solver that, on its own, had problems in finding a starting point.

Apart from the various optimization schemes centered on (Mixed) Integer Programming, the OR community has developed a wide range of algorithms for specific problems, such as scheduling, routing, allocation, packing, and network flows. Some efforts have been taken in order to encapsulate these algorithms in global constraints of CLP (e.g. Baptiste et al., 2001).

The computational studies of hybrid solvers rise a quite large body of evidences that a hybrid CLP-MILP approach can bring both modeling and algorithmic advantages. It includes computational studies involving chemical process design (Raman and Grossmann, 1991; Türkay and Grossmann, 1996), distillation network design (Grossmann et al., 1994; Raman and Grossmann, 1994; Hooker and Osorio, 1999), truss structure design (Bollapragada et al., 2001), machine scheduling (Heipcke, 1999; Jain and Grossmann, 1999), highly combinatorial scheduling (Hooker and Osorio, 1999; Rodošek et al., 1999), production planning and transportation with piecewise linear functions (Thorsteinsson, 2001), warehouse location (Bockmayr and Kasper, 1997), traveling salesman problem with time windows (Focacci, 2000), to name a few.

3.3 Other Hybrid Approaches

In addition to the integration of CLP and MILP, other hybrid approaches have been investigated. However, it is beyond this thesis' scope to survey other directions that have gained interest. In particular, two promising fields are:

(i) The combination of Local Search (LS) and CP, which is explored, for example, by Pesant and Gendreau (1996) and De-Baker et al. (2000);

(ii) The combination of Generalized Disjunctive Programming (GDP) with MILP (e.g. Pinto and Grossmann, 1997; Vecchietti and Grossmann, 2000).

GDP is based on the idea of representing discrete and continuous optimization problems through equations, disjunctions, logic propositions, and also boolean and continuous variables. CLP is similar to GDP because it also involves equations, logic statements and disjunctions. However, the most important difference is that CLP has high-level procedural constructs (global constraints), which aid the formulation of models in compact forms. In contrast, GDP gives rise to declarative models, which are expressed through explicit equations. Furthermore, GDP models can be solved through branch-and-bound, or reformulated as MILP models; CLP models are solved with the aid of implicit enumeration techniques: domain reduction and constraint propagation (Harjunkoski and Grossmann, 2002). Vecchietti and Grossmann (2000) made a comparison between CLP and GDP with several constraint transformations from CLP to GDP.

3.4 Mixed Logical Linear Programming (MLLP)

One of the most promising directions for integrating MILP and CLP techniques is called Mixed Logical Linear Programming (MLLP). MLLP is a general approach to formulating and solving optimization problems, which have both discrete and continuous elements. The motivation behind MLLP is that many mixed discrete/continuous models are traditionally conceived as continuous models in which some of the variables are restricted to be integers. However, many integer variables are often used only to model a given logical constraint, and serve no real purpose in the model relaxation. Therefore, in the MLLP framework, discrete functions and continuous functions are separated.

MLLP has been deeply explored by Hooker and Osorio (1999), Harjunkoski et al. (2000), and Thorsteinsson (2001). The authors advocate that, to get the best from both CLP and MILP, it is necessary to reconsider the modeling process of CLP and MILP. They suggest using neither the CLP model nor the MILP model but rather a whole new model, designed specifically for a hybrid solver. This approach roughly separates the model into a discrete part, the Finite Domain Store (FD-store), and a continuous part, the Linear Programming Store (LP-store). In this model, one can achieve domain reduction on the FD-store using constraint propagation. Inference from the FD-store can be applied to the LP-store using bounds, cutting planes, and a natural relaxation (the LP relaxation). The authors argue that the key to effective integration lies in the design of the modeling language, and their aim is to design a model that can suit the traditional solvers rather than adjust the solvers to suit the traditional models.

MLLP assumes that an LP solver and a branching mechanism are available. Its constraints are written as conditional statements of the form $D \Rightarrow C$, where the antecedent D (Discrete) is a constraint involving discrete variables and the consequent C (Continuous) is a system of linear inequalities. This conditional structure relates to a branching algorithm in a natural way. At each node of the branching tree, the values of the discrete variables may be fixed or restricted in such a way as to satisfy some of the antecedents D. The corresponding consequents C form an LP constraint set that is passed to an LP solver. Checking whether partially determined discrete variables satisfy the antecedents is an inference problem that can be attacked with constraint propagation and domain reduction methods, developed within a CLP framework. The MLLP language architecture not only provides the useful modeling devices associated with constraint satisfaction, but also captures continuous and discrete elements in a way that appears convenient for a wide range of problems (Thorsteinsson, 2001).

In an MLLP framework, CLP can enhance the search by reducing domains of variables, tightening the linear relaxation by adding bounds and cuts (in addition to classical cutting planes), and eliminating search for symmetric solutions. MILP, on the other hand, can be used to find relaxed solutions, and to prove infeasibility. In order to get the best from both methods, a whole new general model is defined, as illustrated in formulation 3.1.

$$\begin{aligned} min \quad & cx \\ s.t. \quad & h_i(y) \rightarrow A^i x \geq b^i \quad i \in I \\ & x \in R^n \quad y \in D \end{aligned} \tag{3.1}$$

In formulation 3.1, c is a cost vector, x is a vector of n continuous variables, A^i is a set of constraint coefficients, b^i is a set of constraint requirements, and y is a vector of discrete variables. D is the cartesian product of some discrete domains and I represents the set of constraints. The antecedents $h_i(y)$ are constraints that can be treated with CLP techniques. The consequents are linear inequality systems that can be inserted into an LP relaxation. A linear constraint $Ax \geq b$, which should be unconditionally enforced, may be written in the conditional form: "*true*"$\rightarrow Ax \geq b$. The absence of discrete variables from the objective function can be algorithmically useful. Costs that depend on discrete variables can be represented with conditional constraints. For example, the objective function $\sum_j c_j x_j$, where $x_j \in \{0, 1\}$, can be written $\sum_j z_j$, with constraints $(x_j = 1) \rightarrow (z_j = c_j)$ and z_j, $c_j \geq 0$ for all j (Thorsteinsson, 2001).

An MLLP model is solved by branching on the discrete variables. The conditionals assign different functions to CLP and LP algorithms. CLP is applied to the discrete constraints in order to reduce the search and help determine when partial assignments satisfy the antecedents. At each node of the branching tree, an LP solver minimizes cx subject to the inequalities $A^i x \geq b^i$, for which $h_i(y)$ is determined to be true. This delayed posting of inequalities leads to small and lean LP models that can be efficiently solved. A feasible solution is obtained when the truth-value of every antecedent is determined and the LP solver finds an optimal solution, subject to the enforced inequalities. More precisely, the algorithm conducts a tree search by branching on values of the discrete variables y_i. When branching on y_i, the algorithm defines each branch by restricting the domain of y_i to a different subset $\bar{D}_i$ of the original domain D_i. The variable y_i is fixed to a particular value when D_i is restricted to a singleton (Thorsteinsson, 2001). A more detailed description of the MLLP solution algorithm can be obtained in Hooker et al. (2000). Furthermore, Ottosson et al. (2002) introduce new symbolic constraints to MLLP, and show how these constraints can be used to model piecewise linear functions.

It is important to highlight that the MLLP framework does not demand the invention of new optimization methods. On the contrary, one may use all existing techniques for CLP and LP. In general, a solver based on MLLP should be able to compete with MILP solvers and CLP solvers. Some computational results for specific problem instances can be obtained in Thorsteinsson (2001).

3.5 Choosing CLP or MILP

Solution methods for both CLP and MILP usually rely on tree search, as previously explained in chapter 2. In CLP the core-engines are constraint propagation and domain reduction. MILP relies on setting bounds on integer variables and creating relaxation submodels that can be solved by LP methods. Whereas MILP alters bounds, CLP alters domains of variables. Some comparative studies between CLP and MILP implementations have been done during the last years, as previously stated in sections 3.1, 3.2, and 3.4, but no general guidelines have been established so far. Thus, it is difficult to give any general recommendation on using CLP, MILP, or a combined approach. Heipcke (1999) explains that a combined approach can suffer from inherited drawbacks, and performing poorer than the root techniques. Herewith are some observations for choosing CLP or MILP based on the analysis of articles presented in sections 3.1, 3.2, and 3.4.

There are some reasons to represent and solve a problem by CLP rather than by MILP. The two main ones are the following:

(i) The representation as a CLP model is often much closer to the original problem: the CLP variables directly correspond to problem entities, and the CLP constraints do not need to be linear. This makes the formulation simpler, the solution easier to understand, and the choice of good heuristics to guide the solution strategy more straightforward;

(ii) The CLP modeling vocabulary is richer than the MILP one. As a result, not all CLP expressions can be easily converted into an equivalent MILP form, and it might not be possible to represent a CLP model in an efficient MILP formulation.

In cases where the domains of variables are large or the constraints involve a large number of variables, constraint propagation may not be very successful leaving (presumably) large domains to be searched afterward. Sometimes the original formulation of a CLP model makes hard to determine if there is any solution at all. In these cases, it might be preferable to use an MILP approach to conclude whether a solution exists or not.

When the constraints are linear or easily translated to a linear form, integer programming might be preferred. In addition, when looking for an optimal solution to a given problem, and not just any feasible solution, the power of linear programming plays a significant role; the LP relaxation can be fruitfully used to establish bounds that help reducing the search space.

The research of Heipcke (1999) involves several examples of finite domain CP and MILP, and it demonstrates that each technique might be advantageous in some cases. Typically, MILP approaches are better for genuinely linear constraints (e.g. the knapsack constraints), whereas CP profits from any further knowledge that may be directly incorporated into the formulation of complex (non-linear) constraint relations. In Heipcke (1999), the author identified factors that influence the applicability and success of each technique separately or in combination. However, Heipcke highlighted that it is difficult to give a detailed decision rule merely based on the studied examples. Herewith are the main observations made by Heipcke (1999):

(i) If it is easy to construct feasible solutions based on partial instantiations, CP alone or in combination quickly leads to feasible solutions;

(ii) If the objective function can be estimated from a (small) subset of the decision variables, enumeration in CP prioritizing these variables quickly gives good solution estimates;

(iii) If the constraints are genuinely linear, the MILP approach may be better suited, but if MILP requires some kind of linearization, CP usually profits from a model that is closer to the original problem;

(iv) Search strategies may profit from a problem specific knowledge (CP preferable) or a structural point of view (MILP preferable);

(v) Valuable information to cut off branches can be obtained from the LP relaxation;

(vi) If the LP solution is close to being integral, CP based rounding heuristics may be successful in a combined approach;

(vii) It is valid to verify whether local or global techniques are more adapted to the nature of a problem: CP consistency algorithms operate "locally" on subsets of variables and constraints; MILP works on the matrix as a whole.

Lustig and Puget (2001) indicate that CLP tends to be better than MILP in applications that concern sequencing and scheduling, and for problems in which an MILP formulation contains much symmetry. In addition, strict feasibility problems are good candidates for applying CLP techniques. MILP seems to be superior for models in which the linear programming relaxations provide strong bounds for the objective function.

According to Hooker and Osorio (1999), it is likely that traditional optimization techniques such as MILP, CLP, and LS will be combined in many more ways in the future. Eventually, the border between different search strategies will become blurred, and search will become seamlessly intertwined with relaxations, inference and other techniques aimed at reducing the search needed. However, the authors highlight that such integration is dependent on the research interest of two fields:

(i) Development of robust and widely acceptable integration schemes for optimization and constraint programming;

(ii) Development of generic modeling languages to integrate CLP and MILP.

In the following chapter (chapter 4), the issue (ii) is somehow addressed. Chapter 4 surveys the directives for the development of high-level MILP modeling structures. These structures can be used to straightforward formulate MILP models. As the CLP formulation tends to be easily created, a combined CLP-MILP approach with high-level MILP structures certainly facilitates the model builder task, and can be seen as a step towards the integration of both techniques.

Chapter 4

MILP: Modeling Structures

This chapter surveys the directives for the development of high-level MILP modeling structures. These structures can be used to straightforward formulate MILP models.

4.1 Building MILP Models

Building models is one of the most creative aspects of Operations Research and it is presumed to be more of an art than a science. The representation of complex OR problems in an easy way has been a long-lasting concern, along with the identification of a suitable solution method for a particular problem. Furthermore, building LP/MILP models remains a challenging task because each model has its own unique features (Murphy and Panchanadam, 1997). Jeroslow and Lowe (1984) say that the automatic transformation of a user description to a formal MIP is one of the most difficult issues to be addressed in the OR field. Without an efficient formulation process, it is difficult to narrow the gap between model builders and decision makers. Therefore, either in a traditional MILP framework or in a combined CLP-MILP approach, the MILP formulation is a difficult task, mainly because it has implicit logical meaning incorporated into constraints.

Some attempts have been presented in the literature in order to translate qualitative specification of problems into MIP optimization models. In particular, this chapter henceforth investigates the parallels between logical inference and optimization that are aimed at helping the MILP modeling task. Nevertheless, the connection between logical inference and optimization methods is a broad field of study, which is, for instance, well discussed by Chandru and Hooker (1999).

Most of the pioneering work in the logic/optimization interface was done, notably, by Jeroslow and Lowe (1984), Williams (1987), and McKinnon and Williams (1989). Jeroslow and Lowe (1984) used IP to solve inference problems and introduced a number of other seminal ideas, as well as Williams (1987) did. McKinnon and Williams (1989) created a modeling language for IP based on nested representations of the "greater-than-or-equal" predicate. This procedure was implemented in Prolog (Colmerauer, 1987), but it was not integrated into a modeling system. Furthermore, Williams (1995) surveys the many connections between the methods of computational logic and integer programming. The author shows how computational problems arising in formal logic can be solved by IP, and how the methods of logic are applicable to modeling and solving IP.

The papers of Hadjiconstantinou and Mitra (1994) and Mitra et al. (1994) demonstrate that propositional logic statements can be expressed as linear equalities or inequalities involving zero-one variables. The authors argue that the real way forward to the model building process is to capture knowledge in the qualitative logic form and reformulate it in the quantitative discrete optimization form, which is amenable to solution by well-established and efficient computational methods. Sharing the same fundamental principle, Yeom and Lee (1998) describe logical operators to assist the formulation of IP models. These operators are subsequently transformed to a solvable conventional IP form. Hürlimann (1998) presents a procedure that translates logical constraints into mathematical constraints containing zero-one variables. The procedure has been integrated with the modeling language LPL (Hürlimann, 2000). LPL (Linear Programming Language) can be used to formulate models that contain mathematical and logical constraints. LPL subsequently translates logical constraints into the equivalent integer form.

Floudas (1995) presents some guidelines in modeling MI(N)LP. The author separates the guidelines in four topics: (i) modeling with 0-1 variables (including the use of propositional logic expressions); (ii) modeling with continuous and linear 0-1 variables (including activation and relaxation of constraints, either-or constraints, and constraint functions in logical expressions); (iii) modeling with bilinear products of continuous and 0-1 variables; (iv) modeling nonlinearities of continuous variables. These guidelines are exemplified in the chemical process synthesis context, and, unfortunately, they are just briefly discussed.

A series of papers of Raman and Grossmann (e.g. Raman and Grossmann, 1991; Raman and Grossmann, 1992; Raman and Grossmann, 1993; Raman and Grossmann, 1994) shows that qualitative knowledge expressed in propositional logic form has an equivalent representation as linear equations and inequalities. These papers are particularly applied to chemical process synthesis.

In Raman and Grossmann (1991), the authors illustrated how logic relations and heuristics expressed in the form of propositional logic can be represented in terms of linear inequalities involving 0-1 variables. Based on this representation, Raman and Grossmann (1992) proposed the integration of logic and heuristic knowledge using a quantitative framework. The authors considered the addition of constraints for the logic relations among units in the MI(N)LP to decrease the relaxation gap[1]. Constraints for heuristic rules and logic relations were used to improve the search in a master MI(N)LP approach. The computational results showed that the addition of logic constraints in MILP models often produces significant computational time reductions.

In Raman and Grossmann (1993), the authors studied the symbolic integration of logic in mixed integer linear programming techniques. Once more, the application was focused on chemical process synthesis. The main objective was to reduce the number of enumerated nodes by using the logic to decide on the branching of variables, and to determine, by symbolic inference, whether additional variables should be fixed at each node. Therefore, the authors proposed a branch-and-bound method that performs logical inferences at each node.

[1]The relaxation gap is defined in section 2.2.

In Raman and Grossmann (1994), a solution algorithm that generalizes the method presented in Raman and Grossmann (1993) is proposed. Thus, it is possible to notice that the papers of Raman and Grossman focused on not only the formulation of propositional logic statements as linear inequalities, but also on how to use logical inference in search procedures.

Last but not least, there is always useful information about model building techniques for MILP in the classical book of Williams (1999).

4.2 Reformulation of Logical Relations

The literature previously cited in section 4.1 indicates that connections between logic and integer programming can be fruitfully used to the construction of mathematical programming models. Therefore, this connection is herewith further investigated. Section 4.2 explains modeling techniques that help transform some logical relations with special, for instance nonlinear, features into conventional mixed integer linear programming models. In particular, in section 4.2.3, this thesis expands some modeling features presented in Hadjiconstantinou and Mitra (1994), Mitra et al. (1994), and Hürlimann (1998).

4.2.1 Basic Logic Concepts

A basic concept used in propositional logic is the *statement.* A statement defines a declarative sentence, which is either true or false. A statement is also called an *atomic proposition.* A proposition can take one of the truth values true (abbreviated by T) or false (abbreviated by F). As no other value is permitted, the calculus of propositions is referred to as a two-valued logic. Propositional calculus enables compound propositions to be formed by connecting simple statements with logical connectives. The primary logical connectives are negation ($\neg p$), conjunction ($p \wedge q$), and disjunction ($p \vee q$), which are understood to represent the semantics of, respectively, not, and, or. However, an additional set of logical connectives, which can be derived from the primary ones (see table 4.2), are also important: implication

($p \rightarrow q$: if p then q), equivalence ($p \leftrightarrow q$: p if and only if q), exclusive disjunction ($p \otimes q$: p xor q), joint denial ($\neg(p \vee q)$: p nor q), and non-conjunction ($\neg(p \wedge q)$: p nand q). Table 4.1 presents the truth values for the main logical connectives[2].

Table 4.1: Truth Table of Logical Connectives.

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$p \otimes q$	$\neg(p \vee q)$	$\neg(p \wedge q)$
0	0	1	0	0	1	1	0	1	1
0	1	1	0	1	1	0	1	0	1
1	0	0	0	1	0	0	1	0	1
1	1	0	1	1	1	1	0	0	0

Two expressions are said to be "equivalent" if and only if their truth values are the same, and this is expressed as $p \equiv q$ (p is equivalent to q). The well-known De Morgan's laws are examples of logical equivalences:

(i) First law: $\neg(p \vee q) \equiv \neg p \wedge \neg q$;

(ii) Second law: $\neg(p \wedge q) \equiv \neg p \vee \neg q$.

By De Morgan's laws, conjunction can be expressed in terms of negation and disjunction, and disjunctions can be expressed in terms of negation and conjunctions. A general compound proposition in the form $R_1 \wedge R_2 \wedge \ldots \wedge R_n$, in which every R_i, $i = 1, 2, \ldots, n$, is a disjunction of atomic propositions is called Conjunctive Normal Form (CNF). A general compound proposition in the form $S_1 \vee S_2 \vee \ldots \vee S_n$, in which every S_i, $i = 1, 2, \ldots, n$, is a conjunction of atomic propositions is called Disjunctive Normal Form (DNF). Table 4.2, which is adapted from Hadjiconstantinou and Mitra (1994), indicates the transformation of logical statements into equivalent forms.

[2]The numerical values one (1) and zero (0) are used to represented, respectively, true (T) and false (F) atomic propositions.

Table 4.2: Transformation of Logical Statements into Equivalent Forms.

Statement	Equivalent Form	Observations
$\neg\neg p$	p	double negation
$p \otimes q$	$(\neg p \wedge q) \vee (p \wedge \neg q)$	exclusive OR (either p or q)
$\neg(p \vee q)$	$\neg p \wedge \neg q$	De Morgan's law
$\neg(p \wedge q)$	$\neg p \vee \neg q$	De Morgan's law
$p \wedge (q \vee r)$	$(p \wedge q) \vee (p \wedge r)$	distributive law
$p \vee (q \wedge r)$	$(p \vee q) \wedge (p \vee r)$	distributive law
$p \rightarrow q$	$\neg p \vee q$	implication (if p then q)
$p \rightarrow q \wedge r$	$(p \rightarrow q) \wedge (p \rightarrow r)$	implication
$p \rightarrow q \vee r$	$(p \rightarrow q) \vee (p \rightarrow r)$	implication
$p \wedge q \rightarrow r$	$(p \rightarrow r) \vee (q \rightarrow r)$	implication
$p \vee q \rightarrow r$	$(p \rightarrow r) \wedge (q \rightarrow r)$	implication
$p \leftrightarrow q$	$(p \rightarrow q) \wedge (q \rightarrow p)$	equivalence (p if and only if q)
$p \leftrightarrow q$	$(p \rightarrow q) \wedge (\neg p \rightarrow \neg q)$	equivalence

4.2.2 Connecting Logical Variables

The basic logic concepts previously described in section 4.2.1 are, in fact, a background knowledge for the main objective of this chapter: the expression of logical relations into a system of mixed integer linear programming constraints. This problem was already addressed, with different emphases (see section 4.1), by Raman and Grossmann (1991), Hadjiconstantinou and Mitra (1994), Mitra et al. (1994), Floudas (1995), Yeom and Lee (1998), and Hürlimann (1998). In order to explain the underlying principles of reformulating logical relations into MILP constraints, this thesis follows the same train of reasoning of Hadjiconstantinou and Mitra (1994), in which the authors distinguished two main topics, namely:

(i) Connecting logical variables (explained in this section (section 4.2.2)); and,

(ii) Logically related linear form constraints (explained and expanded in section 4.2.3).

Initially, let's consider that p_k denotes the k^{th} logical statement, which takes values true (T) or false (F), and represents an atomic proposition describing an action, option, or decision. Furthermore, each type of action (or option) has associated an integer variable. This variable, known as the binary decision variable, is denoted by δ_k, and can only take the values 0 and 1. The connection of δ_k with the propositions is defined by the following relations: $\delta_k = 1$ if and only if proposition p_k is true, and $\delta_k = 0$ if and only if proposition p_k is false. Logical conditions linking the different actions in a model are achieved by expressing these conditions as linear constraints connecting the associated decision variables. For example, the statement $p_1 \vee \neg p_2$ can be associated with the IP formulation $\delta_1 + (1 - \delta_2) \geq 1$.

Table 4.3, which is adapted from Hadjiconstantinou and Mitra (1994), presents a list of standard form "variable transformations". These transformations can be applied to compound statements involving one or more atomic propositions p_k, whereby the compound statements are restated in linear algebraic forms involving decision variables δ_k. The expressions involving p_k's and δ_k's are logically equivalent.

Just in order to illustrate the use of tables 4.1 to 4.3, the logical expression $(p_1 \vee p_2 \vee p_3) \wedge (\neg(p_1 \wedge \neg p_3))$ is herein translated to an equivalent linear constraint form. By table 4.2, $\neg(p \wedge q) \equiv (\neg p \vee \neg q)$, and $\neg\neg p \equiv p$. Therefore, the initial expression can be rewritten as $(p_1 \vee p_2 \vee p_3) \wedge (\neg p_1 \vee p_3)$[3]. By table 4.1, $\neg p \equiv (1 - p)$. Thus, by tables 4.1 and 4.3, $(p_1 \vee p_2 \vee p_3) \equiv (\delta_1 + \delta_2 + \delta_3 \geq 1)$, and $(\neg p_1 \vee p_3) \equiv ((1 - \delta_1) + \delta_3 \geq 1)$, therefore, $(\neg p_1 \vee p_3) \equiv (\delta_1 - \delta_3 \leq 0)$. Hence, the initial logical expression can be rewritten as indicated in equivalence 4.1.

$$(p_1 \vee p_2 \vee p_3) \wedge (\neg(p_1 \wedge \neg p_3)) \equiv \begin{cases} \delta_1 + \delta_2 + \delta_3 \geq 1 \\ \delta_1 - \delta_3 \leq 0 \end{cases} \tag{4.1}$$

Equivalence 4.1 is just a small example of a compound logical proposition that is translated into a set of constraints involving indicator variables. However, a huge set of different compound logical expressions can be specified, thus, it is necessary to establish a systematic procedure to translate such expressions into conventional

[3]At this point, the original expression is in the form $R_1 \wedge R_2$, where R_1 and R_2 are disjunctions of atomic propositions: $R_1 = p_1 \vee p_2 \vee p_3$; $R_2 = \neg p_1 \vee p_3$.

Table 4.3: Variable Transformations: Logic Statement versus Linear Constraint.

Statement	Linear Form Constraint
$\neg p_1$	$\delta_1 = 0$
$p_1 \vee p_2$	$\delta_1 + \delta_2 \geq 1$
$p_1 \otimes p_2$	$\delta_1 + \delta_2 = 1$
$p_1 \wedge p_2$	$\delta_1 = 1, \delta_2 = 1$
$\neg(p_1 \vee p_2)$	$\delta_1 = 0, \delta_2 = 0$
$\neg(p_1 \wedge p_2)$	$\delta_1 + \delta_2 \leq 1$
$p_1 \rightarrow \neg p_2$	$\delta_1 + \delta_2 \leq 1$
$p_1 \rightarrow p_2$	$\delta_1 - \delta_2 \leq 0$
$p_1 \leftrightarrow p_2$	$\delta_1 - \delta_2 = 0$
$p_1 \rightarrow p_2 \wedge p_3$	$\delta_1 \leq \delta_2, \delta_1 \leq \delta_3$
$p_1 \rightarrow p_2 \vee p_3$	$\delta_1 \leq \delta_2 + \delta_3$
$p_1 \wedge p_2 \rightarrow p_3$	$\delta_1 + \delta_2 - \delta_3 \leq 1$
$p_1 \vee p_2 \rightarrow p_3$	$\delta_1 \leq \delta_3, \delta_2 \leq \delta_3$
$p_1 \wedge (p_2 \vee p_3)$	$\delta_1 = 1, \delta_2 + \delta_3 \geq 1$
$p_1 \vee (p_2 \wedge p_3)$	$\delta_1 + \delta_2 \geq 1, \delta_1 + \delta_3 \geq 1$
$p_1 \vee p_2 \vee \ldots \vee p_n$	$\delta_1 + \delta_2 + \ldots + \delta_n \geq 1$
$p_1 \otimes p_2 \otimes \ldots \otimes p_n$	$\delta_1 + \delta_2 + \ldots + \delta_n = 1$
$p_1 \wedge \ldots \wedge p_k \rightarrow p_{k+1} \vee \ldots \vee p_n$	$(1 - \delta_1) + \ldots + \delta_{k+1} + \ldots + \delta_n \geq 1$
At least k out of n are true	$\delta_1 + \delta_2 + \ldots + \delta_n \geq k$
Exactly k out of n are true	$\delta_1 + \delta_2 + \ldots + \delta_n = k$
At most k out of n are true	$\delta_1 + \delta_2 + \ldots + \delta_n \leq k$
$p_n \equiv p_1 \vee p_2 \vee \ldots \vee p_k$	$\delta_1 + \delta_2 + \ldots + \delta_k \geq \delta_n$
	$(\delta_n \geq \delta_j \quad j = 1, 2, \ldots, k)$
$p_n \equiv p_1 \wedge p_2 \wedge \ldots \wedge p_k$	$-\delta_1 - \delta_2 - \ldots - \delta_k + \delta_n \geq 1 - k$
	$(\delta_n \leq \delta_j \quad j = 1, 2, \ldots, k)$

linear form constraints. In general, the approach used in such translation involves a reduction to conjunctive/disjunctive normal forms (CNF/DNF).

According to Chandru and Hooker (1999), the two normal forms (CNF and DNF) are dual representations with symmetric properties. Although there are some applications of propositional logic for which the DNF may be the more accepted normal form, the CNF is the most used. The traditional technique of transforming a given formula of statements connected by $\wedge$ and $\vee$ connectives into an equivalent CNF is show in procedure 1.

Procedure 1: Reduction to CNF (Chandru and Hooker, 1999)

Step 1: Use the transformation rules of De Morgan's law, and the double negation (table 4.2) to absorb all "$\neg$" into the atomic statements;

Step 2: Use the distributive law (table 4.2) to move the conjunctions out of the statements, until each statement is a clause of pure disjunctions.

The study of this translation procedure is beyond the thesis' scope, and the interested reader is referred to Chandru and Hooker (1999). In particular, Chandru and Hooker (1999) proved that procedure 1 is, in the worst case, an exponential-time algorithm.

4.2.3 Logically Related Linear Form Constraints

Section 4.2.2 initially considered that a binary indicator variable denoted by δ_k represents the truth or falsehood of an atomic proposition p_k. In sequence, section 4.2.2 established the equivalence between compound logic statements and linear algebraic constraints involving binary indicator variables (δ_k). However section 4.2.2 does not explicitly indicate the underlying association of mixed integer linear programming constraints with indicator variables δ_k. In fact, much of the literature content cited in section 4.1 is just aimed at the connection between binary variables and compound logical propositions.

In order to express the association of binary variables and MILP constraints, let's initially consider the general statement of a linear mathematical programming model, defined in formulation 4.2. In 4.2, the c_j's are referred to as cost coefficients, the a_{kj}'s are referred to as constraint coefficients on variables x_j's, the b_k's are referred to as requirements[4], J is the set of variables, and K is the set of constraints. The symbol ρ denotes mathematical relations such as $\leq$, $\geq$, $<$, $>$, $=$, and $\neq$. The term $\sum_{j\in J} a_{kj}x_j\{\rho\}b_k$, $\forall k\in K$ is the Linear Form Constraint k (LFC_k). A maximization model can be written as a minimization model by multiplying the objective by (-1) and minimizing it.

$$\begin{aligned} min \quad & \sum_{j\in J} c_j x_j \\ s.t. \quad & \sum_{j\in J} a_{kj}x_j\{\rho\}b_k \quad \forall k\in K \\ & x_j \geq 0 \quad \forall j\in J \end{aligned} \tag{4.2}$$

Considering now the existence of finite upper (U_k) and lower (L_k) bounds on LFC_k, as indicated in 4.3.

$$L_k \leq \sum_{j\in J} a_{kj}x_j - b_k \leq U_k \quad \forall k\in K \tag{4.3}$$

The bounds on inequality 4.3 may be given or, alternatively, can be computed for finite ranges of x_j, as demonstrated by Brearley et al. (1975). For example, if $l_j \leq x_j \leq u_j$, $j\in J$, then the lower and upper bounds on the linear form constraint can be determined as indicated in formulation 4.4.

$$\begin{aligned} L_k &= \sum_{j\in P_k} a_{kj}l_j + \sum_{j\in N_k} a_{kj}u_j - b_k \quad \forall k\in K \\ U_k &= \sum_{j\in P_k} a_{kj}u_j + \sum_{j\in N_k} a_{kj}l_j - b_k \quad \forall k\in K \\ P_k &= \{j : a_{kj} > 0\} \quad N_k = \{j : a_{kj} < 0\} \end{aligned} \tag{4.4}$$

4.2.3.1 Logical Constraint in the Implication Form

A "logical constraint in the implication form" (LCIF) is a logical combination of simple constraints, and is defined as: *if antecedent, then consequent.* In this case,

[4]It is customary to gather all the constants to the right-hand side of an LP constraint.

the antecedent is a logical variable and the consequent is a linear form constraint. In order to model the LCIF, a 0-1 indicator variable (binary variable) is linked to the antecedent. Whether the LFC_k applies ($\delta_k = 1$) or otherwise ($\delta_k = 0$) is indicated by a binary variable δ_k.

Propositions 4.2.1 to 4.2.6 indicate the mixed integer linear programming expression of logical constraints in the implication form. These propositions were developed considering "ρ" of LFC_k as $\leq$, $\geq$, $<$, $>$, $=$, and $\neq$. For notation simplicity, the quantifier $\forall k \in K$ is omitted and the term $\sum_{j\in J} a_{kj}x_j$ is written as $\sum_j a_{kj}x_j$.

Proposition 4.2.1.

$$\underbrace{\left(\delta_k = 1 \rightarrow \sum_j a_{kj}x_j - b_k \leq 0\right)}_{\textit{if antecedent then consequent}} \quad \equiv \quad \underbrace{\sum_j a_{kj}x_j - b_k \leq U_k(1-\delta_k)}_{\textit{MILP expression}}$$

Proof. In proposition 4.2.1, δ_k is a variable that can assume either values 0 or 1. Then, testing both cases in the MILP expression yields:

$$\delta_k = 0 \rightarrow \sum_j a_{kj}x_j - b_k \leq U_k \therefore \text{tautology}$$

$$\delta_k = 1 \rightarrow \sum_j a_{kj}x_j - b_k \leq 0 \therefore \text{proposition holds}$$

Therefore, setting $\delta_k = 0$ imposes the relation $\sum_j a_{kj}x_j - b_k \leq U_k$, which is a tautology (see inequality 4.3). Setting $\delta_k = 1$ makes the constraint $\sum_j a_{kj}x_j - b_k \leq 0$ valid. Thus, proposition 4.2.1 holds. □

Remark 4.2.1. $L_k \leq 0$. As it is stated in formulation 4.4, L_k as well as U_k can be determined prior to applying the "Simplex" algorithm. In general, commercial LP codes bring strong pre-processing algorithms, and L_k and U_k can be effectively determined. Therefore, L_k and U_k are parameters to the MILP formulation. However, in case $L_k > 0$, then $\sum_j a_{kj}x_j - b_k$ must be greater than zero (see inequality 4.3), and the indicator variable δ_k cannot be set to one in any feasible circumstance. Thus, there is no sense in applying proposition 4.2.1 in case $L_k > 0$. This is a "natural constraint" to proposition 4.2.1.

Propositions 4.2.2 to 4.2.6 follow the same train of reasoning of proposition 4.2.1. In propositions 4.2.3, 4.2.4, and 4.2.6, ε is a positive small tolerance value ($\varepsilon > 0$), below which the linear constraint $\sum_j a_{kj}x_j\{\rho\}b_k$, $\rho \in \{<, >\}$ is regarded as having been broken[5]. In proposition 4.2.6, $\delta_k^{'}$ and $\delta_k^{''}$ are auxiliary binary variables.

Proposition 4.2.2.

$$\left(\delta_k = 1 \rightarrow \sum_j a_{kj}x_j - b_k \geq 0\right) \quad \equiv \quad \sum_j a_{kj}x_j - b_k \geq L_k(1 - \delta_k)$$

Proof.

$$\delta_k = 0 \rightarrow \sum_j a_{kj}x_j - b_k \geq L_k \therefore \text{tautology}$$
$$\delta_k = 1 \rightarrow \sum_j a_{kj}x_j - b_k \geq 0 \therefore \text{proposition holds}$$

□

Remark 4.2.2. $U_k \geq 0$. There is no sense in applying proposition 4.2.2 in case $U_k < 0$ because δ_k cannot be set to one. Thus, the "natural constraint" is that $U_k \geq 0$.

Proposition 4.2.3.

$$\left(\delta_k = 1 \rightarrow \sum_j a_{kj}x_j - b_k < 0\right) \quad \equiv \quad \sum_j a_{kj}x_j - b_k \leq (U_k + \varepsilon)(1 - \delta_k) - \varepsilon$$

Proof.

$$\delta_k = 0 \rightarrow \sum_j a_{kj}x_j - b_k \leq U_k \therefore \text{tautology}$$
$$\delta_k = 1 \rightarrow \sum_j a_{kj}x_j - b_k \leq -\varepsilon \therefore \sum_j a_{kj}x_j - b_k < 0 \therefore \text{proposition holds}$$

□

Remark 4.2.3. $L_k \leq -\varepsilon$. There is no sense in applying proposition 4.2.3 in case $L_k \geq 0$ because δ_k cannot be set to one. Thus, the "natural constraint" is that

[5]The determination of a suitable value for ε is dependent upon the accuracy level of each particular software/hardware. For instance, the set of simulations conducted in this thesis (chapters 5 and 6) used $\varepsilon = 10^{-3}$.

$L_k < 0$. However, strict inequalities are not directly specified in LP, and a small tolerance value ε is used in the MILP expression. Therefore, in order to validate such MILP expression, L_k must be lower than or equal $-\varepsilon$ and not just lower than zero.

Proposition 4.2.4.

$$\left(\delta_k = 1 \to \sum_j a_{kj}x_j - b_k > 0\right) \quad \equiv \quad \sum_j a_{kj}x_j - b_k \geq (L_k - \varepsilon)(1-\delta_k) + \varepsilon$$

Proof.

$$\delta_k = 0 \to \sum_j a_{kj}x_j - b_k \geq L_k \therefore \text{tautology}$$
$$\delta_k = 1 \to \sum_j a_{kj}x_j - b_k \geq +\varepsilon \therefore \sum_j a_{kj}x_j - b_k > 0 \therefore \text{proposition holds}$$

□

Remark 4.2.4. $U_k \geq +\varepsilon$.

Proposition 4.2.5.

$$\underbrace{\left(\delta_k = 1 \to \sum_j a_{kj}x_j - b_k = 0\right)}_{\textit{if antecedent then consequent}} \quad \equiv \quad \underbrace{\left\{\begin{array}{l} \sum_j a_{kj}x_j - b_k \leq U_k(1-\delta_k) \\ \sum_j a_{kj}x_j - b_k \geq L_k(1-\delta_k) \end{array}\right.}_{\textit{set of MILP expressions}}$$

Proof. In proposition 4.2.5, δ_k is a variable that can assume either values 0 or 1. Then, testing both cases in the set of MILP expressions yields:

$$\delta_k = 0 \to \left\{\begin{array}{l} \sum_j a_{kj}x_j - b_k \leq U_k \therefore \text{tautology} \\ \sum_j a_{kj}x_j - b_k \geq L_k \therefore \text{tautology} \end{array}\right.$$
$$\delta_k = 1 \to \left\{\begin{array}{l} \sum_j a_{kj}x_j - b_k \leq 0 \\ \sum_j a_{kj}x_j - b_k \geq 0 \end{array}\right\} \therefore \sum_j a_{kj}x_j - b_k = 0 \therefore \text{proposition holds}$$

□

Remark 4.2.5. $(L_k \leq 0) \wedge (U_k \geq 0)$.

Proposition 4.2.6.

$$\underbrace{\left(\delta_k = 1 \rightarrow \sum_j a_{kj}x_j - b_k \neq 0\right)}_{\textit{if antecedent then consequent}} \quad \equiv \quad \underbrace{\left\{\begin{array}{l} \sum_j a_{kj}x_j - b_k \leq (U_k + \varepsilon)(1 - \delta_k^{'}) - \varepsilon \\ \sum_j a_{kj}x_j - b_k \geq (L_k - \varepsilon)(1 - \delta_k^{''}) + \varepsilon \\ \delta_k = \delta_k^{'} + \delta_k^{''} \end{array}\right.}_{\textit{set of MILP expressions}}$$

Proof. In proposition 4.2.6, δ_k is a variable that can assume either values 0 or 1. With $\delta_k = 0$ in the set of MILP expressions, then:

$$\delta_k = 0 \therefore \delta_k^{'} + \delta_k^{''} = 0 \rightarrow (\delta_k^{'} = 0) \wedge (\delta_k^{''} = 0) \therefore$$

$$\therefore \quad (\delta_k^{'} = 0) \wedge (\delta_k^{''} = 0) \rightarrow \left\{\begin{array}{l} \sum_j a_{kj}x_j - b_k \leq U_k \therefore \text{tautology} \\ \sum_j a_{kj}x_j - b_k \geq L_k \therefore \text{tautology} \end{array}\right.$$

With $\delta_k = 1$ in the set of MILP expressions of proposition 4.2.6, then:

$$\delta_k = 1 \therefore \delta_k^{'} + \delta_k^{''} = 1 \rightarrow \underbrace{[(\delta_k^{'} = 1) \wedge (\delta_k^{''} = 0)]}_{\text{Case (i)}} \vee \underbrace{[(\delta_k^{'} = 0) \wedge (\delta_k^{''} = 1)]}_{\text{Case (ii)}} \therefore$$

Case (i):

$$(\delta_k^{'} = 1) \wedge (\delta_k^{''} = 0) \rightarrow \left\{\begin{array}{l} \sum_j a_{kj}x_j - b_k \leq -\varepsilon \therefore \sum_j a_{kj}x_j - b_k < 0 \\ \sum_j a_{kj}x_j - b_k \geq L_k \therefore \text{tautology} \end{array}\right\} \quad \therefore$$

$$\therefore \quad (\delta_k^{'} = 1) \wedge (\delta_k^{''} = 0) \rightarrow \sum_j a_{kj}x_j - b_k < 0$$

Case (ii):

$$(\delta_k^{'} = 0) \wedge (\delta_k^{''} = 1) \rightarrow \left\{\begin{array}{l} \sum_j a_{kj}x_j - b_k \leq U_k \therefore \text{tautology} \\ \sum_j a_{kj}x_j - b_k \geq +\varepsilon \therefore \sum_j a_{kj}x_j - b_k > 0 \end{array}\right\} \quad \therefore$$

$$\therefore \quad (\delta_k^{'} = 0) \wedge (\delta_k^{''} = 1) \rightarrow \sum_j a_{kj}x_j - b_k > 0$$

By Case (i) and Case (ii), therefore:

$$\delta_k = 1 \rightarrow [\sum_j a_{kj}x_j - b_k < 0] \vee [\sum_j a_{kj}x_j - b_k > 0] \therefore$$

$$\therefore \quad \delta_k = 1 \rightarrow \sum_j a_{kj}x_j - b_k \neq 0 \therefore \text{proposition 4.2.6 holds}$$

□

Remark 4.2.6. $(L_k \leq -\varepsilon) \wedge (U_k \geq +\varepsilon)$.

Table 4.4 summarizes the logical constraints in the implication form (LCIF) demonstrated in propositions 4.2.1 to 4.2.6. In these propositions, "ρ" of LFC_k was considered as $\leq$, $\geq$, $<$, $>$, $=$, and $\neq$. Hadjiconstantinou and Mitra (1994) and Mitra et al. (1994) also show the LCIF for ρ as $\leq$, $\geq$, and $=$. Amongst the section 4.1 literature, the papers have also limited the scope of LCIF as in Hadjiconstantinou and Mitra (1994) and Mitra et al. (1994). Therefore, this thesis expands this specific feature of Hadjiconstantinou and Mitra (1994) and Mitra et al. (1994).

Table 4.4: Logical Constraints in the Implication Form.

LCIF ($\forall k \in K$)	Equivalent Set of MILP Expressions ($\forall k \in K$)	Remark ($\forall k \in K$)
$\delta_k = 1 \rightarrow \sum_j a_{kj}x_j \leq b_k$	$\sum_j a_{kj}x_j - b_k \leq U_k(1-\delta_k)$	$L_k \leq 0$
$\delta_k = 1 \rightarrow \sum_j a_{kj}x_j \geq b_k$	$\sum_j a_{kj}x_j - b_k \geq L_k(1-\delta_k)$	$U_k \geq 0$
$\delta_k = 1 \rightarrow \sum_j a_{kj}x_j < b_k$	$\sum_j a_{kj}x_j - b_k \leq (U_k+\varepsilon)(1-\delta_k) - \varepsilon$	$L_k \leq -\varepsilon$
$\delta_k = 1 \rightarrow \sum_j a_{kj}x_j > b_k$	$\sum_j a_{kj}x_j - b_k \geq (L_k-\varepsilon)(1-\delta_k) + \varepsilon$	$U_k \geq +\varepsilon$
$\delta_k = 1 \rightarrow \sum_j a_{kj}x_j = b_k$	$\begin{cases} \sum_j a_{kj}x_j - b_k \leq U_k(1-\delta_k) \\ \sum_j a_{kj}x_j - b_k \geq L_k(1-\delta_k) \end{cases}$	$\begin{cases} L_k \leq 0 \\ U_k \geq 0 \end{cases}$
$\delta_k = 1 \rightarrow \sum_j a_{kj}x_j \neq b_k$	$\begin{cases} \sum_j a_{kj}x_j - b_k \leq (U_k+\varepsilon)(1-\delta_k^{'}) - \varepsilon \\ \sum_j a_{kj}x_j - b_k \geq (L_k-\varepsilon)(1-\delta_k^{''}) + \varepsilon \\ \delta_k = \delta_k^{'} + \delta_k^{''} \end{cases}$	$\begin{cases} L_k \leq -\varepsilon \\ U_k \geq +\varepsilon \end{cases}$

4.2.3.2 Logical Constraint in the Equivalence Form

A "logical constraint in the equivalence form" (LCEF) is a logical combination of simple constraints, and is defined as: *if and only if antecedent, then consequent.* In this case, the antecedent is a logical variable and the consequent is a linear form constraint. In order to model the LCEF, a 0-1 indicator variable (binary variable) is linked to the antecedent. Whether the LFC_k applies ($\delta_k = 1$) or otherwise ($\delta_k = 0$) is indicated by the binary variable δ_k.

Propositions 4.2.7 to 4.2.12 indicate the mixed integer linear programming expression of logical constraints in the equivalence form. These propositions were developed considering "ρ" of LFC_k as $\leq$, $\geq$, $<$, $>$, $=$, and $\neq$. The ε is a positive small tolerance value ($\varepsilon > 0$), below which the linear constraint $\sum_j a_{kj}x_j\{\rho\}b_k$, $\rho \in \{<,>\}$ is regarded as having been broken. For notation simplicity, the quantifier $\forall k \in K$ is omitted and the term $\sum_{j\in J} a_{kj}x_j$ is written as $\sum_j a_{kj}x_j$.

Proposition 4.2.7.

$$\underbrace{\left(\delta_k = 1 \leftrightarrow \sum_j a_{kj}x_j - b_k \leq 0\right)}_{\textit{if and only if antecedent then consequent}} \equiv \underbrace{\begin{cases} \sum_j a_{kj}x_j - b_k \leq U_k(1-\delta_k) \\ \sum_j a_{kj}x_j - b_k \geq (L_k - \varepsilon)\delta_k + \varepsilon \end{cases}}_{\textit{set of MILP expressions}}$$

Proof. From table 4.2, an equivalence can be generically expressed as:

$$p \leftrightarrow q \equiv \underbrace{(p \rightarrow q)}_{(i_1)} \wedge \underbrace{(\neg p \rightarrow \neg q)}_{(i_2)}$$

Hence, if implications (i_1) and (i_2) hold, then the equivalence holds. In the particular equivalence presented in proposition 4.2.7, implications (i_1) and (i_2) can be written as:

$$\text{Implication } (i_1): \quad \delta_k = 1 \rightarrow \sum_j a_{kj}x_j - b_k \leq 0$$

$$\text{Implication } (i_2): \quad \delta_k = 0 \rightarrow \sum_j a_{kj}x_j - b_k \nleq 0 \therefore \sum_j a_{kj}x_j - b_k > 0$$

In proposition 4.2.7, δ_k can assume either values zero or one. Testing both cases in the set of MILP expressions yields:

$$\delta_k = 0 \rightarrow \left\{ \begin{array}{l} \sum_j a_{kj}x_j - b_k \leq U_k \therefore \text{tautology} \\ \sum_j a_{kj}x_j - b_k \geq \varepsilon \therefore \sum_j a_{kj}x_j - b_k > 0 \end{array} \right\} \therefore \text{implication } (i_2) \text{ holds}$$

$$\delta_k = 1 \rightarrow \left\{ \begin{array}{l} \sum_j a_{kj}x_j - b_k \leq 0 \\ \sum_j a_{kj}x_j - b_k \geq L_k \therefore \text{tautology} \end{array} \right\} \therefore \text{implication } (i_1) \text{ holds}$$

Implications (i_1) and (i_2) hold, therefore proposition 4.2.7 holds. □

Remark 4.2.7. $(L_k \leq 0) \wedge (U_k \geq \varepsilon)$.

Propositions 4.2.8 to 4.2.12 follow the same train of reasoning of proposition 4.2.7, and their explanation is somehow shortened. In propositions 4.2.11 and 4.2.12, $\delta_k^{'}$ and $\delta_k^{''}$ are auxiliary binary variables.

Proposition 4.2.8.

$$\left(\delta_k = 1 \leftrightarrow \sum_j a_{kj}x_j - b_k \geq 0\right) \quad \equiv \quad \left\{ \begin{array}{l} \sum_j a_{kj}x_j - b_k \geq L_k(1-\delta_k) \\ \sum_j a_{kj}x_j - b_k \leq (U_k+\varepsilon)\delta_k - \varepsilon \end{array} \right.$$

Proof. From table 4.2, the equivalence can be expressed as:

$$p \leftrightarrow q \equiv \underbrace{(p \rightarrow q)}_{(i_1)} \wedge \underbrace{(\neg p \rightarrow \neg q)}_{(i_2)}$$

Therefore, implications (i_1) and (i_2) must hold:

$$\text{Implication } (i_1): \quad \delta_k = 1 \rightarrow \sum_j a_{kj}x_j - b_k \geq 0$$

$$\text{Implication } (i_2): \quad \delta_k = 0 \rightarrow \sum_j a_{kj}x_j - b_k \ngeq 0 \therefore \sum_j a_{kj}x_j - b_k < 0$$

In proposition 4.2.8, δ_k can assume either values zero or one:

$$\delta_k = 0 \rightarrow \left\{ \begin{array}{l} \sum_j a_{kj}x_j - b_k \geq L_k \therefore \text{tautology} \\ \sum_j a_{kj}x_j - b_k \leq -\varepsilon \therefore \sum_j a_{kj}x_j - b_k < 0 \end{array} \right\} \therefore \text{implication } (i_2) \text{ holds}$$

$$\delta_k = 1 \rightarrow \left\{ \begin{array}{l} \sum_j a_{kj}x_j - b_k \geq 0 \\ \sum_j a_{kj}x_j - b_k \leq U_k \therefore \text{tautology} \end{array} \right\} \therefore \text{implication } (i_1) \text{ holds}$$

Implications (i_1) and (i_2) hold, therefore proposition 4.2.8 holds. □

Remark 4.2.8. $(L_k \leq -\varepsilon) \wedge (U_k \geq 0)$.

Proposition 4.2.9.

$$\left(\delta_k = 1 \leftrightarrow \sum_j a_{kj}x_j - b_k < 0\right) \quad \equiv \quad \begin{cases} \sum_j a_{kj}x_j - b_k \leq (U_k + \varepsilon)(1 - \delta_k) - \varepsilon \\ \sum_j a_{kj}x_j - b_k \geq L_k\delta_k \end{cases}$$

Proof. From table 4.2, the equivalence can be expressed as:

$$p \leftrightarrow q \equiv \underbrace{(p \rightarrow q)}_{(i_1)} \wedge \underbrace{(\neg p \rightarrow \neg q)}_{(i_2)}$$

Therefore, implications (i_1) and (i_2) must hold:

$$\begin{aligned} &\text{Implication } (i_1): \quad \delta_k = 1 \rightarrow \sum_j a_{kj}x_j - b_k < 0 \\ &\text{Implication } (i_2): \quad \delta_k = 0 \rightarrow \sum_j a_{kj}x_j - b_k \nless 0 \therefore \sum_j a_{kj}x_j - b_k \geq 0 \end{aligned}$$

In proposition 4.2.9, δ_k can assume either values zero or one:

$$\delta_k = 0 \rightarrow \left\{ \begin{array}{l} \sum_j a_{kj}x_j - b_k \leq U_k \therefore \text{tautology} \\ \sum_j a_{kj}x_j - b_k \geq 0 \end{array} \right\} \therefore \text{implication } (i_2) \text{ holds}$$

$$\delta_k = 1 \rightarrow \left\{ \begin{array}{l} \sum_j a_{kj}x_j - b_k \leq -\varepsilon \therefore \sum_j a_{kj}x_j - b_k < 0 \\ \sum_j a_{kj}x_j - b_k \geq L_k \therefore \text{tautology} \end{array} \right\} \therefore \text{implication } (i_1) \text{ holds}$$

Implications (i_1) and (i_2) hold, therefore proposition 4.2.9 holds. □

Remark 4.2.9. $(L_k \leq -\varepsilon) \wedge (U_k \geq 0)$.

Proposition 4.2.10.

$$\left(\delta_k = 1 \leftrightarrow \sum_j a_{kj}x_j - b_k > 0\right) \quad \equiv \quad \begin{cases} \sum_j a_{kj}x_j - b_k \geq (L_k - \varepsilon)(1 - \delta_k) + \varepsilon \\ \sum_j a_{kj}x_j - b_k \leq U_k\delta_k \end{cases}$$

Proof. From table 4.2, the equivalence can be expressed as:

$$p \leftrightarrow q \equiv \underbrace{(p \rightarrow q)}_{(i_1)} \wedge \underbrace{(\neg p \rightarrow \neg q)}_{(i_2)}$$

Therefore, implications (i_1) and (i_2) must hold:

$$\text{Implication } (i_1): \quad \delta_k = 1 \rightarrow \sum_j a_{kj}x_j - b_k > 0$$

$$\text{Implication } (i_2): \quad \delta_k = 0 \rightarrow \sum_j a_{kj}x_j - b_k \not> 0 \therefore \sum_j a_{kj}x_j - b_k \leq 0$$

In proposition 4.2.10, δ_k can assume either values zero or one:

$$\delta_k = 0 \rightarrow \left\{ \begin{array}{l} \sum_j a_{kj}x_j - b_k \geq L_k \therefore \text{tautology} \\ \sum_j a_{kj}x_j - b_k \leq 0 \end{array} \right\} \therefore \text{implication } (i_2) \text{ holds}$$

$$\delta_k = 1 \rightarrow \left\{ \begin{array}{l} \sum_j a_{kj}x_j - b_k \geq +\varepsilon \therefore \sum_j a_{kj}x_j - b_k > 0 \\ \sum_j a_{kj}x_j - b_k \leq U_k \therefore \text{tautology} \end{array} \right\} \therefore \text{implication } (i_1) \text{ holds}$$

Implications (i_1) and (i_2) hold, therefore proposition 4.2.10 holds. □

Remark 4.2.10. $(L_k \leq 0) \wedge (U_k \geq +\varepsilon)$.

Proposition 4.2.11.

$$\left(\delta_k = 1 \leftrightarrow \sum_j a_{kj}x_j - b_k = 0\right) \quad \equiv \quad \left\{ \begin{array}{l} \sum_j a_{kj}x_j - b_k \leq U_k(1 - \delta_k^{'}) \\ \sum_j a_{kj}x_j - b_k \geq (L_k - \varepsilon)\delta_k^{'} + \varepsilon \\ \sum_j a_{kj}x_j - b_k \geq L_k(1 - \delta_k^{''}) \\ \sum_j a_{kj}x_j - b_k \leq (U_k + \varepsilon)\delta_k^{''} - \varepsilon \\ \delta_k = \delta_k^{'} + \delta_k^{''} - 1 \end{array} \right.$$

Proof. From table 4.2, the equivalence can be expressed as:

$$p \leftrightarrow q \equiv \underbrace{(p \rightarrow q)}_{(i_1)} \wedge \underbrace{(\neg p \rightarrow \neg q)}_{(i_2)}$$

Therefore, implications (i_1) and (i_2) must hold:

$$\text{Implication } (i_1): \quad \delta_k = 1 \rightarrow \sum_j a_{kj}x_j - b_k = 0$$

$$\text{Implication } (i_2): \quad \delta_k = 0 \rightarrow \sum_j a_{kj}x_j - b_k \neq 0 \therefore$$

$$\therefore \quad \delta_k = 0 \rightarrow \underbrace{(\sum_j a_{kj}x_j - b_k < 0)}_{(i_2^{'})} \vee \underbrace{(\sum_j a_{kj}x_j - b_k > 0)}_{(i_2^{''})}$$

With $\delta_k = 0$ in proposition 4.2.11, then:

$$\delta_k = 0 \therefore \delta_k^{'} + \delta_k^{''} = 1 \rightarrow \underbrace{[(\delta_k^{'} = 1) \wedge (\delta_k^{''} = 0)]}_{\text{Case (i)}} \vee \underbrace{[(\delta_k^{'} = 0) \wedge (\delta_k^{''} = 1)]}_{\text{Case (ii)}} \therefore$$

Case (i):

$$(\delta_k^{'} = 1) \wedge (\delta_k^{''} = 0) \rightarrow \left\{ \begin{array}{l} \sum_j a_{kj}x_j - b_k \leq 0 \\ \sum_j a_{kj}x_j - b_k \geq L_k \therefore \text{tautology} \\ \sum_j a_{kj}x_j - b_k \geq L_k \therefore \text{tautology} \\ \sum_j a_{kj}x_j - b_k \leq -\varepsilon \therefore \sum_j a_{kj}x_j - b_k < 0 \end{array} \right\} \quad \therefore$$

$$\therefore \quad (\delta_k^{'} = 1) \wedge (\delta_k^{''} = 0) \rightarrow \sum_j a_{kj}x_j - b_k < 0 \therefore (i_2^{'}) \text{ holds}$$

Case (ii):

$$(\delta_k^{'} = 0) \wedge (\delta_k^{''} = 1) \rightarrow \left\{ \begin{array}{l} \sum_j a_{kj}x_j - b_k \leq U_k \therefore \text{tautology} \\ \sum_j a_{kj}x_j - b_k \geq +\varepsilon \therefore \sum_j a_{kj}x_j - b_k > 0 \\ \sum_j a_{kj}x_j - b_k \geq 0 \\ \sum_j a_{kj}x_j - b_k \leq U_k \therefore \text{tautology} \end{array} \right\} \quad \therefore$$

$$\therefore \quad (\delta_k^{'} = 0) \wedge (\delta_k^{''} = 1) \rightarrow \sum_j a_{kj}x_j - b_k > 0 \therefore (i_2^{''}) \text{ holds}$$

By Case (i) and Case (ii), therefore:

$$\delta_k = 0 \rightarrow [\sum_j a_{kj}x_j - b_k < 0] \vee [\sum_j a_{kj}x_j - b_k > 0] \therefore$$

$$\therefore \quad \delta_k = 0 \rightarrow \sum_j a_{kj}x_j - b_k \neq 0 \therefore \text{implication } (i_2) \text{ holds}$$

With $\delta_k = 1$ in proposition 4.2.11, then:

$$\delta_k = 1 \therefore \delta_k^{'} + \delta_k^{''} = 2 \rightarrow \underbrace{(\delta_k^{'} = 1) \wedge (\delta_k^{''} = 1)}_{\text{Case (iii)}} \therefore$$

Case (iii):

$$(\delta_k^{'} = 1) \wedge (\delta_k^{''} = 1) \rightarrow \left\{ \begin{array}{l} \sum_j a_{kj}x_j - b_k \leq 0 \\ \sum_j a_{kj}x_j - b_k \geq L_k \therefore \text{tautology} \\ \sum_j a_{kj}x_j - b_k \geq 0 \\ \sum_j a_{kj}x_j - b_k \leq U_k \therefore \text{tautology} \end{array} \right\} \quad \therefore$$

$$\therefore \quad (\delta_k^{'} = 1) \wedge (\delta_k^{''} = 1) \rightarrow (\sum_j a_{kj}x_j - b_k \leq 0) \wedge (\sum_j a_{kj}x_j - b_k \geq 0) \therefore$$

$$\therefore \quad (\delta_k^{'} = 1) \wedge (\delta_k^{''} = 1) \rightarrow (\sum_j a_{kj}x_j - b_k = 0)$$

By case (iii), therefore:

$$\delta_k = 1 \rightarrow \sum_j a_{kj}x_j - b_k = 0 \therefore \text{implication } (i_1) \text{ holds}$$

By cases (i), (ii), and (iii), implications (i_1) and (i_2) hold, therefore proposition 4.2.11 holds. □

Remark 4.2.11. $(L_k \leq -\varepsilon) \wedge (U_k \geq +\varepsilon)$.

Proposition 4.2.12.

$$\left(\delta_k = 1 \leftrightarrow \sum_j a_{kj}x_j - b_k \neq 0\right) \quad \equiv \quad \left\{ \begin{array}{l} \sum_j a_{kj}x_j - b_k \leq U_k\delta_k \\ \sum_j a_{kj}x_j - b_k \geq L_k\delta_k \\ \sum_j a_{kj}x_j - b_k \leq (U_k + \varepsilon)(1 - \delta_k^{'}) - \varepsilon \\ \sum_j a_{kj}x_j - b_k \geq (L_k - \varepsilon)(1 - \delta_k^{''}) + \varepsilon \\ \delta_k = \delta_k^{'} + \delta_k^{''} \end{array} \right.$$

Proof. From table 4.2, the equivalence can be expressed as:

$$p \leftrightarrow q \equiv \underbrace{(p \rightarrow q)}_{(i_1)} \wedge \underbrace{(\neg p \rightarrow \neg q)}_{(i_2)}$$

Therefore, implications (i_1) and (i_2) must hold:

$$\begin{aligned}
\text{Implication } (i_1):\quad & \delta_k = 1 \rightarrow \sum_j a_{kj}x_j - b_k \neq 0 \therefore \\
\therefore\quad & \delta_k = 1 \rightarrow \underbrace{(\sum_j a_{kj}x_j - b_k < 0)}_{(i_1^{'})} \vee \underbrace{(\sum_j a_{kj}x_j - b_k > 0)}_{(i_1^{''})} \\
\text{Implication } (i_2):\quad & \delta_k = 0 \rightarrow \sum_j a_{kj}x_j - b_k = 0
\end{aligned}$$

With $\delta_k = 1$ in proposition 4.2.12, then:

$$\delta_k = 1 \therefore \delta_k^{'} + \delta_k^{''} = 1 \rightarrow \underbrace{[(\delta_k^{'} = 1) \wedge (\delta_k^{''} = 0)]}_{\text{Case (i)}} \vee \underbrace{[(\delta_k^{'} = 0) \wedge (\delta_k^{''} = 1)]}_{\text{Case (ii)}} \therefore$$

Case (i):

$$(\delta_k^{'} = 1) \wedge (\delta_k^{''} = 0) \rightarrow \left\{ \begin{array}{l} \sum_j a_{kj}x_j - b_k \leq U_k \therefore \text{tautology} \\ \sum_j a_{kj}x_j - b_k \geq L_k \therefore \text{tautology} \\ \sum_j a_{kj}x_j - b_k \leq -\varepsilon \therefore \sum_j a_{kj}x_j - b_k < 0 \\ \sum_j a_{kj}x_j - b_k \geq L_k \therefore \text{tautology} \end{array} \right\} \quad \therefore$$

$$\therefore\quad (\delta_k^{'} = 1) \wedge (\delta_k^{''} = 0) \rightarrow \sum_j a_{kj}x_j - b_k < 0 \therefore (i_1^{'}) \text{ holds}$$

Case (ii):

$$(\delta_k^{'} = 0) \wedge (\delta_k^{''} = 1) \rightarrow \left\{ \begin{array}{l} \sum_j a_{kj}x_j - b_k \leq U_k \therefore \text{tautology} \\ \sum_j a_{kj}x_j - b_k \geq L_k \therefore \text{tautology} \\ \sum_j a_{kj}x_j - b_k \leq U_k \therefore \text{tautology} \\ \sum_j a_{kj}x_j - b_k \geq +\varepsilon \therefore \sum_j a_{kj}x_j - b_k > 0 \end{array} \right\} \quad \therefore$$

$$\therefore\quad (\delta_k^{'} = 0) \wedge (\delta_k^{''} = 1) \rightarrow \sum_j a_{kj}x_j - b_k > 0 \therefore (i_1^{''}) \text{ holds}$$

By Case (i) and Case (ii), therefore:

$$\delta_k = 1 \rightarrow [\sum_j a_{kj}x_j - b_k < 0] \vee [\sum_j a_{kj}x_j - b_k > 0] \therefore$$

$$\therefore \quad \delta_k = 1 \rightarrow \sum_j a_{kj}x_j - b_k \neq 0 \therefore \text{implication } (i_1) \text{ holds}$$

With $\delta_k = 0$ in proposition 4.2.12, then:

$$\delta_k = 0 \therefore \delta_k^{'} + \delta_k^{''} = 0 \rightarrow \underbrace{(\delta_k^{'} = 0) \wedge (\delta_k^{''} = 0)}_{\text{Case (iii)}} \therefore$$

Case (iii):

$$(\delta_k^{'} = 0) \wedge (\delta_k^{''} = 0) \rightarrow \left\{ \begin{array}{l} \sum_j a_{kj}x_j - b_k \leq 0 \\ \sum_j a_{kj}x_j - b_k \geq 0 \\ \sum_j a_{kj}x_j - b_k \leq U_k \therefore \text{tautology} \\ \sum_j a_{kj}x_j - b_k \geq L_k \therefore \text{tautology} \end{array} \right\} \therefore$$

$$\therefore \quad (\delta_k^{'} = 0) \wedge (\delta_k^{''} = 0) \rightarrow (\sum_j a_{kj}x_j - b_k \leq 0) \wedge (\sum_j a_{kj}x_j - b_k \geq 0) \therefore$$

$$\therefore \quad (\delta_k^{'} = 0) \wedge (\delta_k^{''} = 0) \rightarrow (\sum_j a_{kj}x_j - b_k = 0)$$

By case (iii), therefore:

$$\delta_k = 0 \rightarrow \sum_j a_{kj}x_j - b_k = 0 \therefore \text{implication } (i_2) \text{ holds}$$

By cases (i), (ii), and (iii), implications (i_1) and (i_2) hold, therefore proposition 4.2.12 holds. □

Remark 4.2.12. $(L_k \leq -\varepsilon) \wedge (U_k \geq +\varepsilon)$.

Table 4.5 summarizes the logical constraints in the equivalence form (LCEF) demonstrated in propositions 4.2.7 to 4.2.12. In these propositions, "ρ" of LFC_k was considered as $\leq$, $\geq$, $<$, $>$, $=$, and $\neq$. Amongst the section 4.1 literature, just Hürlimann (1998) indicates the possibility of attaining the LCEF, but, in fact, the paper does not develop the equivalent set of MILP expressions. Hadjiconstantinou and Mitra (1994) and Mitra et al. (1994) have not addressed the development of LCEF. Therefore, this thesis fulfills this specific feature of Hürlimann (1998).

Table 4.5: Logical Constraints in the Equivalence Form.

LCEF ($\forall k \in K$)	Equivalent Set of MILP Expressions ($\forall k \in K$)	Remark ($\forall k \in K$)
$\delta_k = 1 \leftrightarrow \sum_j a_{kj}x_j \leq b_k$	$\begin{cases} \sum_j a_{kj}x_j - b_k \leq U_k(1-\delta_k) \\ \sum_j a_{kj}x_j - b_k \geq (L_k - \varepsilon)\delta_k + \varepsilon \end{cases}$	$\begin{cases} L_k \leq 0 \\ U_k \geq +\varepsilon \end{cases}$
$\delta_k = 1 \leftrightarrow \sum_j a_{kj}x_j \geq b_k$	$\begin{cases} \sum_j a_{kj}x_j - b_k \geq L_k(1-\delta_k) \\ \sum_j a_{kj}x_j - b_k \leq (U_k + \varepsilon)\delta_k - \varepsilon \end{cases}$	$\begin{cases} L_k \leq -\varepsilon \\ U_k \geq 0 \end{cases}$
$\delta_k = 1 \leftrightarrow \sum_j a_{kj}x_j < b_k$	$\begin{cases} \sum_j a_{kj}x_j - b_k \leq (U_k + \varepsilon)(1-\delta_k) - \varepsilon \\ \sum_j a_{kj}x_j - b_k \geq L_k\delta_k \end{cases}$	$\begin{cases} L_k \leq -\varepsilon \\ U_k \geq 0 \end{cases}$
$\delta_k = 1 \leftrightarrow \sum_j a_{kj}x_j > b_k$	$\begin{cases} \sum_j a_{kj}x_j - b_k \geq (L_k - \varepsilon)(1-\delta_k) + \varepsilon \\ \sum_j a_{kj}x_j - b_k \leq U_k\delta_k \end{cases}$	$\begin{cases} L_k \leq 0 \\ U_k \geq +\varepsilon \end{cases}$
$\delta_k = 1 \leftrightarrow \sum_j a_{kj}x_j = b_k$	$\begin{cases} \sum_j a_{kj}x_j - b_k \leq U_k(1-\delta_k^{'}) \\ \sum_j a_{kj}x_j - b_k \geq (L_k - \varepsilon)\delta_k^{'} + \varepsilon \\ \sum_j a_{kj}x_j - b_k \geq L_k(1-\delta_k^{''}) \\ \sum_j a_{kj}x_j - b_k \leq (U_k + \varepsilon)\delta_k^{''} - \varepsilon \\ \delta_k = \delta_k^{'} + \delta_k^{''} - 1 \end{cases}$	$\begin{cases} L_k \leq -\varepsilon \\ U_k \geq +\varepsilon \end{cases}$
$\delta_k = 1 \leftrightarrow \sum_j a_{kj}x_j \neq b_k$	$\begin{cases} \sum_j a_{kj}x_j - b_k \leq U_k\delta_k \\ \sum_j a_{kj}x_j - b_k \geq L_k\delta_k \\ \sum_j a_{kj}x_j - b_k \leq (U_k + \varepsilon)(1-\delta_k^{'}) - \varepsilon \\ \sum_j a_{kj}x_j - b_k \geq (L_k - \varepsilon)(1-\delta_k^{''}) + \varepsilon \\ \delta_k = \delta_k^{'} + \delta_k^{''} \end{cases}$	$\begin{cases} L_k \leq -\varepsilon \\ U_k \geq +\varepsilon \end{cases}$

4.3 High-Level MILP Modeling Structures

4.3.1 Either-Or Statement

In order to present the either-or statement, which is also briefly illustrated by Williams (1999) and Floudas (1995), let's initially consider the linear programming model defined in formulation 4.5, where one of the constraints c_1 or c_2 must hold. In

4.5, the symbols ρ_1 and ρ_2 denote mathematical relations such as $\leq$, $\geq$, $<$, $>$, $=$, $\neq$, and the remaining symbols were previously defined in formulation 4.2, on page 46.

$$\begin{aligned} min \quad & \sum_{j\in J} c_j x_j \\ s.t. \quad & (c_1) : \sum_{j\in J} a_{1j} x_j \{\rho_1\} b_1 \\ & (c_2) : \sum_{j\in J} a_{2j} x_j \{\rho_2\} b_2 \\ & x_j \geq 0 \quad \forall j \in J \end{aligned} \tag{4.5}$$

The condition that one of the constraints c_1 or c_2 must hold cannot be directly specified in a linear programming framework, where, by definition, all constraints must hold. In order to overcome this difficulty, a rationale similar to the one applied on propositions 4.2.1 to 4.2.12 is adopted. The idea is to use lower (L_k) and upper (U_k) bounds on constraints c_k (c_1 and c_2), a small tolerance value ε ($\varepsilon > 0$), and a binary variable δ to properly express the either-or relation between constraints.

Considering a function f_k, which is dependent on L_k, U_k, ε, and δ, then c_1 and c_2 can be rewritten as indicate in 4.6.

$$\begin{aligned} (c_1) : \sum_{j\in J} a_{1j} x_j - b_1 \{\rho_1\} f_1(L_1, U_1, \varepsilon, \neg\delta) \\ (c_2) : \sum_{j\in J} a_{2j} x_j - b_2 \{\rho_2\} f_2(L_2, U_2, \varepsilon, \delta) \end{aligned} \tag{4.6}$$

Functions f_1 and f_2 must establish that, in case $\delta = 0$ one constraint is obligatorily imposed, and the other is relaxed; when $\delta = 1$ the situation is reversed, and the constraint that was relaxed is imposed and the other is relaxed. Hence, in all cases one of the constraints is imposed and the other is relaxed. Nevertheless, it is important to be aware that the relaxed constraint may also hold. The equivalent "mixed integer program" is specified in formulation 4.7.

$$
\begin{aligned}
min \quad & \sum_{j\in J} c_j x_j \\
s.t. \quad & (c_1): \sum_{j\in J} a_{1j}x_j - b_1\{\rho_1\} f_1(L_1, U_1, \varepsilon, \neg\delta) \\
& (c_2): \sum_{j\in J} a_{2j}x_j - b_2\{\rho_2\} f_2(L_2, U_2, \varepsilon, \delta) \\
& x_j \geq 0 \quad \forall j \in J
\end{aligned}
\tag{4.7}
$$

Tables 4.6, 4.7, and 4.8 contain the final format of constraints "c_1" and "c_2". This format is, in fact, a set of MILP expressions that establish the either-or relations between c_1 and c_2, according to ρ_1, f_1, and ρ_2, f_2. In these tables, the factor e_1 is equivalent to $\sum_{j\in J} a_{1j}x_j - b_1$, and the factor e_2 is equivalent to $\sum_{j\in J} a_{2j}x_j - b_2$. The equivalence between the either-or and the set of MILP expressions is not herein proved, but the rationale is similar to the one applied on propositions 4.2.1 to 4.2.12. For instance, let's take table 4.6, with ρ_1 and ρ_2 equal to $\leq$. Then, the set of equivalent MILP expressions is illustrated in inequalities 4.8.

$$
\begin{aligned}
(c_1): \sum_{j\in J} a_{1j}x_j - b_1\{\rho_1\} f_1(L_1, U_1, \varepsilon, \neg\delta) \quad &\equiv \quad \sum_{j\in J} a_{1j}x_j - b_1 \leq U_1\delta \\
(c_2): \sum_{j\in J} a_{2j}x_j - b_2\{\rho_2\} f_2(L_2, U_2, \varepsilon, \delta) \quad &\equiv \quad \sum_{j\in J} a_{2j}x_j - b_2 \leq U_2(1-\delta)
\end{aligned}
\tag{4.8}
$$

In 4.8, when $\delta = 0$, then c_1 holds and c_2 is relaxed[6]. In case $\delta = 1$, then c_1 is relaxed and c_2 holds. Therefore, either c_1 or c_2 must hold. The remaining either-or statements of tables 4.6, 4.7, and 4.8 can be understood according to the same reasoning illustrated in 4.8.

In particular, the literature of section 4.1 does not present a complete list of either-or statements, as tables 4.6, 4.7, and 4.8 indeed present. Therefore, these tables aid the modeling of either-or statements.

[6]The inequality $\sum_{j\in J} a_{2j}x_j - b_2 \leq U_2$ is a tautology.

Table 4.6: Either-Or Statement: $\rho_1 \in \{\leq, \geq\}$ and $\rho_2 \in \{\leq, \geq, <, >, =, \neq\}$.

ρ_1, ρ_2	Set of MILP Expressions	ρ_1, ρ_2	Set of MILP Expressions
$\leq, \leq$	$\begin{cases} e_1 \leq U_1\delta \\ e_2 \leq U_2(1-\delta) \end{cases}$	$\geq, \leq$	$\begin{cases} e_1 \geq L_1\delta \\ e_2 \leq U_2(1-\delta) \end{cases}$
$\leq, \geq$	$\begin{cases} e_1 \leq U_1\delta \\ e_2 \geq L_2(1-\delta) \end{cases}$	$\geq, \geq$	$\begin{cases} e_1 \geq L_1\delta \\ e_2 \geq L_2(1-\delta) \end{cases}$
$\leq, <$	$\begin{cases} e_1 \leq U_1\delta \\ e_2 \leq (U_2+\varepsilon)(1-\delta)-\varepsilon \end{cases}$	$\geq, <$	$\begin{cases} e_1 \geq L_1\delta \\ e_2 \leq (U_2+\varepsilon)(1-\delta)-\varepsilon \end{cases}$
$\leq, >$	$\begin{cases} e_1 \leq U_1\delta \\ e_2 \geq (L_2-\varepsilon)(1-\delta)+\varepsilon \end{cases}$	$\geq, >$	$\begin{cases} e_1 \geq L_1\delta \\ e_2 \geq (L_2-\varepsilon)(1-\delta)+\varepsilon \end{cases}$
$\leq, =$	$\begin{cases} e_1 \leq U_1\delta \\ e_2 \leq U_2(1-\delta) \\ e_2 \geq L_2(1-\delta) \end{cases}$	$\geq, =$	$\begin{cases} e_1 \geq L_1\delta \\ e_2 \leq U_2(1-\delta) \\ e_2 \geq L_2(1-\delta) \end{cases}$
$\leq, \neq$	$\begin{cases} e_1 \leq U_1\delta \\ e_2 \leq (U_2+\varepsilon)(1-\delta^{'})-\varepsilon \\ e_2 \geq (L_2-\varepsilon)(1-\delta^{''})+\varepsilon \\ \delta = \delta^{'}+\delta^{''} \end{cases}$	$\geq, \neq$	$\begin{cases} e_1 \geq L_1\delta \\ e_2 \leq (U_2+\varepsilon)(1-\delta^{'})-\varepsilon \\ e_2 \geq (L_2-\varepsilon)(1-\delta^{''})+\varepsilon \\ \delta = \delta^{'}+\delta^{''} \end{cases}$

Table 4.7: Either-Or Statement: $\rho_1 \in \{<, >\}$ and $\rho_2 \in \{\leq, \geq, <, >, =, \neq\}$.

ρ_1, ρ_2	Set of MILP Expressions	ρ_1, ρ_2	Set of MILP Expressions
$<, \leq$	$\begin{cases} e_1 \leq (U_1+\varepsilon)\delta-\varepsilon \\ e_2 \leq U_2(1-\delta) \end{cases}$	$>, \leq$	$\begin{cases} e_1 \geq (L_1-\varepsilon)\delta+\varepsilon \\ e_2 \leq U_2(1-\delta) \end{cases}$
$<, \geq$	$\begin{cases} e_1 \leq (U_1+\varepsilon)\delta-\varepsilon \\ e_2 \geq L_2(1-\delta) \end{cases}$	$>, \geq$	$\begin{cases} e_1 \geq (L_1-\varepsilon)\delta+\varepsilon \\ e_2 \geq L_2(1-\delta) \end{cases}$
$<, <$	$\begin{cases} e_1 \leq (U_1+\varepsilon)\delta-\varepsilon \\ e_2 \leq (U_2+\varepsilon)(1-\delta)-\varepsilon \end{cases}$	$>, <$	$\begin{cases} e_1 \geq (L_1-\varepsilon)\delta+\varepsilon \\ e_2 \leq (U_2+\varepsilon)(1-\delta)-\varepsilon \end{cases}$
$<, >$	$\begin{cases} e_1 \leq (U_1+\varepsilon)\delta-\varepsilon \\ e_2 \geq (L_2-\varepsilon)(1-\delta)+\varepsilon \end{cases}$	$>, >$	$\begin{cases} e_1 \geq (L_1-\varepsilon)\delta+\varepsilon \\ e_2 \geq (L_2-\varepsilon)(1-\delta)+\varepsilon \end{cases}$
$<, =$	$\begin{cases} e_1 \leq (U_1+\varepsilon)\delta-\varepsilon \\ e_2 \leq U_2(1-\delta) \\ e_2 \geq L_2(1-\delta) \end{cases}$	$>, =$	$\begin{cases} e_1 \geq (L_1-\varepsilon)\delta+\varepsilon \\ e_2 \leq U_2(1-\delta) \\ e_2 \geq L_2(1-\delta) \end{cases}$
$<, \neq$	$\begin{cases} e_1 \leq (U_1+\varepsilon)\delta-\varepsilon \\ e_2 \leq (U_2+\varepsilon)(1-\delta^{'})-\varepsilon \\ e_2 \geq (L_2-\varepsilon)(1-\delta^{''})+\varepsilon \\ \delta = \delta^{'}+\delta^{''} \end{cases}$	$>, \neq$	$\begin{cases} e_1 \geq (L_1-\varepsilon)\delta+\varepsilon \\ e_2 \leq (U_2+\varepsilon)(1-\delta^{'})-\varepsilon \\ e_2 \geq (L_2-\varepsilon)(1-\delta^{''})+\varepsilon \\ \delta = \delta^{'}+\delta^{''} \end{cases}$

Table 4.8: Either-Or Statement: $\rho_1 \in \{=, \neq\}$ and $\rho_2 \in \{\leq, \geq, <, >, =, \neq\}$.

ρ_1, ρ_2	Set of MILP Expressions	ρ_1, ρ_2	Set of MILP Expressions
$=, \leq$	$\begin{cases} e_1 \leq U_1\delta \\ e_1 \geq L_1\delta \\ e_2 \leq U_2(1-\delta) \end{cases}$	$\neq, \leq$	$\begin{cases} e_1 \leq (U_1+\varepsilon)(1-\delta^{'})-\varepsilon \\ e_1 \geq (L_1-\varepsilon)(1-\delta^{''})+\varepsilon \\ e_2 \leq U_2(1-\delta) \\ \delta = 1-\delta^{'}-\delta^{''} \end{cases}$
$=, \geq$	$\begin{cases} e_1 \leq U_1\delta \\ e_1 \geq L_1\delta \\ e_2 \geq L_2(1-\delta) \end{cases}$	$\neq, \geq$	$\begin{cases} e_1 \leq (U_1+\varepsilon)(1-\delta^{'})-\varepsilon \\ e_1 \geq (L_1-\varepsilon)(1-\delta^{''})+\varepsilon \\ e_2 \geq L_2(1-\delta) \\ \delta = 1-\delta^{'}-\delta^{''} \end{cases}$
$=, <$	$\begin{cases} e_1 \leq U_1\delta \\ e_1 \geq L_1\delta \\ e_2 \leq (U_2+\varepsilon)(1-\delta)-\varepsilon \end{cases}$	$\neq, <$	$\begin{cases} e_1 \leq (U_1+\varepsilon)(1-\delta^{'})-\varepsilon \\ e_1 \geq (L_1-\varepsilon)(1-\delta^{''})+\varepsilon \\ e_2 \leq (U_2+\varepsilon)(1-\delta)-\varepsilon \\ \delta = 1-\delta^{'}-\delta^{''} \end{cases}$
$=, >$	$\begin{cases} e_1 \leq U_1\delta \\ e_1 \geq L_1\delta \\ e_2 \geq (L_2-\varepsilon)(1-\delta)+\varepsilon \end{cases}$	$\neq, >$	$\begin{cases} e_1 \leq (U_1+\varepsilon)(1-\delta^{'})-\varepsilon \\ e_1 \geq (L_1-\varepsilon)(1-\delta^{''})+\varepsilon \\ e_2 \geq (L_2-\varepsilon)(1-\delta)+\varepsilon \\ \delta = 1-\delta^{'}-\delta^{''} \end{cases}$
$=, =$	$\begin{cases} e_1 \leq U_1\delta \\ e_1 \geq L_1\delta \\ e_2 \leq U_2(1-\delta) \\ e_2 \geq L_2(1-\delta) \end{cases}$	$\neq, =$	$\begin{cases} e_1 \leq (U_1+\varepsilon)(1-\delta^{'})-\varepsilon \\ e_1 \geq (L_1-\varepsilon)(1-\delta^{''})+\varepsilon \\ e_2 \leq U_2(1-\delta) \\ e_2 \geq L_2(1-\delta) \\ \delta = 1-\delta^{'}-\delta^{''} \end{cases}$
$=, \neq$	$\begin{cases} e_1 \leq U_1\delta \\ e_1 \geq L_1\delta \\ e_2 \leq (U_2+\varepsilon)(1-\delta^{'})-\varepsilon \\ e_2 \geq (L_2-\varepsilon)(1-\delta^{''})+\varepsilon \\ \delta = \delta^{'}+\delta^{''} \end{cases}$	$\neq, \neq$	$\begin{cases} e_1 \leq (U_1+\varepsilon)(1-\delta_1^{'})-\varepsilon \\ e_1 \geq (L_1-\varepsilon)(1-\delta_1^{''})+\varepsilon \\ e_2 \leq (U_2+\varepsilon)(1-\delta_2^{'})-\varepsilon \\ e_2 \geq (L_2-\varepsilon)(1-\delta_2^{''})+\varepsilon \\ \delta = 1-\delta_1^{'}-\delta_1^{''} \\ \delta = \delta_2^{'}+\delta_2^{''} \end{cases}$

4.3.2 If-Then Statement

The if-then statement is herewith considered as having the form: *If* δ *then* p, where δ is a binary variable[7], and p is a linear constraint. In order to understand this statement, let's initially consider the proposition 4.3.1, which highlights a relation between the if-then statement and the logical constraint in the implication form (LCIF), defined on page 46.

Proposition 4.3.1.

$$\textit{If } \delta \textit{ then } p \quad \equiv \quad (\delta \rightarrow p)$$

Proof. This is trivial, since the proposition matches the if-then statement definition. Hence, if $\delta = 1$ implies p to be true; if $\delta = 0$, then p is relaxed. Therefore, If δ then p is equivalent to $(\delta \rightarrow p)$. □

Remark 4.3.1. Proposition 4.3.1 demonstrates that the LCIF and the if-then statement are interrelated. Table 4.4 summarizes the LCIF previously established in propositions 4.2.1 to 4.2.6. Therefore, table 4.4 can be adapted to generate table 4.9, which indicates a series of if-then statements and the equivalent MILP expression of such statements.

4.3.3 If-Then-Else Statement

The if-then-else statement is herewith considered as having the form: *If* δ *then* p *else* q, where δ is a binary variable, p and q are linear constraints. In order to understand this statement, let's initially consider the proposition 4.3.2, which highlights a relation between the if-then-else statement and the logical constraint in the equivalence form (LCEF), previously defined on page 52.

Proposition 4.3.2.

$$\textit{If } \delta \textit{ then } p \textit{ else } \neg p \quad \equiv \quad (\delta \leftrightarrow p)$$

[7]By convention, δ is true whether it is equal to one, zero otherwise. Therefore, for simplicity, the if-then statement is expressed by *If* δ *then* p instead of *If* $\delta = 1$ *then* p.

Table 4.9: If-Then Statement.

If-Then Statement ($\forall k \in K$)	Equivalent Set of MILP Expressions ($\forall k \in K$)	Remark ($\forall k \in K$)
If δ_k then $\sum_j a_{kj}x_j \leq b_k$	$\sum_j a_{kj}x_j - b_k \leq U_k(1-\delta_k)$	$L_k \leq 0$
If δ_k then $\sum_j a_{kj}x_j \geq b_k$	$\sum_j a_{kj}x_j - b_k \geq L_k(1-\delta_k)$	$U_k \geq 0$
If δ_k then $\sum_j a_{kj}x_j < b_k$	$\sum_j a_{kj}x_j - b_k \leq (U_k+\varepsilon)(1-\delta_k) - \varepsilon$	$L_k \leq -\varepsilon$
If δ_k then $\sum_j a_{kj}x_j > b_k$	$\sum_j a_{kj}x_j - b_k \geq (L_k-\varepsilon)(1-\delta_k) + \varepsilon$	$U_k \geq +\varepsilon$
If δ_k then $\sum_j a_{kj}x_j = b_k$	$\left\{\begin{array}{l}\sum_j a_{kj}x_j - b_k \leq U_k(1-\delta_k)\\ \sum_j a_{kj}x_j - b_k \geq L_k(1-\delta_k)\end{array}\right.$	$\left\{\begin{array}{l}L_k \leq 0\\ U_k \geq 0\end{array}\right.$
If δ_k then $\sum_j a_{kj}x_j \neq b_k$	$\left\{\begin{array}{l}\sum_j a_{kj}x_j - b_k \leq (U_k+\varepsilon)(1-\delta_k^{'}) - \varepsilon\\ \sum_j a_{kj}x_j - b_k \geq (L_k-\varepsilon)(1-\delta_k^{''}) + \varepsilon\\ \delta_k = \delta_k^{'} + \delta_k^{''}\end{array}\right.$	$\left\{\begin{array}{l}L_k \leq -\varepsilon\\ U_k \geq +\varepsilon\end{array}\right.$

Proof. By table 4.2, $(\delta \leftrightarrow p) \equiv (\delta \rightarrow p) \wedge (\neg\delta \rightarrow \neg p)$. Hence, if $\delta = 1$ implies p to be true; if $\delta = 0$ implies $\neg p$ to be true. Therefore, If δ then p else $\neg p$ is equivalent to $(\delta \leftrightarrow p)$. □

Remark 4.3.2. Proposition 4.3.2 demonstrates that the LCEF is a special case of the if δ then p else q statement, where $q = \neg p$. Table 4.5 summarizes the LCEF previously established in propositions 4.2.7 to 4.2.12. Therefore, table 4.5 can be adapted to generate table 4.10, which indicates a series of particular if-then-else statements and the equivalent MILP expression of such statements.

Considering proposition 4.3.2, it is evident that the if-then-else statement has a parallel with logical connectives, mainly the implication. Proposition 4.3.3 indicates this parallel.

Proposition 4.3.3.

$$\textit{If } \delta \textit{ then } p \textit{ else } q \quad \equiv \quad (\delta \rightarrow p) \wedge (\neg\delta \rightarrow q)$$

Proof. This is trivial, since the proposition matches the if-then-else statement definition. Hence, if $\delta = 1$ implies p to be true; if $\delta = 0$ implies q to be true. Therefore,

Table 4.10: If-Then-Else Statement.

If-Then-Else Statement ($\forall k \in K$)	Equivalent Set of MILP Expressions ($\forall k \in K$)	Remark ($\forall k \in K$)
If $\delta_k \begin{cases} \text{then } \sum_j a_{kj}x_j \le b_k \\ \text{else } \sum_j a_{kj}x_j > b_k \end{cases}$	$\begin{cases} \sum_j a_{kj}x_j - b_k \le U_k(1-\delta_k) \\ \sum_j a_{kj}x_j - b_k \ge (L_k - \varepsilon)\delta_k + \varepsilon \end{cases}$	$\begin{cases} L_k \le 0 \\ U_k \ge +\varepsilon \end{cases}$
If $\delta_k \begin{cases} \text{then } \sum_j a_{kj}x_j \ge b_k \\ \text{else } \sum_j a_{kj}x_j < b_k \end{cases}$	$\begin{cases} \sum_j a_{kj}x_j - b_k \ge L_k(1-\delta_k) \\ \sum_j a_{kj}x_j - b_k \le (U_k + \varepsilon)\delta_k - \varepsilon \end{cases}$	$\begin{cases} L_k \le -\varepsilon \\ U_k \ge 0 \end{cases}$
If $\delta_k \begin{cases} \text{then } \sum_j a_{kj}x_j < b_k \\ \text{else } \sum_j a_{kj}x_j \ge b_k \end{cases}$	$\begin{cases} \sum_j a_{kj}x_j - b_k \le (U_k + \varepsilon)(1-\delta_k) - \varepsilon \\ \sum_j a_{kj}x_j - b_k \ge L_k\delta_k \end{cases}$	$\begin{cases} L_k \le -\varepsilon \\ U_k \ge 0 \end{cases}$
If $\delta_k \begin{cases} \text{then } \sum_j a_{kj}x_j > b_k \\ \text{else } \sum_j a_{kj}x_j \le b_k \end{cases}$	$\begin{cases} \sum_j a_{kj}x_j - b_k \ge (L_k - \varepsilon)(1-\delta_k) + \varepsilon \\ \sum_j a_{kj}x_j - b_k \le U_k\delta_k \end{cases}$	$\begin{cases} L_k \le 0 \\ U_k \ge +\varepsilon \end{cases}$
If $\delta_k \begin{cases} \text{then } \sum_j a_{kj}x_j = b_k \\ \text{else } \sum_j a_{kj}x_j \ne b_k \end{cases}$	$\begin{cases} \sum_j a_{kj}x_j - b_k \le U_k(1-\delta_k^{'}) \\ \sum_j a_{kj}x_j - b_k \ge (L_k - \varepsilon)\delta_k^{'} + \varepsilon \\ \sum_j a_{kj}x_j - b_k \ge L_k(1-\delta_k^{''}) \\ \sum_j a_{kj}x_j - b_k \le (U_k + \varepsilon)\delta_k^{''} - \varepsilon \\ \delta_k = \delta_k^{'} + \delta_k^{''} - 1 \end{cases}$	$\begin{cases} L_k \le -\varepsilon \\ U_k \ge +\varepsilon \end{cases}$
If $\delta_k \begin{cases} \text{then } \sum_j a_{kj}x_j \ne b_k \\ \text{else } \sum_j a_{kj}x_j = b_k \end{cases}$	$\begin{cases} \sum_j a_{kj}x_j - b_k \le U_k\delta_k \\ \sum_j a_{kj}x_j - b_k \ge L_k\delta_k \\ \sum_j a_{kj}x_j - b_k \le (U_k + \varepsilon)(1-\delta_k^{'}) - \varepsilon \\ \sum_j a_{kj}x_j - b_k \ge (L_k - \varepsilon)(1-\delta_k^{''}) + \varepsilon \\ \delta_k = \delta_k^{'} + \delta_k^{''} \end{cases}$	$\begin{cases} L_k \le -\varepsilon \\ U_k \ge +\varepsilon \end{cases}$

If δ then p else q is equivalent to $(\delta \rightarrow p) \wedge (\neg\delta \rightarrow q)$. □

Remark 4.3.3. The if-then-else statement poses, in fact, some similarities with the either-or statement, previously discussed in section 4.3.1. If $\delta = 1$ implies c_2 of formulation 4.7 to hold (a statement p is imposed). If $\delta = 0$ implies c_1 of formulation 4.7 to hold (a statement q is imposed). Thus, there is a parallel between the if-then-else and the either-or statements. Therefore, tables 4.6, 4.7, and 4.8 can be also used to formulate "if-then-else" constructions.

4.3.4 Remarks on the Use of δ

The binary indicator variable δ, which appears in both the if-then and the if-then-else statements, can represent the truth or falsehood of a compound logical proposition. Furthermore, this proposition can also be logically related with linear constraints, as demonstrated in section 4.2.3. In order to illustrate this fact, let's consider a hypothetical example, where the following condition has to be modeled in MILP: "If $f(x) \leq 0$ and $g(x) > 0$ then $h(x) \geq 0$ else $h(x) < 0$", where $f(x)$, $g(x)$, and $h(x)$ are linear constraints.

Initially, it is necessary to formulate the statement $f(x) \leq 0$ and $g(x) > 0$ as a compound logical proposition associated with δ. The logical constraints in the equivalence form, summarized in table 4.5, allow the association of binary variables δ_f and δ_g with, respectively, $f(x) \leq 0$ and $g(x) > 0$, as indicated in equivalences 4.9 and 4.10. In 4.9 and 4.10, U_f and U_g are the upper bounds on, respectively, $f(x)$ and $g(x)$; L_f and L_g are the lower bounds on, respectively, $f(x)$ and $g(x)$; ε is a positive small tolerance value ($\varepsilon > 0$).

$$\delta_f = 1 \leftrightarrow f(x) \leq 0 \quad \equiv \quad \begin{cases} f(x) \leq U_f(1-\delta_f) \\ f(x) \geq (L_f - \varepsilon)\delta_f + \varepsilon \end{cases} \tag{4.9}$$

$$\delta_g = 1 \leftrightarrow g(x) > 0 \quad \equiv \quad \begin{cases} g(x) \leq U_g\delta_g \\ g(x) \geq (L_g - \varepsilon)(1-\delta_g) + \varepsilon \end{cases} \tag{4.10}$$

For this particular example, $\delta \leftrightarrow (\delta_f \wedge \delta_g)$. According to table 4.2, $p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$. Therefore, $\delta \leftrightarrow (\delta_f \wedge \delta_g)$ is equivalent to $[\delta \rightarrow (\delta_f \wedge \delta_g)] \wedge [(\delta_f \wedge \delta_g) \rightarrow \delta]$. By table 4.3, the implication $\delta \rightarrow (\delta_f \wedge \delta_g)$ can be expressed by the linear constraints $\delta \leq \delta_f$ and $\delta \leq \delta_g$; the implication $(\delta_f \wedge \delta_g) \rightarrow \delta$ can be expressed by the linear constraint $\delta_f + \delta_g - \delta \leq 1$. Therefore, the statement "$\delta = 1$ if and only if $f(x) \leq 0$ and $g(x) > 0$" is expressed in the set of inequalities 4.11.

$$\begin{cases} f(x) \leq U_f(1-\delta_f) \\ f(x) \geq (L_f - \varepsilon)\delta_f + \varepsilon \\ g(x) \leq U_g \delta_g \\ g(x) \geq (L_g - \varepsilon)(1-\delta_g) + \varepsilon \\ \delta_f + \delta_g - \delta \leq 1 \\ \delta \leq \delta_f \\ \delta \leq \delta_g \end{cases} \tag{4.11}$$

In addition, proposition 4.3.2 states that "If δ then p else $\neg p$" is equivalent to $\delta \leftrightarrow p$. Thus, the if-then-else statement "If δ then $h(x) \geq 0$ else $h(x) < 0$" can be expressed according to equivalence 4.12, which is obtained from table 4.5. In 4.12, U_h and L_h are, respectively, the upper and lower bounds on $h(x)$; ε is a positive small tolerance value ($\varepsilon > 0$).

$$\delta = 1 \leftrightarrow h(x) \geq 0 \quad \equiv \quad \begin{cases} h(x) \geq L_h(1-\delta) \\ h(x) \leq (U_h + \varepsilon)\delta - \varepsilon \end{cases} \tag{4.12}$$

Therefore, the initial statement "If $f(x) \leq 0$ and $g(x) > 0$ then $h(x) \geq 0$ else $h(x) < 0$" can be represented by the MILP set of inequalities indicated in formulation 4.13.

$$\begin{cases} f(x) \leq U_f(1-\delta_f) \\ f(x) \geq (L_f-\varepsilon)\delta_f+\varepsilon \\ g(x) \leq U_g\delta_g \\ g(x) \geq (L_g-\varepsilon)(1-\delta_g)+\varepsilon \\ h(x) \geq L_h(1-\delta) \\ h(x) \leq (U_h+\varepsilon)\delta-\varepsilon \\ \delta_f+\delta_g-\delta \leq 1 \\ \delta \leq \delta_f \\ \delta \leq \delta_g \end{cases} \tag{4.13}$$

4.4 An Illustrative Example of Flexible Storage

In order to illustrate the concepts previously described in sections 4.2 and 4.3, a small example of MILP modeling, which is adapted from Hürlimann (1998), is presented:

> Let's suppose that a refinery produces two oil products A and B in unknown quantities x and y. The products are stored in tanks. The company owns two tanks with capacities a and b. The products cannot be mixed in the same tank. Normally, product A is stored in the first tank and product B in the second one. Therefore, due to storage constraints, the company can produce only the maximum quantity a of product A, and the maximum quantity b of product B. Occasionally, only one product is produced and both tanks can be used to store it. In this case, the company may produce the product at a maximum quantity of $a+b$. Hence, either the company produces both products at quantities less than or equal to a and b, or it produces only one product with quantity less than or equal to $a+b$. How to express such operational conditions in a set of MILP constraints?

The feasible space is a disjunctive set. Therefore, the situation cannot be formulated as a single LP model. One alternative is to create three convex spaces (LP_1, LP_2, and LP_3), as indicated in formulation 4.14[8].

$$\begin{aligned} &LP_1 : x \leq a \text{ and } y \leq b \\ &LP_2 : x \leq a + b \text{ and } y \leq 0 \\ &LP_3 : x \leq 0 \text{ and } y \leq a + b \end{aligned} \tag{4.14}$$

The overall non-convex set can be expressed as $LP_1 \otimes LP_2 \otimes LP_3$. Therefore, the problem can be formulated as a single constraint containing logical and mathematical operators, as indicated in 4.15.

$$(x \leq a \wedge y \leq b) \otimes (x \leq a + b \wedge y \leq 0) \otimes (x \leq 0 \wedge y \leq a + b) \tag{4.15}$$

The disjunctive formulation 4.15 can be translated into a logical form as indicated in 4.16.

$$\begin{aligned} &\delta_1 = 1 \rightarrow (x \leq a) \wedge (y \leq b) \\ &\delta_2 = 1 \rightarrow (x \leq a + b) \wedge (y \leq 0) \\ &\delta_3 = 1 \rightarrow (x \leq 0) \wedge (y \leq a + b) \\ &\delta_1 \otimes \delta_2 \otimes \delta_3 \end{aligned} \tag{4.16}$$

Formulation 4.17 can be obtained by observing that each implication in 4.16 can be independently applied.

$$\begin{aligned} &\delta_1 = 1 \rightarrow x \leq a \\ &\delta_1 = 1 \rightarrow y \leq b \\ &\delta_2 = 1 \rightarrow x \leq a + b \\ &\delta_2 = 1 \rightarrow y \leq 0 \\ &\delta_3 = 1 \rightarrow x \leq 0 \\ &\delta_3 = 1 \rightarrow y \leq a + b \\ &\delta_1 \otimes \delta_2 \otimes \delta_3 \end{aligned} \tag{4.17}$$

[8]See commentary about 4.14 on page 72, paragraph before formulation 4.18.

In a simplified point of view, the lower and upper bounds on constraints can be determined based on the minimum/maximum storage capacity. Therefore, $L_k = 0$, $U_k = a + b\ \forall k \in K$. Using tables 4.3 and 4.4, the disjunctive formulation 4.17 can be translated into an MILP, as indicated in 4.18.

In addition, it is important to highlight that, in 4.14, the terms $y \leq 0$ and $x \leq 0$ could have been specified as, respectively, $y = 0$ and $x = 0$. However, in propositions 4.2.1 to 4.2.12, strict inequalities ($<$, $>$), equality ($=$), and disequality ($\neq$) operators are harder to express than simple inequalities ($\leq$, $\geq$). Therefore, if it were used $y = 0$ and $x = 0$ instead of $y \leq 0$ and $x \leq 0$, this would have increased the number of constraints in formulation 4.18. Thus, as long as possible, it is advisable to use $\leq$ and $\geq$, which yield simplified sets of MILP constraints.

$$\begin{aligned}
&x - a \leq (a+b)(1-\delta_1)\\
&y - b \leq (a+b)(1-\delta_1)\\
&x - (a+b) \leq (a+b)(1-\delta_2)\\
&y \leq (a+b)(1-\delta_2)\\
&x \leq (a+b)(1-\delta_3)\\
&y - (a+b) \leq (a+b)(1-\delta_3)\\
&\delta_1 + \delta_2 + \delta_3 = 1
\end{aligned} \tag{4.18}$$

This is just a simple example, but the direct translation of the problem description into an MILP model is not straightforward, even for this simple example. In addition, the translation task tends to be much more difficult as the problem becomes more complex. However, the intermediate logical structures 4.14 to 4.17 are simpler to formulate than the MILP set of inequalities 4.18. Thus, the use of intermediate logical structures seems to be of great value to aid the MILP modeling process.

4.5 Remarks on Using MILP Modeling Structures

This chapter provides some MILP modeling structures that help the expression of MILP models. However, the structures provided do not necessarily achieve the most computationally efficient model, that is, they do not provide the sharpest MILP formulation. This is not a specific fail of this work, and, in fact, the literature (Hadjiconstantinou and Mitra, 1994; Mitra et al., 1994; Yeom and Lee, 1998; Hürlimann, 1998) has also pointed out this particular negative feature of MILP reformulation procedures. The merit of this work remains on simplifying the logical expression of constraints in an MILP framework. It is also important to highlight that Hürlimann (1998) presents a procedure that translates logical constraints into mathematical constraints containing zero-one variables, and this procedure is completely integrated into a modeling system, the LPL (Hürlimann, 2000). This chapter presents some high-level logical structures that help to build MILP models. However, after the logical formulation task, the user has to also translate the high-level formulation to the base-level MILP formulation. There is no automatic reformulation procedure implemented.

Chapters 1 to 3 stated that the CLP modeling features overcome the MILP modeling features, which are essentially based on inequalities. However, in an integrated CLP-MILP approach, the MILP modeling, which is a hard task (see section 4.1), is indispensable. Sections 4.2 to 4.3 present connections between logical inference and MILP, which can be used to facilitate the MILP modeling task. Thus, the high-level MILP structures presented in chapter 4 can be seamlessly used in an integrated CLP-MILP modeling framework. The overall CLP-MILP modeling approach gains versatility, since CLP is well-known by its rich modeling framework, and the high-level MILP structures also facilitate the model builder task.

Chapter 5

MILP: Application of the High-Level Modeling Structures

This chapter presents an MILP formulation addressing a combinatorial problem. This problem is focused on issues regarding the oil industry, more specifically, involving the management of a multi-product pipeline. The main goal is to demonstrate the applicability of the high-level MILP modeling structures developed in chapter 4.

5.1 Introduction to the Problem Context

The oil industry has a strong influence upon the economic market. Research in this area may provide highly profitable solutions and also avoid environmental damages. The oil distribution-planning problem is within this context. A wide net with trains, tankers, and pipelines are used to link harbors, refineries, and consumers. According to Kennedy (1993), pipelines provide an efficient way to transport oil and gas, and making good use of this transportation medium becomes interesting to the oil industry. However, the operational decision-making in pipeline systems is still based on experience, with the aid of manual calculations. According to Lee et al. (1996), mathematical programming techniques for planning have been extensively studied and implemented, but much less work has been devoted to short-term scheduling, which in fact reproduces the operational decision-making process.

In general lines, the short-term scheduling problem involves the optimal exploitation of available resources in order to optimize a plant production over a given time horizon. These resources consist of, basically, (raw) materials, production units, utilities, and manpower. The problem is to determine the optimal assignment of production tasks to operating units, the best task sequence, and the optimal operating level, in order to optimize a given production objective (Ierapetritou and Floudas, 1998). The short-term scheduling requires the explicit modeling of discrete decisions. The approach to solve this problem is manifold. A general one is to use a mixed integer linear programming (MILP) formulation[1]. One great concern of short-term scheduling problems formulated by MILP models is related to the difficulty of finding solutions in a reasonable computational time (Reklaitis, 1992). According to Applequist et al. (1997), an MILP feature of a practical problem requires a large number of integer variables, and, therefore, the computational expense has to be concerned. Subrahmanyam et al. (1995) demonstrate that decomposition strategies are a valid approach to avoid the combinatorial explosion introduced by integer variables. In addition, Wu and Ierapetritou (2003) present a series of decomposition-based approaches for the efficient solution of short-term scheduling problems, which are, otherwise, intractable.

Another great concern of short-term scheduling problems modeled by MILP is the time representation. In general, such representation can be categorized in two main streams: discrete time formulations and continuous time formulations. In the former, the scheduling horizon is discretized into a number of equal duration intervals, and all events must occur at the edge of the fixed intervals; in the latter, the events can occur at any continuous time in the scheduling horizon, and the occurrence of real-event points determines the timing feature (event-driven formulation). In the continuous time domain, the discretization of the scheduling horizon into a number of subintervals is generally avoided. Reklaitis (1992) thoroughly describes these time representation approaches, but, in a simplified point of view, the major limitation at discrete time formulations is that the time quantum must be specified

[1]Section 2.2 on page 13 brings an explanation about MILP.

to equal the greatest common divisor of all event-duration. For instance, if task processing times range from ten hours to fifteen minutes, the latter value must be chosen for the discretization. If the problem data are rounded, the time quantum can be increased, but the obtained solution can be either too conservative or infeasible. If the problem data are accurately treated, then the model can become too large for routine solution. On the other hand, continuous time formulations tend to produce smaller models than equivalent discrete time formulations, however the integrality gap[2] tends to be bigger. Therefore, it is not possible to ensure whether a short-term scheduling MILP model is computationally efficient just because it employs discrete or continuous time representation (Reklaitis, 1992).

This chapter addresses a short-term scheduling problem by MILP techniques. The problem description is presented in section 5.2, and the methodology used to deal with this problem is explained in section 5.3.

5.2 Problem Description

The considered problem involves the short-term scheduling of activities in a specific pipeline, which connects a harbor to an inland refinery. Figure 5.1 illustrates the pipeline physical structure overview.

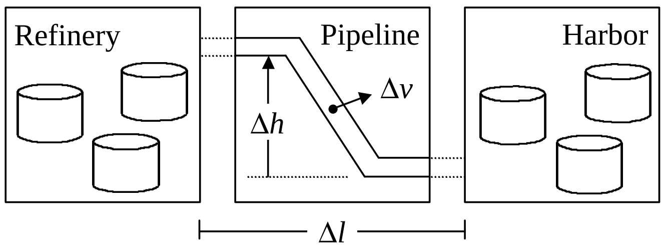


Figure 5.1: Pipeline Physical Structure Overview.

The pipeline presents 93.5 km in length (Δl), it can store a total volume of 7314 m^3 (Δv), and it connects a refinery tank farm to a harbor tank farm going along regions with 900-meter-altitude difference (Δh). The pipe conveys different

[2]The integrality gap is defined in section 2.2.

types of products (gasoline, diesel, kerosene, liquefied petroleum gas, jet fuel, etc), which are, mainly, oil products. Thus, the set of products conveyed by the pipe is limited. The pipeline always operates completely filled, and there is no physical separation between successive products as they are pumped. Consequently, there is a contamination area between miscible products: the *interface*. Some interfaces are operationally not recommended, and a *plug* (small volume of product) can be used to avoid them, even though, *plug* inclusions increase the operational cost. The products can be pumped either from the refinery to the harbor (this is called *"flow"* procedure) or from the harbor to the refinery (this is called *"reflow"* procedure)[3]. A complete pumping operation covers either a *flow* procedure followed by a *reflow* procedure or a *reflow* followed by a *flow* procedure. The tank farm infrastructure, an up-to-date storage scenario, the pipeline flow rate details, and the demand requirements are known a priori. The main task is to specify the pipeline operation during a limited scheduling horizon (H), providing low cost operational procedures. Nevertheless, the scheduling process must also take into account some issues concerning pumping sequence, flow rate determination, and operational requirements. The operational cost is basically influenced by the usage of *plugs*, the time period that the pipe remains pressurized and idle, and the seasonal cost of electric energy (on-peak demand hours). Magatão (2001) presents a detailed description of the problem features, as well as the factors that influence the operational cost. Section 5.4, which describes the mathematical model, also presents additional information about the problem characteristics.

5.3 Methodology

This pipeline-scheduling problem was already addressed by the author of this thesis and co-workers in Magatão et al. (2001), Magatão et al. (2002), and Magatão et al. (2004). The core methodology applied in these papers is mixed integer linear

[3]In fact, *"flow"* procedure and *"reflow"* procedure are operational terms that would be better defined as, respectively, *flow forwards* procedure and *flow backwards* procedure. For simplicity, this thesis uses the operational terms *flow* and *reflow*.

programming, but the high-level MILP modeling structures presented in chapter 4 were not used. In Magatão et al. (2001), the authors developed a monolithic MILP model that relied on uniform time discretization. They demonstrated that even small problem instances have potentially large integer search spaces, and the computational burden should have been deeply considered.

In Magatão et al. (2002) the authors indeed considered the computational expense previously detected by Magatão et al. (2001). The problem was divided in small entities, which were based on the three key elements of scheduling: assignment of resources, sequencing of activities, and determination of resource timing utilization by these activities[4]. The idea was to share the basic scheduling elements among an integrated architecture, which provided a framework that was aimed at reducing the computational expense.

Similarly to the work of Magatão et al. (2002), in Magatão et al. (2004) the authors also developed an optimization structure based on MILP with decomposition strategies. In general, the latter approach tends to produce better scheduling answers than the former one[5]. Thus, the optimization structure henceforth used in this chapter, which is illustrated in figure 5.2, is based on Magatão et al. (2004).

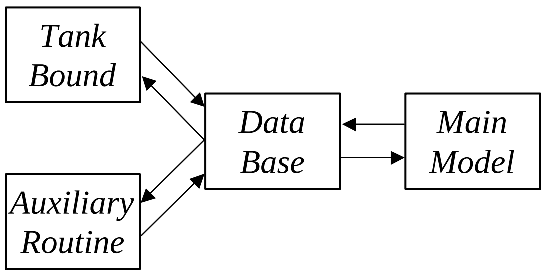


Figure 5.2: Optimization Structure.

The optimization structure proposed by Magatão et al. (2004) is based on an MILP main model (*Main Model*), one auxiliary MILP model (*Tank Bound*), a time computation procedure (*Auxiliary Routine*), and a database (*Data Base*), which gathers the input data and the information provided by the other optimization

[4]The key elements of scheduling are explained, for instance, by Reklaitis (1992).
[5]An early comparison between optimization structures is presented by Magatão (2001).

blocks. The *Tank Bound* task involves the appropriate selection of some resources (tanks) for a given activity (the pumping of demanded products). Its main inputs are demand requirements, product availability, and tankage constraints. As an output, it specifies the tanks to be used in operational procedures.

The *Auxiliary Routine* takes into account the available time horizon, the product flow rate range, and demand requirements. It specifies temporal constraints (time-windows), which must be respected by the *Main Model*.

The *Main Model*, which is based on MILP with uniform time discretization, determines the product pumping sequence and it establishes the initial and the final time of each pumping activity. The final scheduling is attained by first solving the *Tank Bound* and the *Auxiliary Routine*, and, at last, the *Main Model*. The *Main Model* must respect the parameters previously determined by the *Auxiliary Routine*.

This chapter uses the optimization structure illustrated in figure 5.2, but there exist two fundamental differences in relation to the work of Magatão et al. (2004):

(i) The *Main Model* is based on MILP with continuous time approach;

(ii) The *Auxiliary Routine* was reformulated in order to satisfy the continuous time approach.

The *Tank Bound*, however, is essentially the same model of Magatão et al. (2004), which specifies the tanks to be used during the operational procedures. Therefore, for simplicity, the *Tank Bound* is not herein discussed, as well as the features involving the storage resources. The interested reader should consult Magatão et al. (2004) and Magatão (2001).

The continuous time approach was chosen because of two main reasons:

(i) Section 5.1 stated that it is not possible to ensure whether a short-term scheduling MILP model is computationally efficient just because it employs discrete or continuous time representation. Discrete time formulations were applied in the previous work addressing the same problem (Magatão et al., 2001; Magatão et al., 2002; Magatão et al., 2004). Therefore, investigating a continuous formulation seemed to be a necessity in order to verify whether such continuous approach could better suit operational issues or not;

(ii) Discrete time representations tend to handle constraints with ease, since these are explicitly enforced at the discrete time points; on the other hand, continuous time formulations tend to be more challenging from a modeling standpoint (Kondili et al., 1993). This fact turns out to be valuable, because the MILP modeling structures developed in chapter 4 can be applied in a "difficult" continuous time formulation.

5.4 Mathematical Formulation

The general guidelines used to create the mathematical formulation are summarized in conditions 1 to 12, henceforth described:

1. Products can be pumped from two different *origins*: either refinery or harbor. The refinery is the origin for the *flow* procedure; the harbor is the origin for the *reflow* procedure. The set O is used to represent these different origins. By convention, $o = 1$ indicates the *flow* procedure, and $o = 2$ indicates the *reflow* procedure. Therefore, $o \in O$, $O = \{1, 2\}$;

2. A subset of products is pumped from each origin ($p \in P_o$) during the scheduling horizon. A complete pumping operation covers either a *reflow* procedure followed by a *flow* procedure or a *flow* procedure followed by a *reflow* procedure. Therefore, the pipeline scenario involves both *flow forwards* and *flow backwards* procedures. Such a particular operational condition has to be incorporated into the mathematical model;

3. The product flow rate, which is within a range, must be respected. Therefore, there is a minimum and a maximum time interval to pump each product;

4. Each product is pumped only once (one batch) throughout the scheduling horizon. This operational procedure contributes to minimize *interfaces* losses;

5. The set of demanded products must be pumped during the scheduling horizon;
6. A minimum scheduling horizon (H^{min}) must be available for pumping procedures, otherwise ($H < H^{min}$) the problem is infeasible. At $H = H^{min}$ each product is pumped at its maximum flow rate;
7. The pipeline stores 7314 m^3 and it always operates completely filled. There is a time delay in between pumping a product and receiving it. Therefore, an extra product amount must be pumped after the last *flow* sequenced product and the last *reflow* sequenced product. This extra product amount is used to maintain the pipeline filled with no interfaces inside the pipe (see figure 5.10 on page 113);
8. The *plug* volume is significantly smaller than any demanded product volume, so that, its pumping time is neglected;
9. *Plug* inclusions increase the operational cost, so that, use of *plugs* should be minimized;
10. Set-up times are neglected;
11. The pipeline can be stopped during the scheduling horizon. However, there is an operational cost to maintain the pipeline without pumping (pressurized and idle)[6];
12. The electric energy has seasonal costs. Typically, from 6 p.m. to 9 p.m. (on-peak demand hours) the cost is higher than in the rest of the day (COPEL, 2005). Therefore, during the scheduling horizon, the electric energy has intervals that receive different electric cost rates (see figure 5.4 on page 95).

5.4.1 Notation

This section details the notation adopted to formulate the mathematical model. This notation is indispensable for the formulation understanding.

[6]In order to maintain the pipeline pressurized and idle (without pumping), the system operator can, for instance, determine that inlet/outlet valves must remain closed.

Sets

E	set of on-peak electric cost intervals inside the scheduling horizon, $E = \{1, 2, \ldots, ne\}$ (the general parameter ne is defined on page 84);
O	set of origins, $O = \{1, 2\}$;
P_o	set of demanded products from each origin o, $P_1 = \{1, 2, \ldots, np_1\}$ and $P_2 = \{np_1 + 1, np_1 + 2, \ldots, np_1 + np_2\}$ (np_1 and np_2 are defined on page 84 - see the general parameter np_o).

Subscripts

e	electric cost ($e \in E$);
$o,\ \bar{o}$	origins ($o \in O,\ \bar{o} \in O,\ O = \{1, 2\}$)[7];
$p,\ pa$	products ($p \in P_o,\ pa \in P_o$).

General Parameters

C_o^{low}	the electric energy has seasonal costs, which receive different electric cost rates; the parameter C_o^{low} indicates the standard electric cost rate per hour applied when products are pumped from origin o (\$/h);
C_o^{high}	the electric energy has seasonal costs, which receive different electric cost rates; the parameter C_o^{high} indicates the additional value to the standard electric cost rate per hour (C_o^{low}); C_o^{high} is applied when products are pumped from origin o during on-peak demand intervals (\$/h);
C_o^{plug}	average cost to pump a *plug* product from origin o (\$);
C^{pres}	average cost per hour to maintain the pipeline pressurized and idle (\$/h);
d	defines the origin (either refinery or harbor) that starts the pumping procedure ($d \in \{1, 2\},\ d + \bar{d} = 3$)[8]; if $d = 1$ and $\bar{d} = 2$ then the *flow* procedure starts the pumping process, otherwise ($d = 2$ and $\bar{d} = 1$) the *reflow* procedure starts the pumping process;

[7]It is important to notice that $\bar{o}$ is not the complement of o, but rather just a notation.

[8]It is important to notice that $\bar{d}$ is not the complement of d, but rather just a notation.

$\bar{d}$	defines the origin (either harbor or refinery) that finishes the pumping procedure ($\bar{d} \in \{1,2\},\ d+\bar{d}=3$); if $d=1$ and $\bar{d}=2$ then the *reflow* procedure finishes the pumping process, otherwise (d=2 and $\bar{d}$=1) the *flow* procedure finishes the pumping process;
$dem_{p,o}$	demanded amount of product p from origin o (m^3);
ε	non-dimensional small tolerance value ($\varepsilon > 0$, e.g. $\varepsilon = 10^{-3}$) used to express if-then and if-then-else statements (see details in section 4.2.3);
$\varphi_{p,o}^{max}$	maximum flow rate of product p from origin o (m^3/h);
$\varphi_{p,o}^{min}$	minimum flow rate of product p from origin o (m^3/h), when the pipeline is not pressurized and idle[9];
fh_e	during the scheduling horizon, the electric energy has intervals that receive different electric cost rates (see figure 5.4 on page 95, intervals e=1, e=2, ..., e=ne); the parameter fh_e indicates the time value that the on-peak electric cost e finishes (h);
H	scheduling horizon (h);
L	non-dimensional value (lower bound) used to express if-then and if-then-else statements (see section 4.2.3). In particular, the fixed value $L = -(\text{maximum}(H)+1)$ was used;
ne	number of on-peak electric cost intervals during the scheduling horizon (see figure 5.6, page 102);
np_o	number of demanded products from each origin o;
pin	product that establishes the interface with the first pumped p of origin d (origin that starts the pumping procedure). Indeed, pin is already inside the pipe at the initial scheduling time;
$plug_{p,pa}$	1 if the interface between products p and pa demands a *plug*, 0 otherwise;
pv	pipeline internal volume (m^3);

[9]Indeed, when the pipeline is pressurized and idle, the flow rate is equal to zero. However, $\varphi_{p,o}^{min}$ is the minimum allowable flow rate value when product p from origin o is being pumped.

sh_e	during the scheduling horizon, the electric energy has intervals that receive different electric cost rates (see figure 5.4, intervals $e{=}1$, $e{=}2, \ldots, e{=}ne$); the parameter sh_e indicates the time value that the on-peak electric cost e starts (h);
U	non-dimensional value (upper bound) used to express if-then and if-then-else statements (see section 4.2.3). In particular, the fixed value $U = +(\text{maximum}(H) + 1)$ was used.

Parameters Determined by the Auxiliary Routine

$d^{max}_{p,o}$	maximum pumping duration of product p from origin o (h), when p is pumped uninterruptedly;
$d^{min}_{p,o}$	minimum pumping duration of product p from origin o (h);
fd	available time (h) to complete either the *flow* procedure ($d{=}1$, $\bar{d}{=}2$) or the *reflow* procedure ($d{=}2$, $\bar{d}{=}1$) - see figure 5.3 on page 88;
fd^{min}	lower time limit (h) to complete either the *flow* procedure ($d{=}1$, $\bar{d}{=}2$) or the *reflow* procedure ($d{=}2$, $\bar{d}{=}1$) - see figure 5.3;
H^{min}	minimum scheduling horizon (h) to complete pumping procedures (*flow* and *reflow*) - see figure 5.3;
$td^{max}_{p,o}$	maximum pumping duration to fill up the pipeline uninterruptedly with product p from origin o (h);
$td^{min}_{p,o}$	minimum pumping duration to fill up the pipeline with product p from origin o (h).

Main Model Variables

$\delta^{00}_{p,o,e}$	continuous variable used to indicate the pumping temporal position of product p from origin o in relation to electric cost interval e (see figure 5.4, positions 1 and 2) - in fact, just binary values are assigned to $\delta^{00}_{p,o,e}$;
$\delta^{01}_{p,o,e}$	continuous variable used to indicate the pumping temporal position of product p from origin o in relation to electric cost interval e (see figure 5.4, position 3) - in fact, just binary values are assigned to $\delta^{01}_{p,o,e}$;

$\delta^{10}_{p,o,e}$	continuous variable used to indicate the pumping temporal position of product p from origin o in relation to electric cost interval e (see figure 5.4, position 4) - in fact, just binary values are assigned to $\delta^{10}_{p,o,e}$;
$\delta^{11}_{p,o,e}$	continuous variable used to indicate the pumping temporal position of product p from origin o in relation to electric cost interval e (see figure 5.4, positions 5 and 6) - in fact, just binary values are assigned to $\delta^{11}_{p,o,e}$;
$dpres$	continuous variable that indicates the time period that the pipeline remains pressurized and idle during the scheduling horizon (h);
$f_{p,o}$	continuous variable that indicates the pumping finish time of product p from origin o (h);
$fgh_{p,o,e}$	binary variable that indicates whether $f_{p,o}$ is greater than or equal fh_e ($fgh_{p,o,e} = 1$), or otherwise ($fgh_{p,o,e} = 0$) - see figure 5.4;
$first_{p,o}$	binary variable that indicates whether p is the first pumped product from origin o ($first_{p,o} = 1$), or otherwise ($first_{p,o} = 0$);
$gt_{p,pa,o}$	binary variable that indicates whether product p is pumped before product pa ($gt_{p,pa,o} = 1$), or otherwise ($gt_{p,pa,o} = 0$) - both products are pumped from origin o;
$inh_{p,o,e}$	continuous variable that indicates, together with $stop_{p,o,e}$, the pumping duration of product p from origin o in the electric cost interval e (h);
$last_{p,o}$	binary variable that indicates whether p is the last pumped product from origin o ($last_{p,o} = 1$), or otherwise ($last_{p,o} = 0$);
$s_{p,o}$	continuous variable that indicates the pumping start time of product p from origin o (h);
$seq_{p,o}$	continuous variable that indicates the pumping sequence of product p from origin o (in fact, just integer values are assigned to $seq_{p,o}$);
$sgh_{p,o,e}$	binary variable that indicates whether $s_{p,o}$ is greater than or equal sh_e ($sgh_{p,o,e} = 1$), or otherwise ($sgh_{p,o,e} = 0$) - see figure 5.4;
$stop_{p,o,e}$	continuous variable that indicates, together with $inh_{p,o,e}$, the time period that product p from origin o is maintained pressurized and idle during the electric cost interval e (h);

$t_{p,pa,o}$	binary variable that indicates whether the pumping of product p is immediately followed by the pumping of product pa and, therefore, the interface $p - pa$ is established ($t_{p,pa,o} = 1$), or otherwise ($t_{p,pa,o} = 0$) - both products are pumped from origin o;
$tw_{p,pa}$	binary variable that indicates whether the pumping of product p from origin o is followed by the pumping of product pa from origin $\bar{o}$ ($tw_{p,pa} = 1$), or otherwise ($tw_{p,pa} = 0$).

5.4.2 Auxiliary Routine

Equations 5.1 to 5.7 define parameters determined by the *Auxiliary Routine*. These parameters, in fact, establish time windows that must be respected by the pumping procedures. Figure 5.3 illustrates such time windows.

$$d_{p,o}^{min} = dem_{p,o}/\varphi_{p,o}^{max} \qquad \forall o \in O,\ p \in P_o \tag{5.1}$$

$$d_{p,o}^{max} = dem_{p,o}/\varphi_{p,o}^{min} \qquad \forall o \in O,\ p \in P_o \tag{5.2}$$

$$td_{p,o}^{min} = pv/\varphi_{p,o}^{max} \qquad \forall o \in O,\ p \in P_o \tag{5.3}$$

$$td_{p,o}^{max} = pv/\varphi_{p,o}^{min} \qquad \forall o \in O,\ p \in P_o \tag{5.4}$$

$$fd^{min} = \sum_{\substack{o=d,\\ p\in P_o}} d_{p,o}^{min} + \min_{\substack{o=d,\\ p\in P_o}}(td_{p,o}^{min}) \tag{5.5}$$

$$H^{min} = \sum_{\substack{o=d,\\ p\in P_o}} d_{p,o}^{min} + \sum_{\substack{\bar{o}=\bar{d},\\ pa\in P_{\bar{o}}}} d_{pa,\bar{o}}^{min} + \min_{\substack{o=d,\\ p\in P_o}}(td_{p,o}^{min}) + \min_{\substack{\bar{o}=\bar{d},\\ pa\in P_{\bar{o}}}}(td_{pa,\bar{o}}^{min}) \tag{5.6}$$

$$fd = fd^{min} + H - H^{min} \tag{5.7}$$

5.4.3 Main Model

The *Main Model* is created based on the aforementioned conditions 1 to 12 (page 81). Another important premise is considering a continuous time approach. Therefore, the events, such as the pumping finish of a product p, can occur at any continuous time value, inside the scheduling horizon. The *Main Model* formulation

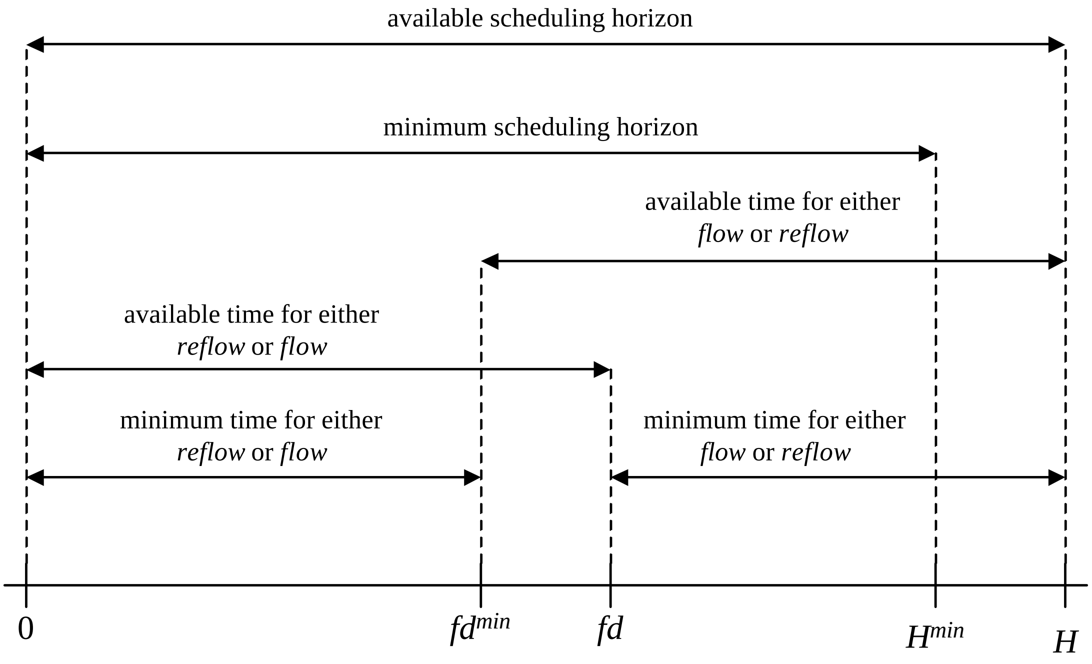

available scheduling horizon
minimum scheduling horizon
available time for either *flow* or *reflow*
available time for either *reflow* or *flow*
minimum time for either *reflow* or *flow*
minimum time for either *flow* or *reflow*
0
fd^{min}
fd
H^{min}
H

Figure 5.3: Auxiliary Routine - An Illustrative Case.

also used the abstraction of temporal blocks, which define the pumping start time ($s_{p,o}$) and the pumping completion time ($f_{p,o}$) of each required product p, in each origin o (see figure 5.4, page 95). The initial and the final boundary of such blocks determine the values of, respectively, $s_{p,o}$ and $f_{p,o}$, which are continuous variables. These continuous variables are used to establish whether a product p from origin o is pumped before a product pa from origin o ($gt_{p,pa,o} = 1$), or otherwise ($gt_{p,pa,o} = 0$) - see expression 5.9.

Based on the value of $gt_{p,pa,o}$, the variables $seq_{p,o}$, $t_{p,pa,o}$, $first_{p,o}$, $last_{p,o}$, and $tw_{p,pa}$ are used to specify the pumping sequence details (see expressions 5.10 to 5.20). The values of $s_{p,o}$ and $f_{p,o}$ are also used to establish the time period that a product is pumped ($inh_{p,o,e}$) or remains pressurized and idle ($stop_{p,o,e}$) during on-peak demand time intervals of electric energy. The variables $fgh_{p,o,e}$, $sgh_{p,o,e}$, $\delta^{00}_{p,o,e}$, $\delta^{01}_{p,o,e}$, $\delta^{10}_{p,o,e}$, and $\delta^{11}_{p,o,e}$ are created in order to detected such electric cost conditions (see expressions 5.21 to 5.27 and figure 5.4). The variable called $dpres$ is used to specify the time period that the pipe remains pressurized and idle (see equation 5.28). In addition, expressions 5.29 to 5.35 are used to determine the minimum/maximum allowable duration of pumping activities.

The high-level MILP modeling structures developed in chapter 4 are widely used in the *Main Model* formulation. Thus, for simplicity, the reader should consult tables 4.9 (page 66) and 4.10 (page 67).

5.4.3.1 Main Model: Objective Function

The *Main Model* objective function (expression 5.8) defines the operational cost minimization. This function is weighed by the following cost factors:

(i) Electric cost variations ($C_o^{low}\ \forall o \in O$, $C_o^{high}\ \forall o \in O$);

(ii) Cost of using *plugs* ($C_o^{plug}\ \forall o \in O$); and,

(iii) Cost of maintaining the pipeline pressurized and idle (C^{pres}).

$$\begin{aligned}
\texttt{minimize}\quad & \\
& \sum_{o \in O} \sum_{p \in P_o} C_o^{low} \cdot (f_{p,o} - s_{p,o}) \\
+\ & \sum_{o \in O} \sum_{p \in P_o} \sum_{e \in E} C_o^{high} \cdot (inh_{p,o,e} - stop_{p,o,e}) \\
+\ & \sum_{o \in O} \sum_{p \in P_o} \sum_{\substack{pa \in P_o, \\ pa \neq p}} C_o^{plug} \cdot t_{p,pa,o} \cdot plug_{p,pa} \\
+\ & \sum_{\substack{o = d, \\ p \in P_o}} \sum_{\substack{\bar{o} = \bar{d}, \\ pa \in P_{\bar{o}}}} C_{\bar{o}}^{plug} \cdot tw_{p,pa} \cdot plug_{p,pa} \\
+\ & \sum_{\substack{p = pin, \\ o = d, \\ pa \in P_o}} C_o^{plug} \cdot first_{p,o} \cdot plug_{p,pa} \\
+\ & C^{pres} \cdot dpres
\end{aligned} \tag{5.8}$$

In a simplified standpoint, expression 5.8 indicates that the time period that a product is pumped during on-peak demand intervals is detected by the model ($inh_{p,o,e} - stop_{p,o,e}$), and receives an additional charge (C_o^{high}). Thus, the solution method seeks scheduling answers that avoid pumping products during on-peak demand hours. At this case, alternatively, the pipe can be maintained pressurized and idle ($stop_{p,o,e} > 0$), but this condition also influences the operational cost by the term "$C^{pres} \cdot dpres$" (see equation 5.28 on page 96). In addition, *plug* inclusions increase the operational cost, and the optimization method has to seek scheduling solutions that minimize the *plug* usage. Therefore, a series of factors must be considered in order to determine the scheduling condition that yields the minimum operational cost.

5.4.3.2 Main Model: Constraints

The *Main Model* is also subject to constraints, which are expressed in 5.9 to 5.35. In the if-then-else statement 5.9, the truth of the binary variable $gt_{p,pa,o}$ ($gt_{p,pa,o}=1$) indicates that product p is pumped before product pa ($f_{p,o} \leq s_{pa,o}$); the falsehood of such variable ($gt_{p,pa,o}=0$) implies that $f_{p,o} > s_{pa,o}$. Both products are pumped from origin o. The inequalities written in brace are, in fact, the equivalent set of MILP "base-level" expressions for the "high-level" if-then-else statement. The base-level is obtained by table 4.10. In fact, general MILP modeling languages can only interpret base-level (in)equalities[10].

$$\textbf{If } gt_{p,pa,o} \textbf{ Then } f_{p,o} \leq s_{pa,o} \textbf{ Else } f_{p,o} > s_{pa,o} \quad \forall o \in O,\ p \in P_o,\ pa \in P_o,\ p \neq pa$$

$$\begin{cases} f_{p,o} - s_{pa,o} \leq U \cdot (1 - gt_{p,pa,o}) & \forall o \in O,\ p \in P_o,\ pa \in P_o,\ p \neq pa \\ f_{p,o} - s_{pa,o} \geq (L - \varepsilon) \cdot gt_{p,pa,o} + \varepsilon & \forall o \in O,\ p \in P_o,\ pa \in P_o,\ p \neq pa \end{cases} \tag{5.9}$$

Equations 5.10 and 5.11 are used to assign the product pumping sequence to variable $seq_{p,o}$. For example, $seq_{1,1} = 2$ indicates that product one ($p = 1$) is the second to be pumped during the *flow* operation ($o = 1$).

$$\sum_{p \in P_o} \sum_{\substack{pa \in P_o, \\ pa \neq p}} gt_{p,pa,o} = \sum_{k=1}^{k=np_o} (k-1) \qquad \forall o \in O \tag{5.10}$$

$$seq_{p,o} = np_o - \sum_{\substack{pa \in P_o, \\ pa \neq p}} gt_{p,pa,o} \qquad \forall o \in O,\ p \in P_o \tag{5.11}$$

In order to exemplify the working of equations 5.10 and 5.11, let's consider a hypothetical case where $np_o = 4\ \forall o \in O$. By the if-then-else statement 5.9, the truth

[10]For simplicity, single values of U and L were used in the *Main Model* formulation instead of U_k and L_k, suggested in tables 4.9 and 4.10. In particular, the fixed values $U = +(\text{maximum}(H) + 1)$ and $L = -(\text{maximum}(H) + 1)$ were used. These values are attained by observing that the upper/lower bounds on all if-then and if-then-else "base-level" inequalities of section 5.4.3 cannot exceed the scheduling horizon value.

of the binary variable $gt_{p,pa,o}$ implies that product p is pumped before product pa. Therefore, for the first sequenced product, $\sum_{\substack{pa\in P_o,\\ pa\neq p}} gt_{p,pa,o}\ \forall o\in O,\ p\in P_o$ is equal to three (the first sequenced product is pumped before three other products); for the second sequenced product, $\sum_{\substack{pa\in P_o,\\ pa\neq p}} gt_{p,pa,o}\ \forall o\in O,\ p\in P_o$ is equal to two; for the third sequenced product, the value of $\sum_{\substack{pa\in P_o,\\ pa\neq p}} gt_{p,pa,o}\ \forall o\in O,\ p\in P_o$ is equal to one; for the fourth sequenced product, $\sum_{\substack{pa\in P_o,\\ pa\neq p}} gt_{p,pa,o}\ \forall o\in O,\ p\in P_o$ is equal to zero. By equation 5.10, for each origin o, the value of $\sum_{k=1}^{k=np_o}(k-1)$ is equal to six (0+1+2+3), and, therefore, $\sum_{p\in P_o}\sum_{\substack{pa\in P_o,\\ pa\neq p}} gt_{p,pa,o} = 6\ \forall o\in O$. By equation 5.11, for all origins o and for each product $p\in P_o$, if $\sum_{\substack{pa\in P_o,\\ pa\neq p}} gt_{p,pa,o}$ is equal to three, then, p is the first sequenced product from origin o ($seq_{p,o} = 4-3$). If $\sum_{\substack{pa\in P_o,\\ pa\neq p}} gt_{p,pa,o}$ is equal to zero, then, p is the last sequenced product from origin o ($seq_{p,o} = 4-0$). The other sequence positions are attained in an analogous way.

In the if-then statement 5.12, the truth of the binary variable $t_{p,pa,o}$ implies that the pumping of product p is immediately followed by the pumping of product pa. Both products are pumped from origin o.

$$\textbf{If}\quad t_{p,pa,o}\quad \textbf{Then}\quad seq_{pa,o}-seq_{p,o}=1\quad \forall o\in O,\ p\in P_o,\ pa\in P_o,\ p\neq pa$$
$$\begin{cases} seq_{pa,o}-seq_{p,o}-1\le U\cdot(1-t_{p,pa,o}) & \forall o\in O,\ p\in P_o,\ pa\in P_o,\ p\neq pa\\ seq_{pa,o}-seq_{p,o}-1\ge L\cdot(1-t_{p,pa,o}) & \forall o\in O,\ p\in P_o,\ pa\in P_o,\ p\neq pa \end{cases} \tag{5.12}$$

Equation 5.13 and inequalities 5.14 and 5.15 establish that a product must be pumped only once throughout the scheduling horizon.

$$\sum_{p\in P_o}\sum_{\substack{pa\in P_o,\\ pa\neq p}} t_{p,pa,o} = np_o - 1 \qquad \forall o\in O \tag{5.13}$$

$$\sum_{\substack{pa\in P_o,\\ pa\neq p}} t_{p,pa,o} \le 1 \qquad \forall o\in O,\ p\in P_o \tag{5.14}$$

$$\sum_{\substack{pa\in P_o,\\ pa\neq p}} t_{pa,p,o} \le 1 \qquad \forall o\in O,\ p\in P_o \tag{5.15}$$

The if-then statement 5.16 and the equation 5.17 determine that the truth of the binary variable $first_{p,o}$ implies p to be the first pumped product from origin o. The same effect of the if-then statement 5.16 and the equation 5.17 could have been achieved by just the unnumbered if-then-else statement that appears right after equation 5.17[11]. However, the reader should be aware that this if-then-else formulation would increase the number of variables and constraints in comparison with the if-then statement 5.16 and the equation 5.17 (e.g. the auxiliary binary variables $first'_{p,o}$, $first''_{p,o}$ $\forall o \in O,\ p \in P_o$ would have been created if the if-then-else formulation were used). In general, it is advisable to look for compact formulations.

$$\begin{gathered}
\textbf{If} \quad first_{p,o} \quad \textbf{Then} \quad seq_{p,o} = 1 \qquad \forall o \in O,\ p \in P_o \\
\begin{cases} seq_{p,o} - 1 \leq U \cdot (1 - first_{p,o}) & \forall o \in O,\ p \in P_o \\ seq_{p,o} - 1 \geq L \cdot (1 - first_{p,o}) & \forall o \in O,\ p \in P_o \end{cases}
\end{gathered} \tag{5.16}$$

$$\sum_{p \in P_o} first_{p,o} = 1 \qquad \forall o \in O \tag{5.17}$$

$$\begin{gathered}
\textbf{If} \quad first_{p,o} \quad \textbf{Then} \quad seq_{p,o} = 1 \quad \textbf{Else} \quad seq_{p,o} \neq 1 \qquad \forall o \in O,\ p \in P_o \\
\begin{cases} seq_{p,o} - 1 \leq U \cdot (1 - first'_{p,o}) & \forall o \in O,\ p \in P_o \\ seq_{p,o} - 1 \geq (L - \varepsilon) \cdot first'_{p,o} + \varepsilon & \forall o \in O,\ p \in P_o \\ seq_{p,o} - 1 \geq L \cdot (1 - first''_{p,o}) & \forall o \in O,\ p \in P_o \\ seq_{p,o} - 1 \leq (U + \varepsilon) \cdot first''_{p,o} - \varepsilon & \forall o \in O,\ p \in P_o \\ first_{p,o} = first'_{p,o} + first''_{p,o} - 1 & \forall o \in O,\ p \in P_o \\ first'_{p,o},\ first''_{p,o} \in \{0, 1\} & \forall o \in O,\ p \in P_o \end{cases}
\end{gathered}$$

The if-then statement 5.18 and the equation 5.19 determine that the truth of the binary variable $last_{p,o}$ implies p to be the last pumped product from origin o.

$$\begin{gathered}
\textbf{If} \quad last_{p,o} \quad \textbf{Then} \quad seq_{p,o} = np_o \qquad \forall o \in O,\ p \in P_o \\
\begin{cases} seq_{p,o} - np_o \leq U \cdot (1 - last_{p,o}) & \forall o \in O,\ p \in P_o \\ seq_{p,o} - np_o \geq L \cdot (1 - last_{p,o}) & \forall o \in O,\ p \in P_o \end{cases}
\end{gathered} \tag{5.18}$$

$$\sum_{p \in P_o} last_{p,o} = 1 \qquad \forall o \in O \tag{5.19}$$

[11]This if-then-else statement is unnumbered because it was not implemented. Instead of it, the if-then statement 5.16 and the equation 5.17 were indeed implemented.

Expression 5.20 establishes that a binary variable $tw_{p,pa}$ is set to one if the pumping of product p from origin o is followed by the pumping of product pa from origin $\bar{o}$, otherwise $tw_{p,pa}$ is set to zero. The inequalities written in brace are, in fact, the MILP base-level expression of the *and* ($\wedge$) statement.

The expression of *and* statements involving binary variables is a well-known subject. Considering, for example, that $c = a \wedge b$, with both a and b binary variables. Then, the equivalent formulation for the *and* statement is indicated by the following set of inequalities: $c \leq a$, $c \leq b$, and $c \geq a + b - 1$ (Schrage, 2000).

$$
\begin{array}{ll}
tw_{p,pa} = (last_{p,o}) \wedge (first_{pa,\bar{o}}) & \forall p \in P_o,\ pa \in P_{\bar{o}},\ o = d,\ \bar{o} = \bar{d} \\
\left\{ \begin{array}{l} tw_{p,pa} \leq last_{p,o} \\ tw_{p,pa} \leq first_{pa,\bar{o}} \\ tw_{p,pa} \geq last_{p,o} + first_{pa,\bar{o}} - 1 \end{array} \right. & \begin{array}{l} \forall p \in P_o,\ pa \in P_{\bar{o}},\ o = d,\ \bar{o} = \bar{d} \\ \forall p \in P_o,\ pa \in P_{\bar{o}},\ o = d,\ \bar{o} = \bar{d} \\ \forall p \in P_o,\ pa \in P_{\bar{o}},\ o = d,\ \bar{o} = \bar{d} \end{array}
\end{array}
\qquad (5.20)
$$

Electric Cost: On-peak Intervals. One great concern of specialists in using pipeline-scheduling models in real-world scenarios is the difference between theoretical models and practical needs. One example of an important practical feature to be addressed is the seasonal variation of electric energy cost. Such cost variation represents on-peak demand hours that are charged at different rates. In a discrete time representation, the seasonal electric cost variation can be addressed with ease, since cost differences are explicitly enforced at the discrete time points (e.g. Magatão et al., 2004). On the other hand, to address electric cost variations on a continuous time formulation tends to be more challenging from a modeling standpoint, as herein presented in expressions 5.21 to 5.28.

The if-then-else statements 5.21 and 5.22 establish that if the binary indicator variables $fgh_{p,o,e}$ and $sgh_{p,o,e}$ are set to one, then, respectively, the conditions $f_{p,o} \geq fh_e$ and $s_{p,o} \geq sh_e$ must hold; otherwise, the conditions $f_{p,o} < fh_e$ and $s_{p,o} < sh_e$ must hold. These binary variables are used to identify the relative position of a pumping time block defined by product p in relation to an on-peak electric time interval e (see figure 5.4 and expressions 5.23 to 5.26).

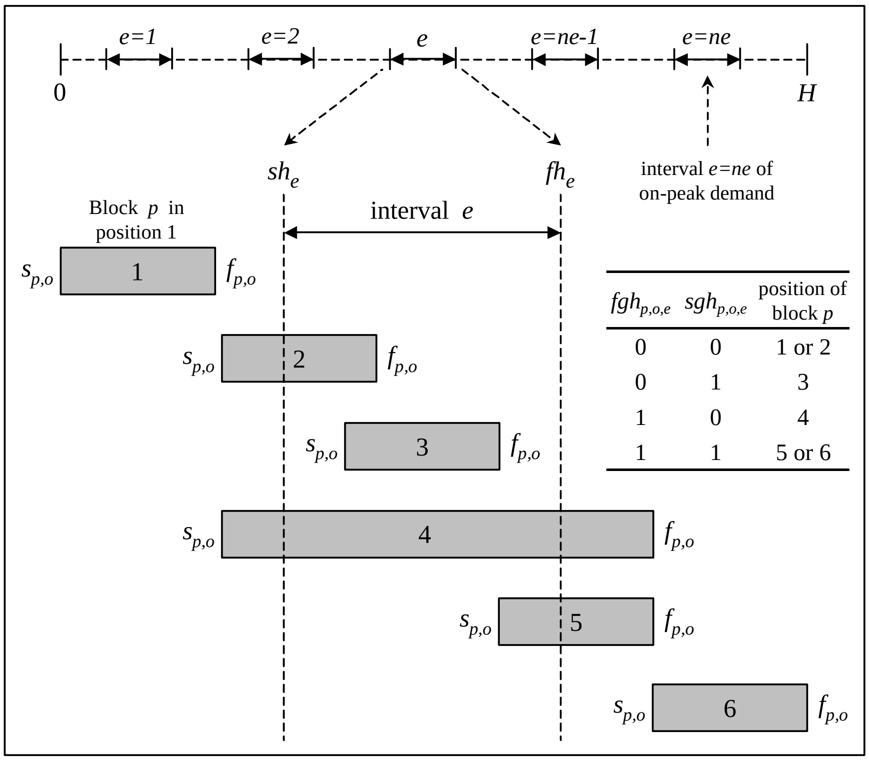
$$\textbf{If } fgh_{p,o,e} \textbf{ Then } f_{p,o} \geq fh_e \textbf{ Else } f_{p,o} < fh_e \quad \forall o \in O,\ p \in P_o,\ e \in E$$
$$\begin{cases} f_{p,o} - fh_e \geq L \cdot (1 - fgh_{p,o,e}) & \forall o \in O,\ p \in P_o,\ e \in E \\ f_{p,o} - fh_e \leq (U + \varepsilon) \cdot fgh_{p,o,e} - \varepsilon & \forall o \in O,\ p \in P_o,\ e \in E \end{cases} \tag{5.21}$$

$$\textbf{If } sgh_{p,o,e} \textbf{ Then } s_{p,o} \geq sh_e \textbf{ Else } s_{p,o} < sh_e \quad \forall o \in O,\ p \in P_o,\ e \in E$$
$$\begin{cases} s_{p,o} - sh_e \geq L \cdot (1 - sgh_{p,o,e}) & \forall o \in O,\ p \in P_o,\ e \in E \\ s_{p,o} - sh_e \leq (U + \varepsilon) \cdot sgh_{p,o,e} - \varepsilon & \forall o \in O,\ p \in P_o,\ e \in E \end{cases} \tag{5.22}$$

Figure 5.4 illustrates the six possible positions of a temporal block p in relation to an interval e. In the first position (1), block p totally precedes the interval e ($f_{p,o} < sh_e$). In the second position (2), block p starts before the interval e ($s_{p,o} < sh_e$), but it finishes inside e ($sh_e \leq f_{p,o} < fh_e$). In the third position (3), block p starts and finishes inside the interval e ($s_{p,o} \geq sh_e$ and $f_{p,o} < fh_e$). In the fourth position (4), block p starts before e ($s_{p,o} < sh_e$) and it finishes just after e ($f_{p,o} \geq fh_e$). In the fifth position (5), block p starts inside e ($sh_e \leq s_{p,o} < fh_e$), but it finishes after e ($f_{p,o} \geq fh_e$). In the sixth position (6), block p totally succeeds the interval e ($s_{p,o} \geq fh_e$). The duration of a temporal block p inside the electric cost interval e is determined by expressions 5.23 to 5.26, herein detailed.

The if-then statement 5.23 establishes that the truth of the logical expression $(\neg fgh_{p,o,e}) \wedge (\neg sgh_{p,o,e})$ implies that the inequality $inh_{p,o,e} \geq f_{p,o} - sh_e$ must hold. In the base-level of such if-then statement, $\delta^{00}_{p,o,e}$ assumes either values zero or one, according to the logical expression $(\neg fgh_{p,o,e}) \wedge (\neg sgh_{p,o,e})$. The truth of this logical expression indicates that a temporal block p is located in positions 1 or 2 of figure 5.4.

$$\textbf{If } (\neg fgh_{p,o,e}) \wedge (\neg sgh_{p,o,e}) \textbf{ Then } inh_{p,o,e} \geq f_{p,o} - sh_e \quad \forall o \in O,\ p \in P_o,\ e \in E$$
$$\begin{cases} \delta^{00}_{p,o,e} \leq 1 - fgh_{p,o,e} & \forall o \in O,\ p \in P_o,\ e \in E \\ \delta^{00}_{p,o,e} \leq 1 - sgh_{p,o,e} & \forall o \in O,\ p \in P_o,\ e \in E \\ \delta^{00}_{p,o,e} \geq 1 - fgh_{p,o,e} - sgh_{p,o,e} & \forall o \in O,\ p \in P_o,\ e \in E \\ inh_{p,o,e} - f_{p,o} + sh_e \geq L \cdot (1 - \delta^{00}_{p,o,e}) & \forall o \in O,\ p \in P_o,\ e \in E \end{cases} \tag{5.23}$$



$fgh_{p,o,e}$	$sgh_{p,o,e}$	position of block p
0	0	1 or 2
0	1	3
1	0	4
1	1	5 or 6

Figure 5.4: Electric Cost.

The if-then statement 5.24 establishes that the truth of the logical expression $(\neg fgh_{p,o,e}) \wedge (sgh_{p,o,e})$ implies that the equation $inh_{p,o,e} = f_{p,o} - s_{p,o}$ must hold. In the base-level of such if-then statement, $\delta^{01}_{p,o,e}$ assumes either values zero or one, according to the logical expression $(\neg fgh_{p,o,e}) \wedge (sgh_{p,o,e})$. The truth of this logical expression indicates that a temporal block p is located in position 3 of figure 5.4.

$$\textbf{If } (\neg fgh_{p,o,e}) \wedge (sgh_{p,o,e}) \textbf{ Then } inh_{p,o,e} = f_{p,o} - s_{p,o} \quad \forall o \in O,\ p \in P_o,\ e \in E$$

$$\begin{cases} \delta^{01}_{p,o,e} \leq 1 - fgh_{p,o,e} & \forall o \in O,\ p \in P_o,\ e \in E \\ \delta^{01}_{p,o,e} \leq sgh_{p,o,e} & \forall o \in O,\ p \in P_o,\ e \in E \\ \delta^{01}_{p,o,e} \geq sgh_{p,o,e} - fgh_{p,o,e} & \forall o \in O,\ p \in P_o,\ e \in E \\ inh_{p,o,e} - f_{p,o} + s_{p,o} \leq U \cdot (1 - \delta^{01}_{p,o,e}) & \forall o \in O,\ p \in P_o,\ e \in E \\ inh_{p,o,e} - f_{p,o} + s_{p,o} \geq L \cdot (1 - \delta^{01}_{p,o,e}) & \forall o \in O,\ p \in P_o,\ e \in E \end{cases} \tag{5.24}$$

The if-then statement 5.25 establishes that the truth of the logical expression $(fgh_{p,o,e}) \wedge (\neg sgh_{p,o,e})$ implies that the equation $inh_{p,o,e} = fh_e - sh_e$ must hold. In the base-level of such if-then statement, $\delta^{10}_{p,o,e}$ assumes either values zero or one, according to the logical expression $(fgh_{p,o,e}) \wedge (\neg sgh_{p,o,e})$. The truth of this logical expression indicates that a temporal block p is located in position 4 of figure 5.4.

If $(fgh_{p,o,e}) \wedge (\neg sgh_{p,o,e})$ **Then** $inh_{p,o,e} = fh_e - sh_e \quad \forall o \in O,\ p \in P_o,\ e \in E$

$$\begin{cases} \delta^{10}_{p,o,e} \leq fgh_{p,o,e} & \forall o \in O,\ p \in P_o,\ e \in E \\ \delta^{10}_{p,o,e} \leq 1 - sgh_{p,o,e} & \forall o \in O,\ p \in P_o,\ e \in E \\ \delta^{10}_{p,o,e} \geq fgh_{p,o,e} - sgh_{p,o,e} & \forall o \in O,\ p \in P_o,\ e \in E \\ inh_{p,o,e} - fh_e + sh_e \leq U \cdot (1 - \delta^{10}_{p,o,e}) & \forall o \in O,\ p \in P_o,\ e \in E \\ inh_{p,o,e} - fh_e + sh_e \geq L \cdot (1 - \delta^{10}_{p,o,e}) & \forall o \in O,\ p \in P_o,\ e \in E \end{cases} \tag{5.25}$$

The if-then statement 5.26 establishes that the truth of the logical expression $(fgh_{p,o,e}) \wedge (sgh_{p,o,e})$ implies that the inequality $inh_{p,o,e} \geq fh_e - s_{p,o}$ must hold. In the base-level of such if-then statement, $\delta^{11}_{p,o,e}$ assumes either values zero or one, according to the logical expression $(fgh_{p,o,e}) \wedge (sgh_{p,o,e})$. The truth of this logical expression indicates that a temporal block p is located in positions 5 or 6 of figure 5.4.

If $(fgh_{p,o,e}) \wedge (sgh_{p,o,e})$ **Then** $inh_{p,o,e} \geq fh_e - s_{p,o} \quad \forall o \in O,\ p \in P_o,\ e \in E$

$$\begin{cases} \delta^{11}_{p,o,e} \leq fgh_{p,o,e} & \forall o \in O,\ p \in P_o,\ e \in E \\ \delta^{11}_{p,o,e} \leq sgh_{p,o,e} & \forall o \in O,\ p \in P_o,\ e \in E \\ \delta^{11}_{p,o,e} \geq fgh_{p,o,e} + sgh_{p,o,e} - 1 & \forall o \in O,\ p \in P_o,\ e \in E \\ inh_{p,o,e} - fh_e + s_{p,o} \geq L \cdot (1 - \delta^{11}_{p,o,e}) & \forall o \in O,\ p \in P_o,\ e \in E \end{cases} \tag{5.26}$$

Inequality 5.27 assigns a bound to $stop_{p,o,e}$. Therefore, the time interval that a product p from origin o is maintained pressurized and idle during the on-peak demand interval e is limited by $inh_{p,o,e}$.

$$stop_{p,o,e} \leq inh_{p,o,e} \qquad \forall o \in O,\ p \in P_o,\ e \in E \tag{5.27}$$

Equation 5.28 indicates that the continuous variable $dpres$ has assigned the time period value that the pipeline remains pressurized and idle. It is interesting to notice that the pipe may remain pressurized and idle during on-peak intervals $(\sum_{o \in O} \sum_{p \in P_o} \sum_{e \in E} stop_{p,o,e} > 0)$ or during intervals that are charged at low rates $(H - \sum_{o \in O} \sum_{p \in P_o} (f_{p,o} - s_{p,o}) > 0)$.

$$dpres = H - \sum_{o \in O} \sum_{p \in P_o} (f_{p,o} - s_{p,o}) + \sum_{o \in O} \sum_{p \in P_o} \sum_{e \in E} stop_{p,o,e} \tag{5.28}$$

Inequalities 5.29 and 5.30 establish, respectively, the minimum and the maximum allowable pumping time interval of each product p from origin o.

$$f_{p,o} - s_{p,o} \geq d_{p,o}^{min} + \sum_{e \in E} stop_{p,o,e} + last_{p,o} \cdot td_{p,o}^{min} \qquad \forall o \in O,\ p \in P_o \qquad (5.29)$$

$$f_{p,o} - s_{p,o} \leq d_{p,o}^{max} + \sum_{e \in E} stop_{p,o,e} + last_{p,o} \cdot td_{p,o}^{max} \qquad \forall o \in O,\ p \in P_o \qquad (5.30)$$

Inequality 5.31 indicates that there must not be overlaps between temporal blocks.

$$f_{p,o} \leq s_{pa,\bar{o}} \qquad \forall p \in P_o,\ pa \in P_{\bar{o}},\ o = d,\ \bar{o} = \bar{d} \qquad (5.31)$$

Inequalities 5.32 to 5.35 establish that the temporal parameters determined by the auxiliary routine must be respected by the *Main Model*.

$$s_{p,o} \geq 0 \qquad \forall p \in P_o,\ o = d \qquad (5.32)$$

$$s_{pa,\bar{o}} \geq fd^{min} \qquad \forall pa \in P_{\bar{o}},\ \bar{o} = \bar{d} \qquad (5.33)$$

$$f_{p,o} \leq fd \qquad \forall p \in P_o,\ o = d \qquad (5.34)$$

$$f_{pa,\bar{o}} \leq H \qquad \forall pa \in P_{\bar{o}},\ \bar{o} = \bar{d} \qquad (5.35)$$

Thus, expressions 5.8 to 5.35 establish the *Main Model* mathematical formulation. This formulation is tailor-made for the unique features of the pipeline-scheduling problem previously described in section 5.2[12]. Nevertheless, the methodology applied to this formulation, which is described in section 5.3, can be adapted to other specific scheduling problems.

5.4.3.3 Main Model: The Number of Variables

According to Williams (1999), the number of (integer) variables in an (M)ILP model can be often regarded as an indicator of the computational difficulty for solving such model. The author deeply exploits the theme, and warns that, sometimes, the task of building an MILP model can just serve to show that the model demands

[12]In addition, other pipeline-scheduling features such as due-dates (or time-windows) for receiving products can be added to the formulation, according to operational necessities.

a prohibitive computational effort. Therefore, prior to applying MILP solution algorithms, it is advisable to estimate the number of variables presented by an MILP model. This can be an "indicator" of whether the generated model is likely to be solved by the state-of-the-art MILP solution strategies or not[13]. Besides the remarkable evolution of MILP solvers (see section 2.2 on page 13), it is recommended to be careful about MILP models that present, for instance, an exponential growth of variables. Thus, one important issue regarding the *Main Model* formulation is to determine its number of binary and continuous variables according to the considered problem instance.

Along the *Main Model* expressions 5.8 to 5.35, it can be observed that the subscripts of variables are products (p, pa), origins (o, $\bar{o}$) and electric cost (e). Hence, the number of binary and continuous variables are dependent upon these subscripts. By the problem definition (section 5.2), the number of different origins equals two (refinery and harbor). The number of demanded products in each origin (np_o) and the number of on-peak electric cost intervals during the scheduling horizon (ne) are input parameters, given according to the problem instance.

Equations 5.36 and 5.37 indicate, respectively, the total number of binary variables (nbv) and continuous variables (ncv) presented in the *Main Model*, as a function of np_o and ne. These equations were attained by observing the individual growth of each *Main Model* variable, according to different values of np_o and ne. For example, let's consider the generic variable $fgh_{p,o,e}$ in an illustrative case with $np_1 = 4$, $np_2 = 3$ and $ne = 2$. Therefore, the following derived variables should be created in the model: $fgh_{1,1,1}$, $fgh_{2,1,1}$, $fgh_{3,1,1}$, $fgh_{4,1,1}$, $fgh_{5,2,1}$, $fgh_{6,2,1}$, $fgh_{7,2,1}$, $fgh_{1,1,2}$, $fgh_{2,1,2}$, $fgh_{3,1,2}$, $fgh_{4,1,2}$, $fgh_{5,2,2}$, $fgh_{6,2,2}$, and $fgh_{7,2,2}$. Hence, it can be observed that the number of $fgh_{p,o,e}$ derived variables is equal to $ne \cdot \sum_{o \in O} np_o$. Following the same train of reasoning for all *Main Model* variables, equations 5.36 and 5.37 can be attained.

[13]In fact, it is very difficult to predict the computational burden of a particular MILP model. Often, large-scale constrained MILP models can be solved in a reasonable computational time (e.g. Kalvelagen, 2003; Magatão et al., 2004). Thus, the most reliable way to determine the computational burden of a particular MILP model seems to be through extensive simulations.

$$nbv = 2 \cdot \sum_{o \in O} np_o^2 + 2 \cdot ne \cdot \sum_{o \in O} np_o + \prod_{o \in O} np_o \tag{5.36}$$

$$ncv = 1 + (3 + 6 \cdot ne) \cdot \sum_{o \in O} np_o \tag{5.37}$$

In order to illustrate the use of equations 5.36 and 5.37, let's consider a hypothetical case with $np_o = 4 \ \forall o \in O$, $ne = 5$. Therefore, by equation 5.36, nbv is equal to $2 \cdot (4^2 + 4^2) + 2 \cdot 5 \cdot (4 + 4) + 4 \cdot 4$, that is, $nbv = 160$ (see table 5.3, page 105, column TNBV, $ne = 5$). By equation 5.37, ncv is equal to $1 + (3 + 6 \cdot 5) \cdot (4 + 4)$, that is, $ncv = 265$. Hence, the total number of variables, which is given by $nbv + ncv$, is equal to 425 (see table 5.3, column TNV, $ne = 5$).

Alternatively, equation 5.38 expresses in an algebraic form the MILP model total number of variables (tnv_{MILP}). Equation 5.38 is obtained by adding 5.36 to 5.37. The term $nbv + ncv$ was renamed tnv_{MILP}. Hence, if $np_o = 4 \ \forall o \in O$ and $ne = 5$ then tnv_{MILP} can be calculated as: $2 \cdot (4^2 + 4^2) + (3 + 8 \cdot 5) \cdot (4 + 4) + (4 \cdot 4) + 1$, that is, $tnv_{\text{MILP}} = 425$.

$$tnv_{\text{MILP}} = 2 \cdot \sum_{o \in O} np_o^2 + (3 + 8 \cdot ne) \cdot \sum_{o \in O} np_o + \prod_{o \in O} np_o + 1 \tag{5.38}$$

The considered problem involves the short-term-scheduling of activities in a real-world pipeline scenario. Thus, there is no sense in using a scheduling horizon greater than one week, because no complete information is known about the input data, mainly the required demands. In addition, the on-peak demand time intervals are considered to occur between 6 p.m. and 9 p.m., thus, in one week, $ne = 7$. Figure 5.5 illustrates the growth in the number of *Main Model* variables, according to the total number of demanded products. This figure was plotted based on equations 5.36 and 5.37 for $ne = 7$, $np_o = np_{\bar{o}}$ ($o = 1$, $\bar{o} = 2$), and $2 \leq \sum_{o \in O} np_o \leq 20$.

As stated in section 5.2, the pipeline conveys a limited set of oil products, which are demanded in batches. Typically, each batch demands a pumping time that ranges from many hours to an entire day. In addition, the pipeline operates completely filled, and an extra product amount should be pumped after the last *flow* and the last *reflow* sequenced products. In a typical operational condition,

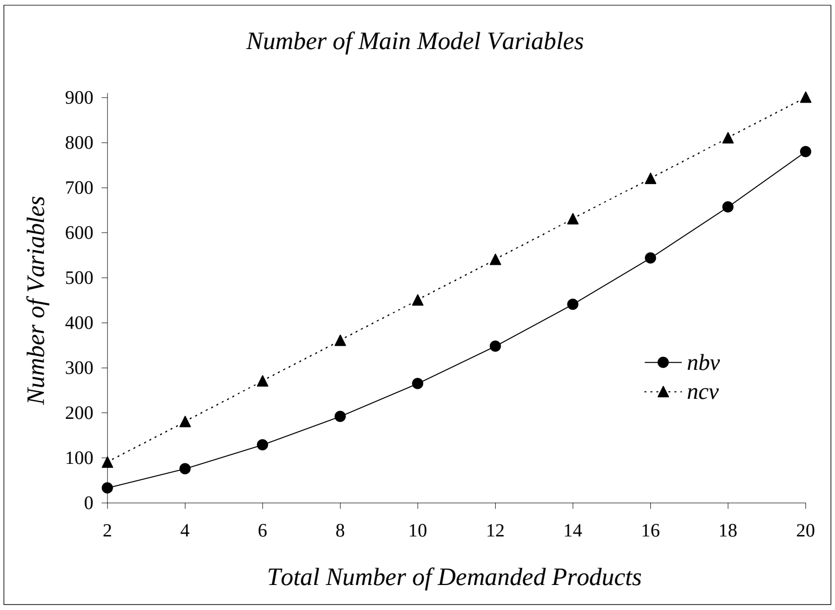


Figure 5.5: Number of Main Model Variables ($2 \leq \sum_{o \in O} np_o \leq 20$, $ne = 7$).

$\varphi_{p,o}^{max} = 300\,\mathrm{m}^3/\mathrm{h}\ \forall o \in O, p \in P_o$. Considering the pipeline volume as equal to $7314\,\mathrm{m}^3$ (pv), hence, 24 h are spent to fill up the pipeline after either *flow* or *reflow* operations: $\frac{7314\,\mathrm{m}^3}{300\,\mathrm{m}^3/\mathrm{h}} = 24.38 \approx 24\,\mathrm{h}$. A complete pumping operation covers both *flow* and *reflow* procedures. Therefore, 48 h are spent to fill up the pipeline after the last sequenced products. Thus, in one week (168 h), there exist, in fact, an available time of roughly 120 h to pump demanded products ($168 - 48$). Considering a hypothetical case, where each demanded batch takes six hours of pumping, thus, in one week, twenty different products (120/6) could be pumped. This is indeed a theoretical hypothesis, since the batches typically take more than six hours to be pumped in the real-world scenario. By equations 5.36 and 5.37, with $np_o = 10\ \forall o \in O$ ($\sum_{o \in O} np_o = 20$), then $nbv = 780$ and $ncv = 901$. The MILP literature[14] has pointed out the solution of far bigger MILP models in reasonable computational time (seconds to few minutes). Therefore, the use of this MILP model formulation for this specific pipeline-scheduling problem does not produce too large MILP models. Section 5.5, which considers a typical operational scenario of the real-world pipeline modeled, indicates that this assumption is indeed true (see table 5.3, page 105).

[14]See, for instance, Kalvelagen (2003).

5.5 Results

5.5.1 An Operational Scenario

In order to investigate the computational results presented by the mathematical model (section 5.4), a typical operational scenario of the real-world pipeline modeled is herein stated:

- An example involving the pumping of four products from the harbor to the refinery followed by other four pumped from the refinery to the harbor is henceforth considered, thus $np_1 = 4$, $np_2 = 4$, $d = 2$, and $\bar{d} = 1$.

- The illustrative example is simulated in a range of scheduling horizons. This range spans since the minimum scheduling horizon ($H^{min} = 114\,\text{h}$), which is determined by equation 5.6 of the *Auxiliary Routine*, up to 144 h.

- The parameters $C_o^{plug}\ \forall o \in O$, $C_o^{low}\ \forall o \in O$, and C^{pres} are considered unitary. In a first computational experiment, the value of $C_o^{high}\ \forall o \in O$ is set greater than the value of $C_o^{low}\ \forall o \in O$ by a factor of five (COPEL, 2005); in a second computational experiment, alternative C_o^{high} values are also considered ($C_o^{high} = 1$, 2, 3, 4, and 10 $\forall o \in O$).

- The electric energy on-peak demand was considered to occur during the following intervals: 18 h – 21 h, 42 h – 45 h, 66 h – 69 h, 90 h – 93 h, 114 h – 117 h, and 138 h – 141 h, as illustrated in figure 5.6. Therefore, in case $H \leq 114\,\text{h}$ then $ne = 4$; if $114\,\text{h} < H \leq 138\,\text{h}$ then $ne = 5$; if $138\,\text{h} < H \leq 144\,\text{h}$ then $ne = 6$. In particular, figure 5.6 indicates six on-peak demand intervals ($e = 1$, $e = 2, \ldots,$ $e = 6$).

- At the initial time the pipeline is filled with a product called *pin*, which requires no *plug* usage with the other products ($plug_{pin,p} = 0 \quad \forall o \in O,\ p \in P_o$).

- Table 5.2 presents input data for the considered scenario. In addition, it was considered that $\varphi_{p,o}^{min} = 150\,\text{m}^3/\text{h} \ \ \forall o \in O,\ p \in P_o$; $\varphi_{p,o}^{max} = 300\,\text{m}^3/\text{h}\ \forall o \in O$, $p \in P_o$; and $pv = 7200\,\text{m}^3$. Thus, $td_{p,o}^{min}$ can be calculated as $\frac{7200\,\text{m}^3}{300\,\text{m}^3/\text{h}} = 24\,\text{h}$

$(td^{min}_{p,o} = 24\,\mathrm{h}\ \forall o \in O,\ p \in P_o)$ and $td^{max}_{p,o}$ can be calculated as $\frac{7200\,\mathrm{m}^3}{150\,\mathrm{m}^3/\mathrm{h}} = 48\,\mathrm{h}$ $(td^{max}_{p,o} = 48\,\mathrm{h}\ \forall o \in O,\ p \in P_o)$, in accordance with equations 5.3 and 5.4 of the *Auxiliary Routine*[15].

- The non-dimensional auxiliary parameters U, L, and ε used to express the "base-level" of if-then and if-then-else statements (section 5.4.3) were considered, respectively, $+145$, -145, and 10^{-3}. As stated in chapter 4, U and L can be determined for each constraint (U_k and L_k). However, for simplicity, the illustrative instances henceforth mentioned adopted the fixed values ± 145 $(\pm(\text{maximum}(H) + 1))$. These values are attained by observing that the upper/lower bounds on all if-then and if-then-else "base-level" inequalities of section 5.4.3 can not exceed the scheduling horizon value[16].

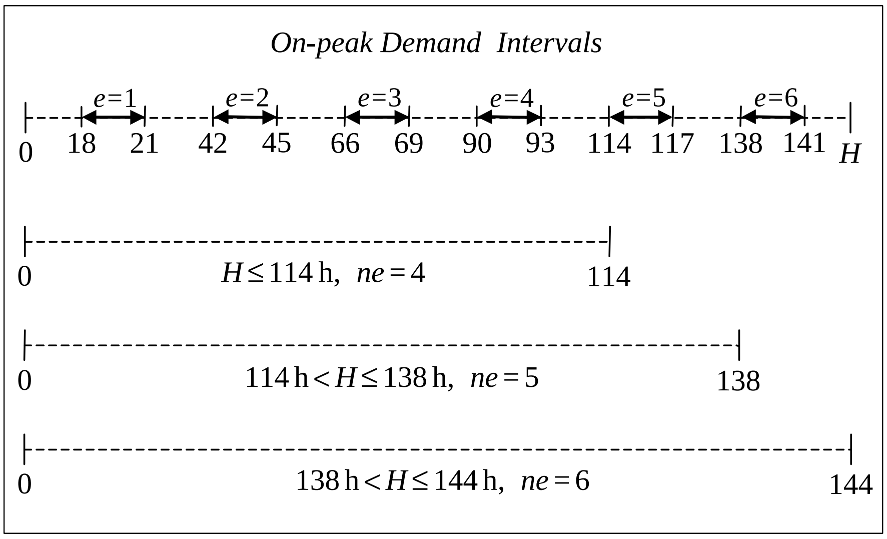


Figure 5.6: The *ne* Value.

In this chapter, the modeling and optimization tool Extended LINGO/PC Release 8.0 (LINDO, 2002) is used to implement and solve the optimization structure. LINGO is a commercial solver, which has its own modeling language and allows the

[15]For simplicity, an approximate value of pv was adopted ($7200\,\mathrm{m}^3$) instead of the exactly one ($7314\,\mathrm{m}^3$).

[16]In fact, the values of U and L are conservatives for some if-then statements (e.g statements 5.12, 5.16, and 5.18). In such cases, alternative values could be adopted (e.g. $U^{'} = +(\text{maximum}_{o \in O}(np_o) + 1)$ and $L^{'} = -(\text{maximum}_{o \in O}(np_o) + 1)$).

Table 5.2: Input Data for the Considered Scenario.

Operation	Product	$dem_{p,o}$ (m^3)	*Plug* Necessity ($plug_{p,pa}$)
Flow	$p1$	$dem_{1,1} = 1800$	$plug_{1,pa} = 1,\ pa = 2,4,6,8$
	$p2$	$dem_{2,1} = 3600$	$plug_{2,pa} = 1,\ pa = 1,5$
	$p3$	$dem_{3,1} = 1800$	$plug_{3,pa} = 1,\ pa = 4,8$
	$p4$	$dem_{4,1} = 1800$	$plug_{4,pa} = 1,\ pa = 1,3,5,7$
Reflow	$p5$	$dem_{5,2} = 1800$	$plug_{5,pa} = 1,\ pa = 2,4,6,8$
	$p6$	$dem_{6,2} = 1800$	$plug_{6,pa} = 1,\ pa = 1,5$
	$p7$	$dem_{7,2} = 3600$	$plug_{7,pa} = 1,\ pa = 4,8$
	$p8$	$dem_{8,2} = 3600$	$plug_{8,pa} = 1,\ pa = 1,3,5,7$

interface with databases. The tool can be used in the process of formulating linear and non-linear large models, solving them, and analyzing the solution. There is a series of algorithm settings to be defined in this MILP solver. These settings include, for example, the linear programming method (primal simplex, dual simplex, barrier), the branch-and-bound direction (up, down, both), the node selection (depth first, worst bound, best bound), the use of optimality margin, to name a few. Each of these settings can directly influence the search procedure. For an in-depth discussion, the interested reader is referred to LINDO (2002). Nevertheless, for simplicity, in the illustrative examples herein presented, these settings are maintained in the default option of LINGO[17], in exception of the available solver memory, which is altered from 32 Mbyte to 512 Mbyte.

5.5.2 Operational Cost versus Scheduling Horizon

5.5.2.1 First Computational Experiment

Table 5.3 (page 105) provides information about the optimization structure simulation on a Pentium 4, 2.4 GHz, 1 Gbyte RAM. For each scheduling horizon (H), the optimization structure is run, and a specific cost is attained (value of expression 5.8). The *Auxiliary Routine* and the *Tank Bound* simulation data are

[17]For a detailed description of LINGO's default option see LINDO (2002).

neglected. These structures required a computational time lower than one second, for all illustrative instances ($114\,\text{h} \leq H \leq 144\,\text{h}$). No uncertainties are considered. The model is solved for fixed product demands. In table 5.3, H is the scheduling horizon, ne is the number of on-peak electric time intervals, TNV stands for the total number of variables (tnv_{MILP} - see equation 5.38), TNBV stands for the total number of binary variables (see equation 5.36), TNC stands for the total number of constraints, CT stands for the computational time spent to solve the model to optimality[18], BRANCHES indicates the total number of branches visited by the branch-and-bound algorithm, IT stands for the total number of iterations, and *Cost* indicates the operational cost (value of expression 5.8).

Magatão et al. (2004) previously addressed the same pipeline-scheduling problem by a discrete time formulation (see section 5.3). The authors reported a hypothetical instance involving the pumping of four products from the harbor to the refinery followed by other four pumped from the refinery to the harbor. The *re-flow/flow* procedures were covered in 120 h, using a discretized time interval of 1 h. This instance gave rise to a discrete MILP *Main Model* with 3253 variables, 782 binary variables, and 8896 constraints. This large scale MILP was solved to optimality in 1375 s in a Pentium 4, 1.8 GHz, 512 Mbyte RAM running the Extended LINGO/PC Release 8.0 (LINDO, 2002).

Figure 5.7 is based upon table 5.3 data, and it indicates the computational time presented by the continuous formulation as a function of the available scheduling horizon. One can observe that such computational time ranged from seconds to few minutes[19]. According to table 5.3, the continuous time approach tends to produce smaller formulations than the discrete time model presented by Magatão et al. (2004). These smaller formulations are likely to be solved faster than the large scale discrete MILP model, as indicated in figure 5.7.

Figure 5.8 is based upon table 5.3 data, and it shows the operational cost (expression 5.8 value) as a time horizon function. Figure 5.8 highlights the existence

[18]For all simulation instances, near optimal integer solutions were attained before 20 seconds of processing, but the branch-and-bound algorithm was run up to the optimal solution determination.

[19]In the worst case the computational time was $\approx$ 7.5 minutes (449 seconds); in average it was $\approx$ 2.9 minutes (171 seconds).

Table 5.3: Computational Data for the Main Model Illustrative Instances.

H (h)	*ne*	TNV	TNBV	TNC	CT (s)	BRANCHES	IT ($\times 10^6$)	*Cost* ($)
114	4	361	144	1132	107	12506	0.66	**174**
115	5	425	160	1324	148	17756	0.89	175
116	5	425	160	1324	181	22292	1.09	176
117	5	425	160	1324	319	28850	1.92	**177**
118	5	425	160	1324	284	37202	1.64	174
119	5	425	160	1324	30	3454	0.18	171
120	5	425	160	1324	314	43690	1.64	168
121	5	425	160	1324	336	47727	1.62	164
122	5	425	160	1324	322	41539	1.67	160
123	5	425	160	1324	309	37375	1.41	156
124	5	425	160	1324	252	30943	1.20	153
124.5	5	425	160	1324	266	37528	1.22	151.5
125	5	425	160	1324	115	13401	0.60	150
125.5	5	425	160	1324	282	38644	1.29	148.5
126	5	425	160	1324	441	54322	1.80	147
126.5	5	425	160	1324	420	50939	1.74	145.5
127	5	425	160	1324	**449**	53508	1.96	144
127.5	5	425	160	1324	389	49060	1.61	142.5
128	5	425	160	1324	356	42084	1.52	141
128.25	5	425	160	1324	154	15392	0.69	138.375
128.5	5	425	160	1324	177	19602	0.79	139.5
128.75	5	425	160	1324	177	17855	0.76	138.75
129	5	425	160	1324	117	10301	0.55	**138**
129.25	5	425	160	1324	141	12689	0.71	138.25
129.5	5	425	160	1324	142	12744	0.68	138.5
129.75	5	425	160	1324	185	17895	0.87	138.75
130	5	425	160	1324	137	12279	0.66	139
130.5	5	425	160	1324	237	23204	1.18	139.5
131	5	425	160	1324	119	9024	0.63	140
131.5	5	425	160	1324	130	9546	0.66	140.5
132	5	425	160	1324	97	7623	0.48	139
132.5	5	425	160	1324	88	6983	0.44	139.5
133	5	425	160	1324	66	4214	0.37	140
133.5	5	425	160	1324	72	3884	0.38	140.5
134	5	425	160	1324	104	11641	0.50	141
134.5	5	425	160	1324	62	5299	0.33	141.5
135	5	425	160	1324	82	6368	0.42	142
135.5	5	425	160	1324	76	4381	0.41	142.5
136	5	425	160	1324	73	6496	0.39	143
136.5	5	425	160	1324	76	4173	0.43	143.5
137	5	425	160	1324	74	3691	0.42	144
137.5	5	425	160	1324	90	6189	0.47	144.5
138	5	425	160	1324	80	6211	0.41	144
138.5	6	489	176	1516	64	6609	0.29	145.5
139	6	489	176	1516	89	6831	0.44	145
140	6	489	176	1516	66	4340	0.36	146
141	6	489	176	1516	81	5447	0.45	147
142	6	489	176	1516	55	2373	0.30	148
143	6	489	176	1516	55	3479	0.29	149
144	6	489	176	1516	89	8931	0.43	150

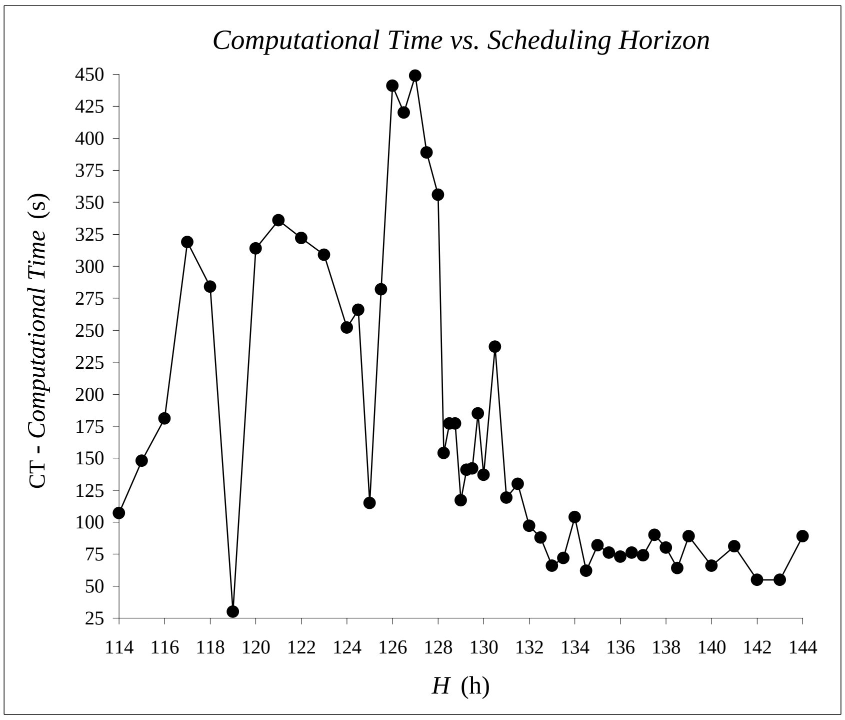


Figure 5.7: Computational Time versus Scheduling Horizon.

of specific scheduling horizons (e.g. $128\,\mathrm{h} \leq H \leq 130\,\mathrm{h}$) that yield low operational costs. **The cost versus scheduling horizon function clearly demonstrates that a correct pipeline operation can provide significant cost saving.**

Commentaries on Figure 5.8: Figure 5.8 demonstrates that in $H = H^{min} = 114\,\mathrm{h}$ the operational cost is equal to 174 normalized cost units ($Cost = 174\,\$$). This cost reflects a pumping procedure where *plug* inclusions were minimized[20], and each demanded product was pumped at its maximum flow rate, with no interruptions (e.g. $stop_{p,o,e} = 0\ \forall o \in O,\ p \in P_o,\ e \in E$). Therefore, products were indeed pumped during the on-peak demand intervals $18\,\mathrm{h} < H \leq 21\,\mathrm{h}$, $42\,\mathrm{h} < H \leq 45\,\mathrm{h}$, $66\,\mathrm{h} < H \leq 69\,\mathrm{h}$, and $90\,\mathrm{h} < H \leq 93\,\mathrm{h}$, even if the procedure of pumping products during such intervals causes an increase in the operational cost value ($C_o^{high} = 5\ \forall o \in O$, whereas $C_o^{low} = 1\ \forall o \in O$). However, for $114\,\mathrm{h} < H \leq 144\,\mathrm{h}$ there exists an available scheduling horizon greater than the minimum scheduling horizon ($H > H^{min}$), and

[20]In fact, for all illustrative instances ($114\,\mathrm{h} \leq H \leq 144\,\mathrm{h}$), the optimization structure determined pumping sequences where no *plugs* were used. Further details are given in section 5.5.3, page 112.

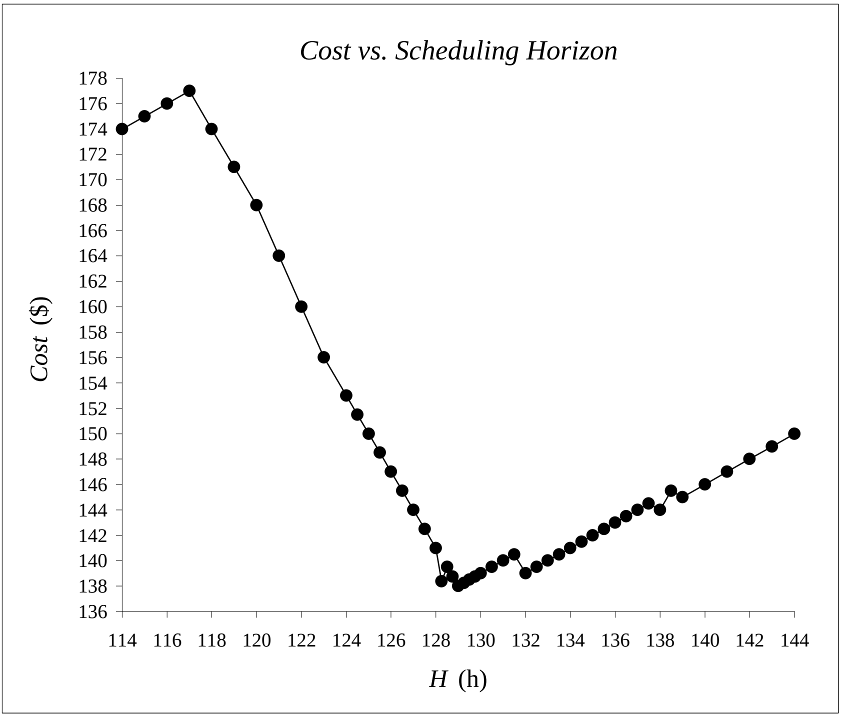


Figure 5.8: Cost versus Scheduling Horizon ($C_o^{high} = 5\ \forall o \in O$).

the optimization structure can manage this "spare time" $(H - H^{min})$ in order to achieve low cost operational procedures. The main subtlety is to stop pumping during on-peak demand intervals, as far as possible.

Moreover, figure 5.8 indicates that during the interval $114\,\mathrm{h} < H \leq 117\,\mathrm{h}$ the operational cost function increased up to the normalized value 177 ($Cost = 177\,\$$). In fact, for $114\,\mathrm{h} < H \leq 117\,\mathrm{h}$ there exists another on-peak demand (see figure 5.6 on page 102, interval $e = 5$). For any scheduling horizon inside the period $114\,\mathrm{h} < H \leq 117\,\mathrm{h}$, the "spare time" is charged at high electric rates, and pumping products inside this period turns out being too costly. Then, the pipeline is just maintained pressurized and idle, which increased the operational cost ($Cost = 175$, 176, $177\,\$$), but was the most economical procedure. However, right after $H = 117\,\mathrm{h}$, the cost function presents a turning point, and it started to decrease. In order to justify this cost reduction, let the scheduling horizon be 118 h. Therefore, the interval $117\,\mathrm{h} < H \leq 118\,\mathrm{h}$ has a low electric cost rate, and pumping products at this one-hour period is economically more advantageous than pumping products at any previous on-peak demand interval (see figure 5.6, intervals $e = 1$, $e = 2, \ldots,$ $e = 5$). So that, the most economical decision is to stop pumping during one hour of a previous on-peak demand interval, and continuing pumping during the low rate period $117\,\text{h} < H \leq 118\,\text{h}$. In case of $H = 119\,\text{h}$, there exists a two-hour interval with a low electric cost rate ($117\,\text{h} < H \leq 119\,\text{h}$). Without loss of generality, each scheduling horizon, such that $H > H^{min}$, can potentially have a "spare time" with a low electric cost rate. Furthermore, maintaining the pipeline pressurized and idle during on-peak demand intervals and pumping during low electric cost rate intervals is a heuristic, and figure 5.8 indicates a decreasing tendency on the cost function ($117\,\text{h} < H \leq 129\,\text{h}$) that is originated from pumping during low rate intervals instead of on-peak intervals.

For $H > 129\,\text{h}$, figure 5.8 indicates an average increasing tendency on the objective function value. In fact, a scheduling horizon of 129 h provides enough time for both:

(i) Maintain the pipeline pressurized and idle during all the five previous on-peak demand intervals (see figure 5.6, intervals $e{=}1$, $e{=}2, \ldots, e{=}5$);

(ii) Pump each product during electric cost intervals that are charged at low rates ($0\,\text{h} \leq H \leq 18\,\text{h}$, $21\,\text{h} < H \leq 42\,\text{h}$, $45\,\text{h} < H \leq 66\,\text{h}$, $69\,\text{h} < H \leq 90\,\text{h}$, $93\,\text{h} < H \leq 114\,\text{h}$, and $117\,\text{h} < H \leq 129\,\text{h}$). In this case, each product is pumped at its maximum flow rate during the low cost intervals.

As it can be observed in the objective function (expression 5.8, page 89), the time period that is spent to pump a product is charged at $\sum_{o \in O} \sum_{p \in P_o} C_o^{low} \cdot (f_{p,o} - s_{p,o})$; so that, to spread out a product pumping for too long may not be a wise procedure. On the other hand, if all products are pumped before H, the pipeline must remain pressurized and idle, and this condition also influences the operational cost ($C^{pres} \cdot dpres$). As the objective function is weighed by a series of factors, too long scheduling horizons may be inadequate, and just cause cost increases (e.g. $H > 129\,\text{h}$). Even the opportunity of stop pumping during the on-peak interval $138\,\text{h} < H \leq 141\,\text{h}$ did not cause a significant overall cost reduction, as it can be observed in figure 5.8.

5.5.2.2 Second Computational Experiment

Table 5.3 reports that the normalized cost value ranged from 138 units to 177 units, according to the available scheduling horizon. Therefore, table 5.3 indicates a potential cost variation, and, in particular, figure 5.8 highlights this fact. However, one must notice that C_o^{high}, C_o^{low}, C_o^{plug}, and C^{pres}, which are parameters that weigh the objective function (expression 5.8), directly influence the cost value. Variations on such parameters can cause changes in the objective function value. In the first computational experiment of section 5.5.2, these parameters were considered to be equal to one, in exception of C_o^{high} ($C_o^{high} = 5\ \forall o \in O$).

This computational experiment further investigates the C_o^{high} influence on the operational cost value. For this task, the same pipeline-scheduling model, with the input data stated in section 5.5.1 (e.g. $114\,\text{h} \leq H \leq 144\,\text{h}$, $C_o^{low} = 1\ \forall o \in O$, $C_o^{plug} = 1\ \forall o \in O$, $C^{pres} = 1$), is (re)simulated, but alternative C_o^{high} values are herein also considered ($C_o^{high} = 1$, 2, 3, 4, and 10 $\forall o \in O$). Table 5.4 indicates the obtained operational cost values.

In table 5.4, H is the scheduling horizon, ne is the number of on-peak electric time intervals, TNV stands for the total number of variables (tnv_{MILP} - see equation 5.38), TNC stands for the total number of constraints, CT stands for the computational time (in seconds)[21], and *Cost* indicates the normalized operational cost (expression 5.8 value), according to C_o^{high} variations.

Figure 5.9, which is based on table 5.4 data, evidences the obtained operational cost functions for $114\,\text{h} \leq H \leq 144$ and $C_o^{high} = 1$, 2, 3, 4, 5, 10 $\forall o \in O$.

Commentaries on Figure 5.9: Figure 5.9 indicates that the operational cost value is influenced by C_o^{high} variations. In case of $C_o^{high} = 1\ \forall o \in O$, the generated cost function presents a minimum value ($Cost = 126\,\$$) at $H = H^{min} = 114\,\text{h}$ (details on numerical cost values are given in table 5.4); for $114\,\text{h} < H \leq 144\,\text{h}$, the operational cost has an increasing tendency. In contrast, for $C_o^{high} = 2$, 3, 4, 5, 10 $\forall o \in O$, the

[21]In table 5.4, the average values of CT for $C_o^{high} = 1$, 2, 3, 4, 5, and 10 were, respectively, 95 s, 153 s, 176 s, 179 s, 172 s, and 181 s.

Table 5.4: Computational Data for the Main Model (C_o^{high} = 1, 2, 3, 4, 5, 10 $\forall o \in O$).

				$C_o^{high}=1$		$C_o^{high}=2$		$C_o^{high}=3$		$C_o^{high}=4$		$C_o^{high}=5$		$C_o^{high}=10$	
H	ne	TNV	TNC	CT	Cost	CT	Cost	CT	Cost	CT	Cost	CT	Cost	CT	Cost
(h)				(s)	($)	(s)	($)	(s)	($)	(s)	($)	(s)	($)	(s)	($)
114	4	361	1132	87	**126**	221	**138**	171	**150**	102	**162**	107	**174**	213	**234**
115	5	425	1324	97	127	196	139	146	151	139	163	148	175	216	235
116	5	425	1324	97	128	138	140	211	152	186	164	181	176	124	236
117	5	425	1324	125	129	150	141	166	153	179	165	319	177	182	237
118	5	425	1324	112	130	256	141	278	152	226	163	284	174	213	229
119	5	425	1324	146	131	321	141	269	151	296	161	30	171	250	221
120	5	425	1324	143	132	208	141	259	150	275	159	314	168	45	214
121	5	425	1324	167	132	360	140	300	148	300	156	336	164	282	204
122	5	425	1324	128	132	292	139	364	146	342	153	322	160	336	195
123	5	425	1324	77	132	233	138	345	144	284	150	309	156	442	186
124	5	425	1324	94	133	183	138	362	143	334	148	252	153	345	178
125	5	425	1324	98	134	182	138	228	142	243	146	115	150	332	170
126	5	425	1324	133	135	237	138	392	141	460	144	441	147	469	162
127	5	425	1324	186	136	196	138	286	140	447	142	449	144	313	154
128	5	425	1324	104	136	183	138	214	139	313	140	356	141	388	146
129	5	425	1324	95	136	176	138	202	138	118	138	117	138	185	138
130	5	425	1324	89	137	118	139	125	139	131	139	137	139	113	139
131	5	425	1324	76	138	111	139	93	140	221	140	119	140	105	140
132	5	425	1324	91	139	109	139	85	139	87	139	97	139	71	139
133	5	425	1324	90	140	64	140	81	140	86	140	66	140	96	140
134	5	425	1324	84	141	121	141	87	141	68	141	104	141	92	141
135	5	425	1324	66	141	96	142	82	142	76	142	82	142	84	142
136	5	425	1324	70	142	65	143	91	143	61	143	73	143	89	143
137	5	425	1324	59	143	82	144	96	144	80	144	74	144	94	144
138	5	425	1324	49	144	69	144	86	144	58	144	80	144	73	144
139	6	489	1516	55	145	64	145	35	145	53	145	89	145	74	145
140	6	489	1516	51	146	57	146	80	146	103	146	66	146	65	146
141	6	489	1516	77	147	66	147	66	147	65	147	81	147	85	147
142	6	489	1516	74	148	69	148	78	148	76	148	55	148	74	148
143	6	489	1516	56	149	66	149	78	149	65	149	55	149	80	149
144	6	489	1516	75	150	68	150	90	150	85	150	89	150	89	150

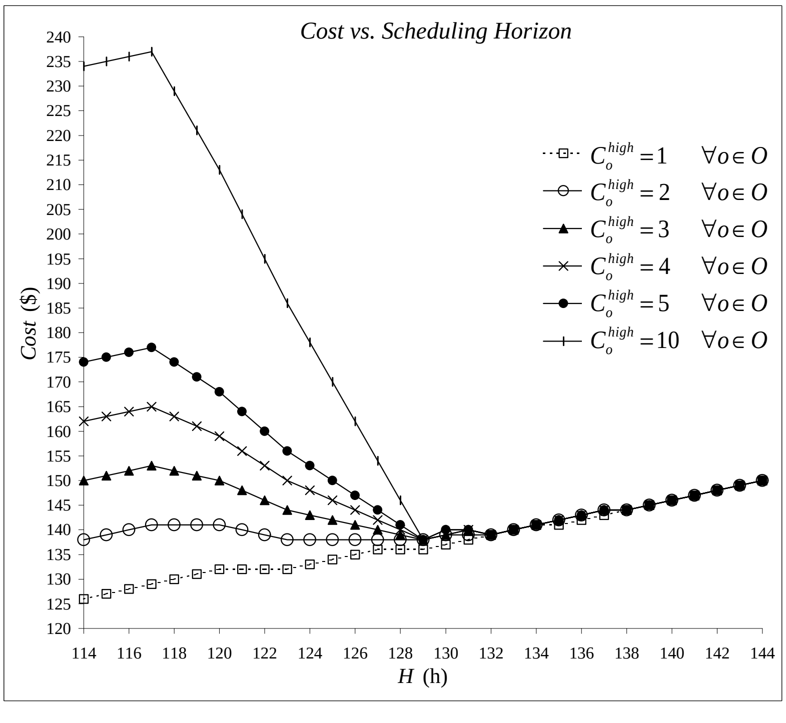


Figure 5.9: Cost versus Scheduling Horizon (C_o^{high} = 1, 2, 3, 4, 5, 10 $\forall o \in O$).

generated cost functions present minimum operational cost points[22] for $H > H^{min}$. This decreasing cost tendency is directly related with the fact that the optimization structure indicates scheduling solutions where the pipeline is stopped during high electric cost intervals (commentaries on figure 5.8, page 106, further explained this fact). In addition, figure 5.9 demonstrates that the interval $128\,\mathrm{h} \leq H \leq 130\,\mathrm{h}$ represents a low operational cost region, at least for $C_o^{high} = 2$, 3, 4, 5, and 10 $\forall o \in O$.

Moreover, figure 5.9 reinforces a remark about too long scheduling horizons, previously discussed in commentaries of figure 5.8: such scheduling horizons can be inadequate and just cause cost increases (e.g. $H > 129\,\mathrm{h}$). In particular, figure 5.9 highlights that the six tested C_o^{high} values ($C_o^{high} = 1$, 2, 3, 4, 5, 10 $\forall o \in O$) yield (almost) the same cost value for $H > 129\,\mathrm{h}$. As previously stated, a scheduling horizon of 129 h provides enough time for both:

(i) Maintain the pipeline pressurized and idle during all the five previous on-peak demand intervals (see figure 5.6, intervals $e = 1$, $e = 2, \ldots, e = 5$);

(ii) Pump each product during electric cost intervals that are charged at low rates ($0\,\mathrm{h} \leq H \leq 18\,\mathrm{h}$, $21\,\mathrm{h} < H \leq 42\,\mathrm{h}$, $45\,\mathrm{h} < H \leq 66\,\mathrm{h}$, $69\,\mathrm{h} < H \leq 90\,\mathrm{h}$, $93\,\mathrm{h} < H \leq 114\,\mathrm{h}$, and $117\,\mathrm{h} < H \leq 129\,\mathrm{h}$). In this case, each product is pumped at its maximum flow rate during the low cost intervals.

Thus, for $H > 129\,\mathrm{h}$, the most economical procedure evidenced during simulations was to pump the demanded products at the available low electric cost rate intervals, and to maintain the pipeline pressurized and idle during the remaining time. Therefore, in a practical standpoint, no products were pumped during high electric cost intervals, and the C_o^{high} differences did not alter the overall cost value. As a consequence, the cost functions were "equalized" for $H > 129\,\mathrm{h}$, $C_o^{high} = 1$, 2, 3, 4, 5, and 10 $\forall o \in O$, as evidenced in figure 5.9.

Furthermore, figure 5.9 indicates that a change in a factor that weighs the objective function can decisively alters the operational cost value, even if one considers the same scheduling horizon value. For example, let the scheduling horizon

[22] In fact, $C_o^{high} = 2$ also has a minimum operational cost point ($Cost = 138\,\$$) at $H = H^{min}$.

be $H = H^{min} = 114\,\text{h}$, then the cost values for $C_o^{high} = 1, 2, 3, 4, 5$, and $10\ \forall o \in O$ are, respectively, 126, 138, 150, 162, 174, and 234 normalized units (details on numerical cost values are given in table 5.4). Therefore, an accurate determination of C_o^{high}, C_o^{low}, C_o^{plug}, C^{pres} values is advisable in order to obtain a realistic cost function.

5.5.3 An Illustrative Case of Pumping Procedure

Figure 5.10 illustrates the pumping operation predicted by the optimization structure for $H = 129\,\text{h}$. In addition, figure 5.11 indicates the pipeline flow rate behavior during such pumping operation. By figure 5.10, the determined pumping sequence is $p8$, $p6$, $p7$, and $p5$ (*reflow* procedure); $p1$, $p3$, $p2$, and $p4$ (*flow* procedure). According to table 5.2, this pumping sequence implies no use of *plugs*[23].

The pumping process starts at *time* equal to zero, with the *reflow* procedure. At *time* equal to 12 the entire demanded amount of $p8$ is inside the pipe, but the refinery just receives it later. Consequently, there is a time in between sending a product and receiving it. During the interval $18 < time \leq 21$ the pipeline remains pressurized and idle, as indicated in figure 5.11. The condition of maintaining the pipe pressurized and idle also occurs during the intervals $42 < time \leq 45$, $66 < time \leq 69$, $90 < time \leq 93$, and $114 < time \leq 117$.

At *time* equal to 66 the refinery has already received the products from the harbor ($p8$, $p6$, $p7$, and $p5$), and an extra product amount of $p5$ has filled the pipeline. Then, the *flow* procedure is started with the pumping of $p1$ (in fact, the pumping is just started right after 69 time units). Set-up times are neglected. At *time* equal to 129 the harbor has already received the products from the refinery ($p1$, $p3$, $p2$, and $p4$), and an extra product amount of $p4$ has filled the pipeline.

According to figure 5.11, the pipeline remained pressurized and idle during the on-peak demand intervals $e = 1$, $e = 2, \ldots$, $e = 5$, indicated in figure 5.6 (page 102). Furthermore, after the completion of pumping procedures ($H > 129\,\text{h}$) the pipeline also remained pressurized and idle.

[23] $plug_{pin,8} = plug_{8,6} = plug_{6,7} = plug_{7,5} = plug_{5,1} = plug_{1,3} = plug_{3,2} = plug_{2,4} = 0.$

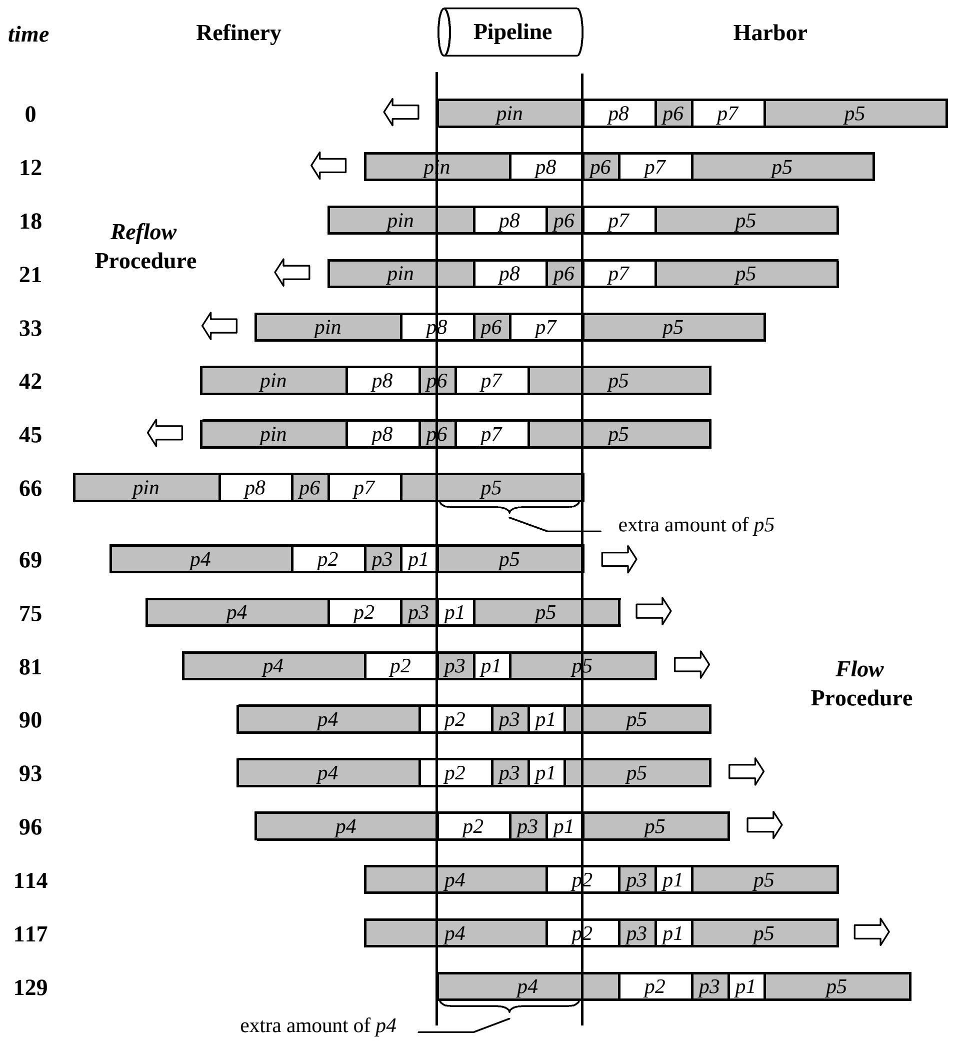


Figure 5.10: Pumping Procedure: An Illustrative Case.

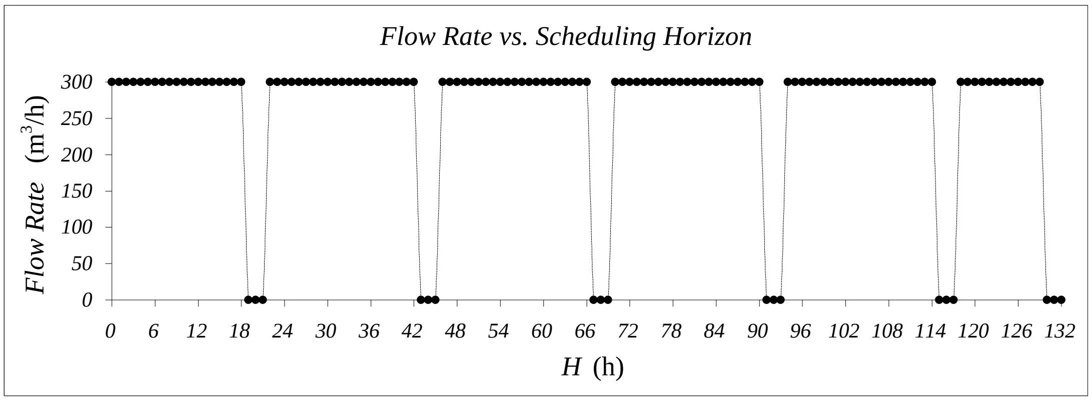


Figure 5.11: Pumping Procedure: The Flow Rate Behavior.

5.6 Remarks on Chapter 5

This chapter addressed the problem of developing an optimization structure to aid the operational decision-making of scheduling activities in a real-world pipeline scenario. The pipeline connects an inland refinery to a harbor, conveying different types of oil products. The optimization structure was developed based on mixed integer linear programming (MILP) with continuous time approach. The MILP well-known computational burden was avoided by the proposed decomposition strategy (see figure 5.2). Illustrative instances have demonstrated that the optimization structure is able to define new operational points to the pipeline system, providing significant cost saving (see figure 5.8).

The main goal of chapter 5 was to demonstrate the applicability of the "high-level" MILP modeling structures presented in chapter 4. Therefore, the MILP model formulation presented in section 5.4.3 extensively uses such high-level structures, in particular, the if-then and the if-then-else "high-level" statements. In section 5.4.3, some "base-level" MILP expressions derived from if-then and if-then-else "high-level" statements are presented. It is difficult to consider the direct modeling of the base-level expressions without the if-then and the if-then-else high-level statements. This would be certainly a harder task. Therefore, the if-then and the if-then-else "high-level" structures decisively aid the MILP model formulation. Moreover, the author of this thesis has already addressed the same pipeline-scheduling problem by the traditional inequality-based approach (Magatão, 2001), and, in a practical standpoint, the high-level structures use make the formulation task simpler and more straightforward.

Chapter 6

CLP-MILP: Towards a Unified Modeling Framework

This chapter presents a CLP-MILP formulation addressing the same problem exploited in chapter 5. The main goal is to demonstrate the applicability of the high-level MILP modeling structures developed in chapter 4 in an integrated CLP-MILP modeling framework.

6.1 Modeling Premises

This chapter addresses the same pipeline-scheduling problem studied in chapter 5, and some concepts previously presented are henceforth also used. For the sake of a better understanding, the reader is advised to review the following sections of chapter 5 that are presupposed by chapter 6:

- Sections 5.1 (page 75) and 5.2 (page 77), which introduce the main features about the problem description;

- Section 5.3 (page 78), which describes the methodology adopted to deal with the problem. In particular, this methodology is also used in chapter 6, in exception that the *Main Model* (see figure 5.2 on page 79) is developed based on different approaches than MILP. Therefore, the parameters previously determined by the *Tank Bound* (Magatão et al., 2004) and the *Auxiliary Routine* (see equations 5.1 to 5.7 on page 87) should be respected;

- Section 5.4 (page 81), which presents the general guidelines used to create the mathematical formulation, as well as the notation adopted (section 5.4.1). These guidelines and notation are also observed in chapter 6.

In addition to the modeling guidelines inherited from chapter 5, it has to be observed that a combined CLP-MILP formulation demands that CLP and MILP models be developed. At this particular case, the MILP part was already addressed in chapter 5. In section 6.2, a CLP model is developed. Subsequently, in section 6.3, a combined CLP-MILP approach is demonstrated. The functionalities of the commercial tool ILOG OPL Studio 3.6.1 (ILOG, 2002b) are used in this chapter for the investigations involving a combined CLP-MILP approach.

OPL Studio is an integrated development environment for mathematical programming and combinatorial optimization problems. The development of such environment has its basis on modeling languages such as AMPL (Fourer et al., 1990), GAMS (Brooke and Meeraus, 1982), and LINGO (LINDO, 2002), which provide computer code representations to traditional algebraic notation. OPL, which stands for Optimization Programming Language, provides similar support for modeling MILP models, and it gives access to state-of-the-art linear/integer programming algorithms. Furthermore, beyond the traditional support for LP and MILP, OPL is an attempt to combine the strengths of mathematical programming languages and constraint logic programming. It aims at both, increasing the applicability of modeling languages by incorporating techniques from CLP, and improving the expressiveness of traditional CLP tools by borrowing ideas from modeling languages. Another OPL's functionality is its high-level support for scheduling and resource allocation problems, which are ubiquitous in industry. An in-depth discussion about OPL's functionalities can be found in ILOG (2002b). In addition, the OPL language details are specified in ILOG (2002a). The models henceforth presented in this chapter are implemented and tested with the ILOG OPL Studio 3.6.1 functionalities.

6.2 Building a CLP Model

The main goal of chapter 5 was to demonstrate the applicability of the "high-level" MILP modeling structures presented in chapter 4. Therefore, the MILP model formulation presented in section 5.4.3 extensively uses such high-level structures, in particular, the if-then and the if-then-else statements. These high-level structures were tailor-made to aid the MILP modeling process, which relies on an inequality-based vocabulary. On the other hand, CLP is known by its rich modeling framework. In particular, the if-then and the if-then-else statements used in section 5.4.3, which demanded the "base-level" MILP expressions, can be directly modeled in a CLP framework. Therefore, there is no necessity of such base-level expressions in the equivalent CLP formulation. For example, the if-then-else statement 5.9 (page 90) can be represented by the equivalence $gt_{p,pa,o} = 1 \leftrightarrow f_{p,o} \leq s_{pa,o}\ \forall o \in O,\ p \neq pa \in P_o$. Equivalences and implications are ordinary features in a CLP framework, and the if-then and the if-then-else statements presented in section 5.4.3 are replaced by, respectively, implications ($\rightarrow$) and equivalences ($\leftrightarrow$) in the CLP approach. Further details are given in section 6.2.2[1].

Another important characteristic of a CLP framework is the availability of specialized algorithms to deal with specific classes of problems, such as scheduling problems. In order to use specialized scheduling algorithms at this CLP model, the former definition of variables $f_{p,o}$ and $s_{p,o}$ (see section 5.4.1 on page 82) was incorporated into one of the most fundamental concepts for scheduling applications presented by the OPL tool: the *activity*. An activity can be thought of as an object containing three items: a starting date, a duration, and an ending date, together with the duration constraint. The duration constraint states that the ending date of an activity is its starting date plus its duration (ILOG, 2002a). Therefore, in this CLP model, a new variable called $pump_{p,o}\ \forall o \in O,\ p \in P_o$ is declared as having the type *activity*. Thus, the pumping process starts at time $pump.start_{p,o}$ (abbreviated by $p.s_{p,o}$), it finishes at time $pump.end_{p,o}$ (abbreviated by $p.e_{p,o}$), and it has duration

[1]Chapter 4 brings an in-depth discussion about the correspondence of if-then-else statements and equivalences, and if-then statements and implications.

of $pump.duration_{p,o}$ (abbreviated by $p.d_{p,o}$). It is important to notice that $p.s_{p,o}$ is the "equivalent CLP variable" of $s_{p,o}$, $p.e_{p,o}$ is the equivalent CLP variable of $f_{p,o}$, and $p.d_{p,o}$ has assigned a value equivalent to $f_{p,o} - s_{p,o}$. The OPL tool has also a series of other specific scheduling and resource-allocation built-in applications. They are generally organized around the various types of resources, and defined over a global time interval, which spans since the scheduling origin to the scheduling horizon. The interested reader should consult the OPL's language manual (ILOG, 2002a).

Expressions 5.8 (page 89) to 5.35 (page 97) establish the *Main Model* mathematical formulation, according to an MILP approach. The CLP model is created based on this MILP approach, in exception that:

(i) Variables $f_{p,o}$ and $s_{p,o}$ are replaced by, respectively, $p.e_{p,o}$ and $p.s_{p,o}$;

(ii) The if-then and the if-then-else statements are directly expressed by implications and equivalences;

(iii) Global constraints and a "search procedure" are incorporated into the model.

Sections 6.2.1 to 6.2.4 present a CLP formulation for the pipeline-scheduling problem previously defined in chapter 5.

6.2.1 CLP Objective Function

The objective function, which defines the operational cost minimization, is maintained as defined in expression 5.8, in exception that the term $f_{p,o} - s_{p,o}$ is replaced by $p.d_{p,o}$.

6.2.2 CLP Constraints

The if-then-else statement 5.9 is rewritten as indicated in equivalence 6.1 ($f_{p,o}$ is replaced by $p.e_{p,o}$ and $s_{pa,o}$ is replaced by $p.s_{pa,o}$). Equations 5.10 and 5.11 are maintained as originally stated. In equivalence 6.1, "$gt_{p,pa,o} = 1 \leftrightarrow p.e_{p,o} \leq p.s_{pa,o}$" is expressed as "$gt_{p,pa,o} \leftrightarrow p.e_{p,o} \leq p.s_{pa,o}$". Thus, the antecedent "$gt_{p,pa,o} = 1$" is simply expressed as "$gt_{p,pa,o}$"; the consequent, however, is maintained without syntactical simplification. This convention is henceforth used in CLP constraints that involve equivalences and implications. Therefore, either an equivalence or an implication holds if its antecedent holds, that is, if its antecedent is true[2].

$$gt_{p,pa,o} \leftrightarrow p.e_{p,o} \leq p.s_{pa,o} \qquad \forall o{\in}O,\ p{\in}P_o,\ pa{\in}P_o,\ p{\neq}pa \tag{6.1}$$

The if-then statement 5.12 is rewritten as indicated in implication 6.2. Equation 5.13 and inequalities 5.14 and 5.15 are maintained as originally stated.

$$t_{p,pa,o} \rightarrow seq_{pa,o} - seq_{p,o} = 1 \qquad \forall o{\in}O,\ p{\in}P_o,\ pa{\in}P_o,\ p{\neq}pa \tag{6.2}$$

The if-then statement 5.16 is rewritten as indicated in implication 6.3. Equation 5.17 is maintained as originally stated.

$$first_{p,o} \rightarrow seq_{p,o} = 1 \qquad \forall o{\in}O,\ p{\in}P_o \tag{6.3}$$

The if-then statement 5.18 is rewritten as indicated in implication 6.4. Equation 5.19 is maintained as originally stated.

$$last_{p,o} \rightarrow seq_{p,o} = np_o \qquad \forall o{\in}O,\ p{\in}P_o \tag{6.4}$$

The *and* statement modeled in 5.20 by means of three sets of inequalities is directly expressed, according to 6.5.

$$tw_{p,pa} = (last_{p,o})\ \textsf{and}\ (first_{pa,\bar{o}}) \qquad \forall p{\in}P_o,\ pa{\in}P_{\bar{o}},\ o = d,\ \bar{o} = \bar{d} \tag{6.5}$$

The if-then-else statements 5.21 and 5.22 are rewritten as indicated in, respectively, equivalences 6.6 and 6.7 ($f_{p,o}$ is replaced by $p.e_{p,o}$ and $s_{p,o}$ is replaced by $p.s_{p,o}$).

$$fgh_{p,o,e} \leftrightarrow p.e_{p,o} \geq fh_e \qquad \forall o{\in}O,\ p{\in}P_o,\ e{\in}E \tag{6.6}$$

$$sgh_{p,o,e} \leftrightarrow p.s_{p,o} \geq sh_e \qquad \forall o{\in}O,\ p{\in}P_o,\ e{\in}E \tag{6.7}$$

The if-then statements 5.23 to 5.26 are rewritten as indicated in, respectively, implications 6.8 to 6.11 ($f_{p,o}$ is replaced by $p.e_{p,o}$ and $s_{p,o}$ is replaced by $p.s_{p,o}$).

[2]In case of binary variables, the antecedent is true whether it is equal to one, zero otherwise.

Inequality 5.27 is maintained as originally stated. Equation 5.28 and inequalities 5.29 and 5.30 are also maintained as originally stated, in exception that the term $f_{p,o} - s_{p,o}$ is replaced by $p.d_{p,o}$.

$$(\neg fgh_{p,o,e}) \wedge (\neg sgh_{p,o,e}) \rightarrow inh_{p,o,e} \geq p.e_{p,o} - sh_e \qquad \forall o \in O,\ p \in P_o,\ e \in E \tag{6.8}$$

$$(\neg fgh_{p,o,e}) \wedge (sgh_{p,o,e}) \rightarrow inh_{p,o,e} = p.e_{p,o} - p.s_{p,o} \qquad \forall o \in O,\ p \in P_o,\ e \in E \tag{6.9}$$

$$(fgh_{p,o,e}) \wedge (\neg sgh_{p,o,e}) \rightarrow inh_{p,o,e} = fh_e - sh_e \qquad \forall o \in O,\ p \in P_o,\ e \in E \tag{6.10}$$

$$(fgh_{p,o,e}) \wedge (sgh_{p,o,e}) \rightarrow inh_{p,o,e} \geq fh_e - p.s_{p,o} \qquad \forall o \in O,\ p \in P_o,\ e \in E \tag{6.11}$$

Inequalities 5.31 to 5.35 are rewritten as indicated in 6.12 to 6.16 ($f_{p,o}$ is replaced by $p.e_{p,o}$, $f_{pa,\bar{o}}$ is replaced by $p.e_{pa,\bar{o}}$, $s_{p,o}$ is replaced by $p.s_{p,o}$, and $s_{pa,\bar{o}}$ is replaced by $p.s_{pa,\bar{o}}$).

$$p.e_{p,o} \leq p.s_{pa,\bar{o}} \qquad \forall p \in P_o,\ pa \in P_{\bar{o}},\ o = d,\ \bar{o} = \bar{d} \tag{6.12}$$

$$p.s_{p,o} \geq 0 \qquad \forall p \in P_o,\ o = d \tag{6.13}$$

$$p.s_{pa,\bar{o}} \geq fd^{min} \qquad \forall pa \in P_{\bar{o}},\ \bar{o} = \bar{d} \tag{6.14}$$

$$p.e_{p,o} \leq fd \qquad \forall p \in P_o,\ o = d \tag{6.15}$$

$$p.e_{pa,\bar{o}} \leq H \qquad \forall pa \in P_{\bar{o}},\ \bar{o} = \bar{d} \tag{6.16}$$

6.2.3 CLP Global Constraints

Another important feature of a CLP framework is the availability of global constraints. Global constraints, a fundamental tool for solving a variety of combinatorial optimization problems, enforce complex relationships among variables (ILOG, 2002b). In particular, OPL offers a variety of global constraints over discrete values. In fact, the CLP mechanisms in OPL are based on finite domain constraint logic programming devices (ILOG, 2002a). Therefore, some functionalities, such as global constraints, are just available for variables ranging in a finite domain of values.

The pipeline-scheduling problem was previously addressed by a continuous time MILP approach. The pumping starting time ($s_{p,o}$) and the pumping completion time ($f_{p,o}$) were considered to assume real values inside the scheduling horizon. In the CLP formulation, variables $p.s_{p,o}$ ($pump.start_{p,o}$) and $p.e_{p,o}$ ($pump.end_{p,o}$) can just assume integer values in the scheduling horizon. Therefore, the former MILP continuous time approach is modified in the CLP model, which just accepts integer-valued variables.

Expressions 6.17 to 6.23 define global constraints[3] in the CLP model. In particular, expressions 6.21 to 6.23 use one of the built-in scheduling features of OPL: the *precedence* constraint. This constraint can be used in variables of type *activity*[4]. In order to exemplify the precedence constraint effect, let's consider the hypothetical statement "a `precedes` b". This statement specifies that the ending date of an activity "a" (a.end) must be smaller or equal than the starting date of an activity "b" (b.start).

$$\texttt{alldifferent}(seq_{p,o}) \qquad \forall p \in P_o,\ o = d \tag{6.17}$$

$$\texttt{alldifferent}(seq_{pa,\bar{o}}) \qquad \forall pa \in P_{\bar{o}},\ \bar{o} = \bar{d} \tag{6.18}$$

$$\texttt{alldifferent}(p.s_{p,o}) \qquad \forall o \in O,\ p \in P_o \tag{6.19}$$

$$\texttt{alldifferent}(p.e_{p,o}) \qquad \forall o \in O,\ p \in P_o \tag{6.20}$$

$$gt_{p,pa,o} \leftrightarrow pump_{p,o}\ \texttt{precedes}\ pump_{pa,o} \qquad \forall o \in O,\ p \in P_o,\ pa \in P_o,\ p \neq pa \tag{6.21}$$

$$first_{p,o} \rightarrow pump_{p,o}\ \texttt{precedes}\ pump_{pa,o} \qquad \forall o \in O,\ p \in P_o \tag{6.22}$$

$$last_{p,o} \rightarrow pump_{pa,o}\ \texttt{precedes}\ pump_{p,o} \qquad \forall o \in O,\ p \in P_o \tag{6.23}$$

6.2.4 CLP Search Procedure

As stated in section 2.1 the CLP mechanisms of constraint propagation and domain reduction can be applied to reduce the search space of variables. However, while they may determine whether a model is infeasible, they do not necessarily find solutions to the model. To do this, one must program a search procedure, which typically consists of two main parts (Hentenryck et al., 2000):

(i) A *search component* defining the search tree to be explored; and,

[3]Further details about global constraints are given in section 2.1.
[4]The variable $pump_{p,o}$ $\forall o \in O,\ p \in P_o$ is declared as having the type *activity*.

(ii) A *strategy component* specifying how to explore the search tree.

OPL has a number of default search procedures, and it also offers the ability to specify search procedures tailored to the application at hand. The OPL's search functionalities allow that special-purpose heuristics be incorporated in the model. Traditionally, modeling languages and most mathematical programming packages hide the search procedures from users, who can "control" them only through a set of parameters (ILOG, 2002a).

At this CLP model, a predefined search component called `generate` is used. This component simply receives a discrete variable, or an array of discrete variables, and generates values for the specified variables, respecting the domain conditions (ILOG, 2002a). Therefore, the search tree is created based on the OPL's built-in command `generate`[5]. Expression 6.24 defines the adopted search component. In addition, the generated search tree is explored by means of the traditional depth-first approach (strategy component), which is described, for instance, in ILOG (2002a).

$$\texttt{generate}(seq_{p,o}) \qquad \forall o \in O,\ p \in P_o \tag{6.24}$$

Therefore, based on section 6.2 directives, the *Main Model* is rewritten according to CLP modeling features. At this point the reader must be aware that a complete CLP model, which indeed addresses the pipeline-scheduling problem under analysis, is already formulated.

6.2.5 CLP Model: The Number of Variables/Constraints

Equation 6.25 indicates the total number of variables presented by the CLP model (tnv_{CLP}) heretofore described in section 6.2. This equation is obtained in a similar reasoning explained in section 5.4.3.3 (page 97). One can observe that equation 5.38 (page 99), which defines the total number of variables presented by the MILP formulation (tnv_{MILP}), and equation 6.25 differ. The CLP model was derived from the MILP model, but the CLP formulation does not demand some

[5] "The order in which one generates the nodes of the search tree can have a dramatic effect on the size of the tree." (Hooker, 2000)

auxiliary variables, such as $\delta^{00}_{p,o,e}$, $\delta^{01}_{p,o,e}$, $\delta^{10}_{p,o,e}$, and $\delta^{11}_{p,o,e}$, that were necessary in the MILP approach. Additionally, the CLP model incorporates specialized scheduling features for pumping activities, and auxiliary variables such as $p.s_{p,o}$, $p.e_{p,o}$, and $p.d_{p,o}$ were created for the CLP approach[6]. In particular, the MILP variables $s_{p,o}$ and $f_{p,o}$ were replaced by, respectively, $p.s_{p,o}$ and $p.e_{p,o}$.

$$tnv_{\mathrm{CLP}} \;=\; 2\cdot\sum_{o\in O} np_o^2 \;+\; (8+4\cdot ne)\cdot\sum_{o\in O} np_o \;+\; \prod_{o\in O} np_o \;+\; 1 \tag{6.25}$$

In order to illustrate the use of equation 6.25, let's consider a hypothetical example with $np_o = 6\ \forall o\in O$ and $ne = 5$. Hence, tnv_{CLP} can be calculated as: $2\cdot(6^2+6^2) + (8+4\cdot 5)\cdot(6+6) + (6\cdot 6) + 1$, that is, $tnv_{\mathrm{CLP}} = 517$ (see table 6.3, page 140, column "Number of Variables", sub-column CLP). In a similar way, equation 5.38 yields $tnv_{\mathrm{MILP}} = 2\cdot(6^2+6^2) + (3+8\cdot 5)\cdot(6+6) + (6\cdot 6) + 1$, that is $tnv_{\mathrm{MILP}} = 697$ (see table 6.3, column "Number of Variables", sub-column MILP). Therefore, $tnv_{\mathrm{CLP}} \neq tnv_{\mathrm{MILP}}$.

In a similar reasoning used to verify that $tnv_{\mathrm{CLP}} \neq tnv_{\mathrm{MILP}}$, one can observe that the number of MILP constraints differs from the number of CLP constraints (see table 6.3, column "Number of Constraints", sub-columns MILP and CLP). As stated in section 6.2.2, some MILP constraints are maintained in the CLP model (e.g. equation 5.13, inequalities 5.14 and 5.15 - page 91), but other ones are rewritten according to CLP modeling devices, such as implications ($\rightarrow$) and equivalences ($\leftrightarrow$). For example, the if-then statement 5.16 (page 92) is rewritten by the implication 6.3 (page 119). However, the if-then 5.16 demands two sets of base level inequalities to state the same modeling condition directly expressed as the CLP implication 6.3. Thus, the number of constraints used to model equivalent MILP/CLP conditions may differ. For simplicity, this thesis does not state equations that determine neither the CLP nor the MILP number of constraints.

6.3 Building a CLP-MILP Model

The combined CLP-MILP version of the *Main Model* is composed by the CLP formulation established in section 6.2, and the MILP formulation presented in sec-

[6]Further details about the *activity concept* (ILOG, 2002a) are given in section 6.2, page 117.

tion 5.4.3. The search procedure of both models is integrated by means of an OPL's built-in feature, which essentially uses the MILP linear relaxation as the main element of integration.

OPL offers the functionality of adding information provided by MILP linear relaxations into CLP search procedures. This functionality is activated by the keywords `with linear relaxation` (e.g. `minimize with linear relaxation` instead of `minimize`). This functionality adds the linear relaxation of any linear or integer constraint to the constraint store[7]. In other words, the constraint store receives the relaxation of an integer constraint c whenever c becomes a float (real) constraint, and all its integer variables are considered of type float (ILOG, 2002a). Thus, at any time the OPL's solver, which is a branch-and-bound algorithm that uses a cooperation between a constraint-based domain reduction and a simplex code, produces a lower bound (minimization model), this bound is used to tighten the search (ILOG, 2002a). As stated in section 2.1, the CLP search process lacks the global relaxation of a model. In this integrated approach, the MILP linear relaxation serves as such global model relaxation, contributing to the search process reduction. Section 6.4 (numerical results) indeed confirms this fact. In addition, ILOG (2002b) brings an example that displays the generated search trees with and without the `minimize with linear relaxation` functionality[8]. The example evidences the linear relaxation influence on the search tree pruning.

In a practical standpoint, the CLP formulation (section 6.2) and the MILP formulation (section 5.4.3) are written in a unified OPL's framework. Then, the solver is informed by specific keywords that it should perform an "integrated" search procedure, which uses linear relaxations on all linear/integer constraints. Thus, the linear programming engine (CPLEX) produces a bound at each node of the solver search tree (ILOG, 2002b). In particular, if one takes either the CLP model or the MILP model, this specific formulation could, in theory, be solved[9], giving insights into the pipeline-scheduling resolution. The fact that has to be investigated is

[7]The constraint store is a constraint-solving system reasoning about fundamental properties of constraints, such as satisfiability and entailment (ILOG, 2002a).

[8]For details see Chapter 7 of ILOG (2002b).

[9]The computational burden can hinder the model resolution.

whether the integrated approach presents a computational performance better than the root techniques or not. Section 6.4 gives particular attention to computational issues involving the MILP, the CLP, and the CLP-MILP models.

6.3.1 CLP-MILP Objective Function

Expression 6.26 illustrates the CLP-MILP objective function, which defines the operational cost minimization. This function is maintained as in formulation 5.8, in exception that the keywords `with linear relaxation` are added.

$$\begin{aligned}
& \texttt{minimize with linear relaxation} \\
& \sum_{o \in O} \sum_{p \in P_o} C_o^{low} \cdot (f_{p,o} - s_{p,o}) \\
+ \; & \sum_{o \in O} \sum_{p \in P_o} \sum_{e \in E} C_o^{high} \cdot (inh_{p,o,e} - stop_{p,o,e}) \\
+ \; & \sum_{o \in O} \sum_{p \in P_o} \sum_{\substack{pa \in P_o, \\ pa \neq p}} C_o^{plug} \cdot t_{p,pa,o} \cdot plug_{p,pa} \\
+ \; & \sum_{\substack{o=d, \\ p \in P_o}} \sum_{\substack{\bar{o}=\bar{d}, \\ pa \in P_{\bar{o}}}} C_{\bar{o}}^{plug} \cdot tw_{p,pa} \cdot plug_{p,pa} \\
+ \; & \sum_{\substack{p=pin, \\ o=d, \\ pa \in P_o}} C_o^{plug} \cdot first_{p,o} \cdot plug_{p,pa} \\
+ \; & C^{pres} \cdot dpres
\end{aligned} \tag{6.26}$$

6.3.2 CLP-MILP Constraints

The CLP-MILP constraints involve the CLP constraints established in section 6.2 and the MILP constraints presented in section 5.4.3. In order to simplify the CLP-MILP model understanding, all constraints are herein clustered in expressions 6.27 to 6.81. The reader must notice that the CLP constraints can be identified by the presence of symbols $\leftrightarrow$ and $\rightarrow$, by variables $\boldsymbol{p.s_{p,o}}$, $\boldsymbol{p.d_{p,o}}$, and $\boldsymbol{p.e_{p,o}}$, or by commands written in `typewriter` font.

6.3.2.1 High-Level CLP-MILP Structures

Constraints 6.27 to 6.46 gather the high-level MILP statements (section 5.4.3.2) and the equivalent CLP formulations of such statements (section 6.2.2).

$$\textbf{If } gt_{p,pa,o} \textbf{ Then } f_{p,o} \leq s_{pa,o} \textbf{ Else } f_{p,o} > s_{pa,o} \quad \forall o \in O,\ p \in P_o,\ pa \in P_o,\ p \neq pa$$

$$\begin{cases} f_{p,o} - s_{pa,o} \leq U \cdot (1 - gt_{p,pa,o}) & \forall o \in O,\ p \in P_o,\ pa \in P_o,\ p \neq pa \\ f_{p,o} - s_{pa,o} \geq (L - \varepsilon) \cdot gt_{p,pa,o} + \varepsilon & \forall o \in O,\ p \in P_o,\ pa \in P_o,\ p \neq pa \end{cases} \tag{6.27}$$

$$gt_{p,pa,o} \leftrightarrow p.e_{p,o} \leq p.s_{pa,o} \qquad \forall o \in O,\ p \in P_o,\ pa \in P_o,\ p \neq pa \tag{6.28}$$

$$\textbf{If} \quad t_{p,pa,o} \quad \textbf{Then} \quad seq_{pa,o} - seq_{p,o} = 1 \quad \forall o \in O,\ p \in P_o,\ pa \in P_o,\ p \neq pa$$

$$\begin{cases} seq_{pa,o} - seq_{p,o} - 1 \leq U \cdot (1 - t_{p,pa,o}) & \forall o \in O,\ p \in P_o,\ pa \in P_o,\ p \neq pa \\ seq_{pa,o} - seq_{p,o} - 1 \geq L \cdot (1 - t_{p,pa,o}) & \forall o \in O,\ p \in P_o,\ pa \in P_o,\ p \neq pa \end{cases} \tag{6.29}$$

$$t_{p,pa,o} \longrightarrow seq_{pa,o} - seq_{p,o} = 1 \qquad \forall o \in O,\ p \in P_o,\ pa \in P_o,\ p \neq pa \tag{6.30}$$

$$\textbf{If} \quad first_{p,o} \quad \textbf{Then} \quad seq_{p,o} = 1 \qquad \forall o \in O,\ p \in P_o$$

$$\begin{cases} seq_{p,o} - 1 \leq U \cdot (1 - first_{p,o}) & \forall o \in O,\ p \in P_o \\ seq_{p,o} - 1 \geq L \cdot (1 - first_{p,o}) & \forall o \in O,\ p \in P_o \end{cases} \tag{6.31}$$

$$first_{p,o} \longrightarrow seq_{p,o} = 1 \qquad \forall o \in O,\ p \in P_o \tag{6.32}$$

$$\textbf{If} \quad last_{p,o} \quad \textbf{Then} \quad seq_{p,o} = np_o \qquad \forall o \in O,\ p \in P_o$$

$$\begin{cases} seq_{p,o} - np_o \leq U \cdot (1 - last_{p,o}) & \forall o \in O,\ p \in P_o \\ seq_{p,o} - np_o \geq L \cdot (1 - last_{p,o}) & \forall o \in O,\ p \in P_o \end{cases} \tag{6.33}$$

$$last_{p,o} \longrightarrow seq_{p,o} = np_o \qquad \forall o \in O,\ p \in P_o \tag{6.34}$$

If $fgh_{p,o,e}$ **Then** $f_{p,o} \geq fh_e$ **Else** $f_{p,o} < fh_e \quad \forall o{\in}O,\ p{\in}P_o,\ e{\in}E$

$$\begin{cases} f_{p,o} - fh_e \geq L \cdot (1 - fgh_{p,o,e}) & \forall o{\in}O,\ p{\in}P_o,\ e{\in}E \\ f_{p,o} - fh_e \leq (U + \varepsilon) \cdot fgh_{p,o,e} - \varepsilon & \forall o{\in}O,\ p{\in}P_o,\ e{\in}E \end{cases} \tag{6.35}$$

$$fgh_{p,o,e} \leftrightarrow p.e_{p,o} \geq fh_e \qquad \forall o{\in}O,\ p{\in}P_o,\ e{\in}E \tag{6.36}$$

If $sgh_{p,o,e}$ **Then** $s_{p,o} \geq sh_e$ **Else** $s_{p,o} < sh_e \quad \forall o{\in}O,\ p{\in}P_o,\ e{\in}E$

$$\begin{cases} s_{p,o} - sh_e \geq L \cdot (1 - sgh_{p,o,e}) & \forall o{\in}O,\ p{\in}P_o,\ e{\in}E \\ s_{p,o} - sh_e \leq (U + \varepsilon) \cdot sgh_{p,o,e} - \varepsilon & \forall o{\in}O,\ p{\in}P_o,\ e{\in}E \end{cases} \tag{6.37}$$

$$sgh_{p,o,e} \leftrightarrow p.s_{p,o} \geq sh_e \qquad \forall o{\in}O,\ p{\in}P_o,\ e{\in}E \tag{6.38}$$

If $(\neg fgh_{p,o,e}) \wedge (\neg sgh_{p,o,e})$ **Then** $inh_{p,o,e} \geq f_{p,o} - sh_e \quad \forall o{\in}O,\ p{\in}P_o,\ e{\in}E$

$$\begin{cases} \delta^{00}_{p,o,e} \leq 1 - fgh_{p,o,e} & \forall o{\in}O,\ p{\in}P_o,\ e{\in}E \\ \delta^{00}_{p,o,e} \leq 1 - sgh_{p,o,e} & \forall o{\in}O,\ p{\in}P_o,\ e{\in}E \\ \delta^{00}_{p,o,e} \geq 1 - fgh_{p,o,e} - sgh_{p,o,e} & \forall o{\in}O,\ p{\in}P_o,\ e{\in}E \\ inh_{p,o,e} - f_{p,o} + sh_e \geq L \cdot (1 - \delta^{00}_{p,o,e}) & \forall o{\in}O,\ p{\in}P_o,\ e{\in}E \end{cases} \tag{6.39}$$

$$(\neg fgh_{p,o,e}) \wedge (\neg sgh_{p,o,e}) \longrightarrow inh_{p,o,e} \geq p.e_{p,o} - sh_e \qquad \forall o{\in}O,\ p{\in}P_o,\ e{\in}E \tag{6.40}$$

If $(\neg fgh_{p,o,e}) \wedge (sgh_{p,o,e})$ **Then** $inh_{p,o,e} = f_{p,o} - s_{p,o} \quad \forall o{\in}O,\ p{\in}P_o,\ e{\in}E$

$$\begin{cases} \delta^{01}_{p,o,e} \leq 1 - fgh_{p,o,e} & \forall o{\in}O,\ p{\in}P_o,\ e{\in}E \\ \delta^{01}_{p,o,e} \leq sgh_{p,o,e} & \forall o{\in}O,\ p{\in}P_o,\ e{\in}E \\ \delta^{01}_{p,o,e} \geq sgh_{p,o,e} - fgh_{p,o,e} & \forall o{\in}O,\ p{\in}P_o,\ e{\in}E \\ inh_{p,o,e} - f_{p,o} + s_{p,o} \leq U \cdot (1 - \delta^{01}_{p,o,e}) & \forall o{\in}O,\ p{\in}P_o,\ e{\in}E \\ inh_{p,o,e} - f_{p,o} + s_{p,o} \geq L \cdot (1 - \delta^{01}_{p,o,e}) & \forall o{\in}O,\ p{\in}P_o,\ e{\in}E \end{cases} \tag{6.41}$$

$$(\neg fgh_{p,o,e}) \wedge (sgh_{p,o,e}) \longrightarrow inh_{p,o,e} = p.e_{p,o} - p.s_{p,o} \qquad \forall o{\in}O,\ p{\in}P_o,\ e{\in}E \tag{6.42}$$

If $(fgh_{p,o,e}) \wedge (\neg sgh_{p,o,e})$ **Then** $inh_{p,o,e} = fh_e - sh_e \quad \forall o \in O,\ p \in P_o,\ e \in E$

$$\begin{cases} \delta^{10}_{p,o,e} \leq fgh_{p,o,e} & \forall o \in O,\ p \in P_o,\ e \in E \\ \delta^{10}_{p,o,e} \leq 1 - sgh_{p,o,e} & \forall o \in O,\ p \in P_o,\ e \in E \\ \delta^{10}_{p,o,e} \geq fgh_{p,o,e} - sgh_{p,o,e} & \forall o \in O,\ p \in P_o,\ e \in E \\ inh_{p,o,e} - fh_e + sh_e \leq U \cdot (1 - \delta^{10}_{p,o,e}) & \forall o \in O,\ p \in P_o,\ e \in E \\ inh_{p,o,e} - fh_e + sh_e \geq L \cdot (1 - \delta^{10}_{p,o,e}) & \forall o \in O,\ p \in P_o,\ e \in E \end{cases} \tag{6.43}$$

$$(fgh_{p,o,e}) \wedge (\neg sgh_{p,o,e}) \longrightarrow inh_{p,o,e} = fh_e - sh_e \qquad \forall o \in O,\ p \in P_o,\ e \in E \tag{6.44}$$

If $(fgh_{p,o,e}) \wedge (sgh_{p,o,e})$ **Then** $inh_{p,o,e} \geq fh_e - s_{p,o} \quad \forall o \in O,\ p \in P_o,\ e \in E$

$$\begin{cases} \delta^{11}_{p,o,e} \leq fgh_{p,o,e} & \forall o \in O,\ p \in P_o,\ e \in E \\ \delta^{11}_{p,o,e} \leq sgh_{p,o,e} & \forall o \in O,\ p \in P_o,\ e \in E \\ \delta^{11}_{p,o,e} \geq fgh_{p,o,e} + sgh_{p,o,e} - 1 & \forall o \in O,\ p \in P_o,\ e \in E \\ inh_{p,o,e} - fh_e + s_{p,o} \geq L \cdot (1 - \delta^{11}_{p,o,e}) & \forall o \in O,\ p \in P_o,\ e \in E \end{cases} \tag{6.45}$$

$$(fgh_{p,o,e}) \wedge (sgh_{p,o,e}) \longrightarrow inh_{p,o,e} \geq fh_e - p.s_{p,o} \qquad \forall o \in O,\ p \in P_o,\ e \in E \tag{6.46}$$

6.3.2.2 Common CLP-MILP Constraints

Constraints 6.47 to 6.54 are part of either the root CLP model (section 6.2.2) or the root MILP model (section 5.4.3.2). For simplicity, when the combined CLP-MILP model is formulated, these constraints, which are common in the root models, are written just once.

$$\sum_{p \in P_o} \sum_{\substack{pa \in P_o, \\ pa \neq p}} gt_{p,pa,o} = \sum_{k=1}^{k=np_o} (k-1) \qquad \forall o \in O \tag{6.47}$$

$$seq_{p,o} = np_o - \sum_{\substack{pa \in P_o, \\ pa \neq p}} gt_{p,pa,o} \qquad \forall o \in O,\ p \in P_o \tag{6.48}$$

$$\sum_{p \in P_o} \sum_{\substack{pa \in P_o, \\ pa \neq p}} t_{p,pa,o} = np_o - 1 \qquad \forall o \in O \tag{6.49}$$

$$\sum_{\substack{pa \in P_o, \\ pa \neq p}} t_{p,pa,o} \leq 1 \qquad \forall o \in O,\ p \in P_o \tag{6.50}$$

$$\sum_{\substack{pa \in P_o, \\ pa \neq p}} t_{pa,p,o} \leq 1 \qquad \forall o \in O,\ p \in P_o \tag{6.51}$$

$$\sum_{p \in P_o} first_{p,o} = 1 \qquad \forall o \in O \tag{6.52}$$

$$\sum_{p \in P_o} last_{p,o} = 1 \qquad \forall o \in O \tag{6.53}$$

$$stop_{p,o,e} \leq inh_{p,o,e} \qquad \forall o \in O,\ p \in P_o,\ e \in E \tag{6.54}$$

6.3.2.3 Ordinary CLP-MILP Structures

Constraints 6.55 to 6.72 are also part of the CLP-MILP model. In addition, equations 6.73 and 6.74, which link the CLP variables $p.s_{p,o}$ and $p.e_{p,o}$ with, respectively, the MILP variables $s_{p,o}$ and $f_{p,o}$, are added to the formulation.

$$\begin{array}{ll} tw_{p,pa} = (last_{p,o}) \wedge (first_{pa,\bar{o}}) & \forall p \in P_o,\ pa \in P_{\bar{o}},\ o = d,\ \bar{o} = \bar{d} \\ \left\{ \begin{array}{l} tw_{p,pa} \leq last_{p,o} \\ tw_{p,pa} \leq first_{pa,\bar{o}} \\ tw_{p,pa} \geq last_{p,o} + first_{pa,\bar{o}} - 1 \end{array} \right. & \begin{array}{l} \forall p \in P_o,\ pa \in P_{\bar{o}},\ o = d,\ \bar{o} = \bar{d} \\ \forall p \in P_o,\ pa \in P_{\bar{o}},\ o = d,\ \bar{o} = \bar{d} \\ \forall p \in P_o,\ pa \in P_{\bar{o}},\ o = d,\ \bar{o} = \bar{d} \end{array} \end{array} \tag{6.55}$$

$$tw_{p,pa} = (last_{p,o})\ \texttt{and}\ (first_{pa,\bar{o}}) \qquad \forall p \in P_o,\ pa \in P_{\bar{o}},\ o = d,\ \bar{o} = \bar{d} \tag{6.56}$$

$$dpres = H - \sum_{o \in O} \sum_{p \in P_o} (f_{p,o} - s_{p,o}) + \sum_{o \in O} \sum_{p \in P_o} \sum_{e \in E} stop_{p,o,e} \tag{6.57}$$

$$dpres = H - \sum_{o \in O} \sum_{p \in P_o} p.d_{p,o} + \sum_{o \in O} \sum_{p \in P_o} \sum_{e \in E} stop_{p,o,e} \tag{6.58}$$

$$f_{p,o} - s_{p,o} \geq d^{min}_{p,o} + \sum_{e \in E} stop_{p,o,e} + last_{p,o} \cdot td^{min}_{p,o} \qquad \forall o \in O,\ p \in P_o \tag{6.59}$$

$$f_{p,o} - s_{p,o} \leq d^{max}_{p,o} + \sum_{e \in E} stop_{p,o,e} + last_{p,o} \cdot td^{max}_{p,o} \qquad \forall o \in O,\ p \in P_o \tag{6.60}$$

$$p.d_{p,o} \geq d^{min}_{p,o} + \sum_{e \in E} stop_{p,o,e} + last_{p,o} \cdot td^{min}_{p,o} \qquad \forall o \in O,\ p \in P_o \tag{6.61}$$

$$p.d_{p,o} \leq d^{max}_{p,o} + \sum_{e \in E} stop_{p,o,e} + last_{p,o} \cdot td^{max}_{p,o} \qquad \forall o \in O,\ p \in P_o \tag{6.62}$$

$$f_{p,o} \leq s_{pa,\bar{o}} \qquad \forall p \in P_o,\ pa \in P_{\bar{o}},\ o = d,\ \bar{o} = \bar{d} \tag{6.63}$$

$$s_{p,o} \geq 0 \qquad \forall p \in P_o,\ o = d \tag{6.64}$$

$$s_{pa,\bar{o}} \geq fd^{min} \qquad \forall pa \in P_{\bar{o}},\ \bar{o} = \bar{d} \tag{6.65}$$

$$f_{p,o} \leq fd \qquad \forall p \in P_o,\ o = d \tag{6.66}$$

$$f_{pa,\bar{o}} \leq H \qquad \forall pa \in P_{\bar{o}},\ \bar{o} = \bar{d} \tag{6.67}$$

$$p.e_{p,o} \leq p.s_{pa,\bar{o}} \qquad \forall p \in P_o,\ pa \in P_{\bar{o}},\ o = d,\ \bar{o} = \bar{d} \tag{6.68}$$

$$p.s_{p,o} \geq 0 \qquad \forall p \in P_o,\ o = d \tag{6.69}$$

$$p.s_{pa,\bar{o}} \geq fd^{min} \qquad \forall pa \in P_{\bar{o}},\ \bar{o} = \bar{d} \tag{6.70}$$

$$p.e_{p,o} \leq fd \qquad \forall p \in P_o,\ o = d \tag{6.71}$$

$$p.e_{pa,\bar{o}} \leq H \qquad \forall pa \in P_{\bar{o}},\ \bar{o} = \bar{d} \tag{6.72}$$

$$p.s_{p,o} = s_{p,o} \qquad \forall o \in O,\ p \in P_o \tag{6.73}$$

$$p.e_{p,o} = f_{p,o} \qquad \forall o \in O,\ p \in P_o \tag{6.74}$$

6.3.2.4 CLP-MILP Global Constraints

$$\texttt{alldifferent}(seq_{p,o}) \qquad \forall p \in P_o,\ o = d \tag{6.75}$$

$$\texttt{alldifferent}(seq_{pa,\bar{o}}) \qquad \forall pa \in P_{\bar{o}},\ \bar{o} = \bar{d} \tag{6.76}$$

$$\texttt{alldifferent}(p.s_{p,o}) \qquad \forall o \in O,\ p \in P_o \tag{6.77}$$

$$\texttt{alldifferent}(p.e_{p,o}) \qquad \forall o \in O,\ p \in P_o \tag{6.78}$$

$$gt_{p,pa,o} \leftrightarrow pump_{p,o}\ \texttt{precedes}\ pump_{pa,o} \qquad \forall o \in O,\ p \in P_o,\ pa \in P_o,\ p \neq pa \tag{6.79}$$

$$first_{p,o} \rightarrow pump_{p,o}\ \texttt{precedes}\ pump_{pa,o} \qquad \forall o \in O,\ p \in P_o \tag{6.80}$$

$$last_{p,o} \rightarrow pump_{pa,o}\ \texttt{precedes}\ pump_{p,o} \qquad \forall o \in O,\ p \in P_o \tag{6.81}$$

6.3.3 CLP-MILP Search Component

$$\texttt{generate}(seq_{p,o}) \qquad \forall o\in O,\ p\in P_o \tag{6.82}$$

6.3.4 CLP-MILP Model: The Number of Variables

Equation 6.83 indicates the total number of variables presented by the integrated CLP-MILP model ($tnv_{\text{CLP-MILP}}$) described in sections 6.3.1 to 6.3.3. This equation is obtained in a similar reasoning explained in section 5.4.3.3 (page 97). At first sight, one could think that the total number of variables presented by the CLP-MILP model can be obtained by adding the results presented by equations 6.25 and 5.38, which determine, respectively, the total number variables presented by the CLP (tnv_{CLP}) and the MILP (tnv_{MILP}) models. Nevertheless, many variables are used by both models (e.g. $dpres$; $fgh_{p,o,e}$; $first_{p,o}$; $gt_{p,pa,o}$; $inh_{p,o,e}$; $last_{p,o}$; $seq_{p,o}$; $sgh_{p,o,e}$; $stop_{p,o,e}$; $t_{p,pa,o}$; and $tw_{p,pa}$), but are declared just once in the CLP-MILP framework. Therefore, in a practical standpoint, $tnv_{\text{CLP-MILP}}$ is different from the result provided by $tnv_{\text{CLP}} + tnv_{\text{MILP}}$, as it can be observed by equations 6.83 and 6.84.

$$\begin{aligned} tnv_{\text{CLP-MILP}} &= 2\cdot\sum_{o\in O} np_o^2 + (10+8\cdot ne)\cdot\sum_{o\in O} np_o + \prod_{o\in O} np_o + 1 && (6.83)\\ tnv_{\text{CLP}} + tnv_{\text{MILP}} &= 4\cdot\sum_{o\in O} np_o^2 + (11+12\cdot ne)\cdot\sum_{o\in O} np_o + 2\cdot\prod_{o\in O} np_o + 2 && (6.84)\end{aligned}$$

In order to illustrate the use of equations 6.83 and 6.84, let's consider a hypothetical example with $np_o\!=\!6\ \forall o\in O$ and $ne\!=\!5$. Hence, $tnv_{\text{CLP-MILP}}$ can be calculated as: $2\cdot(6^2{+}6^2) + (10{+}8\cdot5)\cdot(6{+}6) + (6\cdot6) + 1$, that is, $tnv_{\text{CLP-MILP}}\!=\!781$ (see table 6.3, page 140, column "Number of Variables", sub-column CLP-MILP). In a similar way, equation 6.84 yields $tnv_{\text{CLP}}{+}tnv_{\text{MILP}}\ {=}4\cdot(6^2{+}6^2) + (11{+}12\cdot5)\cdot(6{+}6) + 2\cdot(6\cdot6) + 2$, that is, $tnv_{\text{CLP}}{+}tnv_{\text{CLP}} = 1214$ (see table 6.3, column "Number of Variables", and add the values presented in a row of sub-columns CLP and MILP). Therefore, $tnv_{\text{CLP-MILP}} \neq tnv_{\text{CLP}}{+}tnv_{\text{MILP}}$ $(781 \neq 1214\,(517 + 697))$. Other numerical results presented in tables 6.1 (page 134) and 6.3 indeed confirms this fact. Furthermore, the computational results of section 6.4 have demonstrated that the combined CLP-MILP approach tends to have smaller running times than the root MILP and CLP models. Numerical details are given, for instance, in tables 6.1, 6.3, and 6.4 (page 145).

6.4 Results

6.4.1 First Computational Experiment

This section considers a pumping scenario previously exploited in chapter 5, and the input data presented in section 5.5.1 are herein also valid. Therein section 6.4.1, however, numerical comparisons amongst three different *Main Model* versions, which are formulated according to MILP, CLP, and CLP-MILP approaches, are developed.

Table 6.1 provides information about the optimization structure simulation on a Pentium 4, 2.4 GHz, 1 Gbyte RAM. The modeling and optimization tool ILOG OPL Studio 3.6.1 (ILOG, 2002b) is used to implement and solve the optimization structure. There is a series of algorithm settings to be defined in this tool. These settings include, for example, the linear programming method (primal simplex, dual simplex, barrier), the branch-and-bound direction (up, down, both), the node selection (depth first, worst bound, best bound), the use of optimality margin, to name a few. Each of these settings can directly influence the search procedure. For an in-depth discussion, the interested reader is referred to ILOG (2002b). Nevertheless, for simplicity, in the illustrative example herein presented, these settings are maintained in the OPL's default option[10].

In addition, the computational experiments henceforth conducted demand that several instances of different models (MILP, CLP, and CLP-MILP) be solved. In such a task, the functionalities of OPLScript (ILOG, 2002a), a script language for composing and controlling optimization models, were exploited. Basically, OPLScript enables the user to (systematically) solve different instances of the same model, make

[10]For a detailed description of OPL's default option see ILOG (2002b).

data modifications, format output data, and create algorithm solutions where the output of one model is used as the input of a second model. A complete description of OPLScript is given in ILOG (2002a).

In table 6.1, H is the scheduling horizon, ne is the number of on-peak electric time intervals, the MILP label refers to the formulation presented in section 5.4, the CLP label refers to the formulation explained in section 6.2, and the CLP-MILP label refers to the formulation presented in section 6.3.

For each scheduling horizon (H), the optimization structure is run, and a specific cost is attained (objective function value). The *Auxiliary Routine* and the *Tank Bound* simulation data are neglected. These structures required a computational time lower than one second, for all illustrative instances ($114\,\text{h} \leq H \leq 144\,\text{h}$). No uncertainties are considered. The model is solved for fixed product demands.

The columns named "Number of Variables" and "Number of Constraints" indicate the number of variables and constraints presented by the MILP, CLP, and CLP-MILP models. The column "Optimum found in (s)" indicates the processing time (in seconds) spent by each optimization approach to find the optimum solution, according to values of H . At this time, the solution optimality is not yet proved. On the other hand, the column "Optimality proved in (s)" indicates the processing time (in seconds) spent by each optimization approach to prove the solution optimality, for each value of H. The column "Nodes Exploited" indicates the total number of nodes visited by the "branch-and-bound" algorithm, according to the optimization model and the H value, in order to prove the solution optimality. In these three columns, a dash (-) indicates that the *Main Model* was not run to optimality to the specific problem instance, and the computational data are neglected. In fact, the simulation was aborted after 2 h (7200 s) of running. The column "Cost" indicates objective function values, according to the available scheduling horizon (H).

Table 6.1: Main Model Illustrative Instances I – MILP, CLP, and CLP-MILP Formulations.

H (h)	ne	Number of Variables			Number of Constraints			Optimum found in (s)			Optimality proved in (s)			Nodes Exploited			Cost ($)
		MILP	CLP	CLP-MILP	MILP	CLP	CLP-MILP	MILP	CLP	CLP-MILP	MILP	CLP	CLP-MILP	MILP	CLP	CLP-MILP	
114	4	361	273	417	1132	571	1620	11	0.7	0.8	15	0.8	5.0	9453	575	575	174
115	5	425	305	481	1324	631	1860	15	7.9	1.1	17	69	7.3	11573	$156\,10^3$	614	175
116	5	425	305	481	1324	631	1860	2	41	1.1	19	315	7.4	13492	$627\,10^3$	714	176
117	5	425	305	481	1324	631	1860	2	55	2.3	22	417	16	15708	$870\,10^3$	1655	177
118	5	425	305	481	1324	631	1860	9	86	2.4	23	629	16	15028	$1.25\,10^6$	1642	174
119	5	425	305	481	1324	631	1860	2	151	2.5	34	1075	16	23091	$1.90\,10^6$	1783	171
120	5	425	305	481	1324	631	1860	27	1069	5.1	42	6312	33	28509	$13.7\,10^6$	3596	168
121	5	425	305	481	1324	631	1860	13	-	5.8	41	-	37	25145	-	4073	164
122	5	425	305	481	1324	631	1860	34	-	5.8	46	-	36	32258	-	3976	160
123	5	425	305	481	1324	631	1860	21	-	7.2	75	-	44	49079	-	5046	156
124	5	425	305	481	1324	631	1860	51	-	6.0	85	-	39	56613	-	4509	153
125	5	425	305	481	1324	631	1860	30	-	5.9	61	-	38	37645	-	4480	150
126	5	425	305	481	1324	631	1860	3	-	6.6	64	-	49	44967	-	5911	147
127	5	425	305	481	1324	631	1860	2	-	6.6	91	-	46	61202	-	5419	144
128	5	425	305	481	1324	631	1860	34	-	6.5	78	-	40	48898	-	4685	141
129	5	425	305	481	1324	631	1860	20	-	5.0	45	-	25	25999	-	2749	138
130	5	425	305	481	1324	631	1860	90	-	5.1	90	-	25	50686	-	2879	139
131	5	425	305	481	1324	631	1860	13	-	5.4	36	-	26	22185	-	2988	140
132	5	425	305	481	1324	631	1860	40	-	7.0	60	-	25	31740	-	2888	139
133	5	425	305	481	1324	631	1860	37	-	7.0	57	-	26	31651	-	2900	140
134	5	425	305	481	1324	631	1860	8	-	6.9	50	-	25	27005	-	2783	141
135	5	425	305	481	1324	631	1860	43	-	7.1	91	-	26	47142	-	2878	142
136	5	425	305	481	1324	631	1860	29	-	7.2	84	-	26	48055	-	2883	143
137	5	425	305	481	1324	631	1860	8	-	7.1	64	-	26	34860	-	2844	144
138	5	425	305	481	1324	631	1860	45	-	5.6	73	-	22	36620	-	2547	144
139	6	489	337	545	1516	691	2100	50	-	6.9	52	-	25	27746	-	2835	145
140	6	489	337	545	1516	691	2100	55	-	7.0	88	-	26	43455	-	2873	146
141	6	489	337	545	1516	691	2100	116	-	7.5	170	-	28	62829	-	3067	147
142	6	489	337	545	1516	691	2100	3	-	7.6	113	-	28	60221	-	3091	148
143	6	489	337	545	1516	691	2100	30	-	7.4	73	-	29	32219	-	3116	149
144	6	489	337	545	1516	691	2100	2	-	18	57	-	77	29202	-	8445	150

6.4.1.1 Results: Commentaries on Table 6.1

Table 6.1 indicates that the computational effort demanded by the CLP model is greater than the one demanded by the MILP and the CLP-MILP approaches by orders of magnitude. In some illustrative instances ($121\,\text{h} \leq H \leq 144\,\text{h}$), unfortunately, the results regarding the optimal solution could not be achieved by the CLP model. It is necessary to highlight, however, that feasible solutions were attained by the CLP model in few seconds of processing. This fact was observed during simulations, and it is reported in table 6.2. In this table, H is the available scheduling horizon, z_s indicates the best (sub)optimal solution found by the CLP search procedure after ten (10) seconds of processing, z_o is the optimal solution to the specific problem instance, which can be attained in table 6.1, column "Cost", and the term $(z_s - z_o)/z_o$ indicates the difference (in percentage) between z_s and z_o. Therefore, table 6.2 indicates that, in the worst case ($H = 132\,\text{h}$), the CLP model found a suboptimal solution within 31.7 % from the optimal value after ten seconds of processing.

Table 6.2: Main Model Illustrative Instances – CLP (Sub)Optimal Solutions After Ten Seconds.

H (h)	z_s (\$)	z_o (\$)	$(z_s - z_o)/z_o$ (%)	H (h)	z_s (\$)	z_o (\$)	$(z_s - z_o)/z_o$ (%)
114	174	174	0	130	181	139	30.2
115	175	175	0	131	182	140	30.0
116	179	176	1.7	132	183	139	**31.7**
117	181	177	2.6	133	184	140	31.4
118	178	174	2.3	134	185	141	31.2
119	175	171	2.3	135	186	142	31.0
120	172	168	2.4	136	187	143	30.8
121	170	164	3.7	137	188	144	30.6
122	178	160	11.3	138	189	144	31.3
123	174	156	11.5	139	190	145	31.0
124	171	153	11.8	140	191	146	30.8
125	168	150	12.0	141	192	147	30.6
126	165	147	12.2	142	193	148	30.4
127	166	144	15.3	143	194	149	30.2
128	167	141	18.4	144	195	150	30.0
129	156	138	13.0				

The `generate` procedure was used as the CLP search component (see section 6.2.4). In scheduling problems, feasible solutions can be usually defined by a subset of variables (e.g. $seq_{p,o}$), and, in general, it is possible to construct feasible solutions by searching into valid values of strategic variable-subsets. So that, as indicated in table 6.2, without much information about the solution quality, feasible scheduling answers were attained by searching over the $seq_{p,o}$ variable-domain (see expression 6.24). At present, the optimality proof remains a challenging feature for the CLP approach. However, it is also important to highlight that the size of variables' domain has great influence on the solution time of general CLP models, mainly if the constraints cannot rule out large parts of the search space (Heipcke, 1999). At the considered CLP model, an increase on H value causes that the domains of some variables (e.g. $dpres$, $p.e_{p,o}$, and $p.s_{p,o}$) be increased. As indicated in table 6.1, the CLP model tends to exploit a large number of nodes as the domains of variables are increased, and, therefore, the running time becomes prohibitive.

Unlike the CLP formulation, both the MILP and the CLP-MILP approaches demanded a reasonable computational effort (seconds to few minutes). In particular, table 6.1 presents the MILP computational results obtained for the OPL Studio 3.6.1, running the well-known CPLEX MILP solver (ILOG, 2001d). In chapter 5, the same MILP model is implemented and solved in the modeling and optimization tool Extended LINGO/PC Release 8.0 (LINDO, 2002), and the computational results are presented in table 5.3. The same Pentium 4, 2.4 GHz, 1 Gbyte RAM was used for running either OPL or LINGO. In both cases, the computational times were reasonable, however, OPL presented, in the great majority of instances, smaller times than LINGO. In fact, if one compares the same instances simulated either in LINGO or in OPL, the former spent an average time of 172.5 s (value calculated in table 5.3, column CT (s), for H as an integer) and, the latter, 61.8 s (average value calculated in table 6.1, column "Optimality proved in (s)", sub-column MILP). Therefore, in average, OPL was **2.8 times faster** than LINGO in proving a solution optimality.

Table 6.1 also allows a comparison between the MILP and the CLP-MILP computational results. If one takes the column "Optimum found in (s)", sub-column MILP, the average time spent by the MILP model, for each H value, is equal to 27.3 s; in the sub-column CLP-MILP, the average time is equal to 5.9 s. Therefore, the CLP-MILP model found the optimal solution, in average, **4.6 times faster** than the MILP model. Now, taking the column "Optimality proved in (s)", sub-columns MILP and CLP-MILP, the average times are equal to, respectively, 61.8 s and 28.9 s.

Thus, in average, the CLP-MILP model gave the final scheduling answer **2.1 times faster** than the MILP model. Hence, table 6.1 indicates that the CLP-MILP model tends to find the optimal solution instance, and prove its optimality, even faster than the MILP model. According to Heipcke (1999) a combined CLP-MILP approach can suffer from inherited drawbacks from the root techniques. At present, the isolated CLP model does not present reasonable computational results as the H value increases (e.g. $H > H^{min}+6\,\mathrm{h}$). It is likely that improvements in such CLP model can also cause improvements in the CLP-MILP model computational results.

Now, if one compares the CLP and the CLP-MILP models, the "Nodes Exploited" column indicates that the CLP-MILP version tends to exploit fewer nodes than the CLP model. The search mechanisms applied by both approaches are essentially the same, in exception of the auxiliary bounds provided by the MILP formulation in a combined CLP-MILP approach (section 6.3, page 123, further explains this fact). Therefore, it is evident that the linear relaxation provided by the MILP model decisively aids the search space pruning at this CLP-MILP model. In particular, the mathematical programming community has developed effective techniques for calculating lower bounds on relaxed minimization models since the sixties (Dantzig, 1963). The integration of these techniques in CLP search procedures has, obviously, a large potential.

6.4.2 Second Computational Experiment

In the first computational experiment (section 6.4.1), the three different *Main Model* versions (MILP, CLP, and CLP-MILP approaches) were simulated in a realistic pipeline-scheduling scenario, which demanded running times of seconds to few minutes, at least for the MILP and CLP-MILP approaches (details are given in table 6.1). In this section (section 6.4.2), the three *Main Model* versions are tested in some hypothetical problem instances. Such instances do not necessarily represent typical operational scenarios. In fact, the main goal is to further investigate the computational effort trend presented by the different *Main Model* versions in theoretically more time-consuming problem instances. The computational results are reported in table 6.3 (page 140). The main parameters used during section 6.4.2 simulations are herein stated in (i) to (vi):

(i) The number of demanded products is progressively increased from 8 to 12 ($8 \leq \sum_{o \in O} np_o \leq 12$);

(ii) The model is solved for fixed product demands, such that $dem_{p,o} = 1800$ m^3 $\forall o \in O$, $p \in P_o$;

(iii) For each different set of demanded products ($\sum_{o \in O} np_o = 8, 9, \dots, 12$), the scheduling horizon varies in the following interval: $H^{min} \leq H \leq H^{min} + 6\,\mathrm{h}$;

(iv) The ne value varies ($ne = 4$ or $ne = 5$) according to the available scheduling horizon (see figure 5.6, page 102);

(v) The *plug* necessity follows the hypothetical rule: $plug_{p,pa} = 1$ if $p + pa$ is an odd number, and $plug_{p,pa} = 0$ if $p + pa$ is an even number. For example, if $np_o = np_{\bar{o}} = 6$, then $plug_{1,pa} = 1$ for $pa = 2, 4, 6, 8, 10, 12$, and $plug_{1,pa} = 0$ for $pa = 1, 3, 5, 7, 9, 11$;

(vi) The simulation parameters that were not mentioned in statements (i) to (v) are maintained as originally presented in section 5.5.1.

Table 6.3 provides information about the optimization structure simulation on a Pentium 4, 2.4 GHz, 1 Gbyte RAM. The modeling and optimization tool ILOG OPL Studio 3.6.1 (ILOG, 2002b) is used to implement and solve the optimization structure. In addition, the functionalities of OPLScript (ILOG, 2002a), a script language for composing and controlling optimization models, were exploited in order to solve several instances of different *Main Model* versions (MILP, CLP, and CLP-MILP approaches).

In table 6.3, H is the scheduling horizon, ne is the number of on-peak electric time intervals, np_o and $np_{\bar{o}}$ indicate the number of demanded products in origins o and $\bar{o}$, while $\sum_{o \in O} np_o$ represents the total number of demanded products. The set of parameters H, ne, np_o, and $np_{\bar{o}}$ represents a specific problem instance. The MILP label refers to the formulation presented in section 5.4, the CLP label refers to the formulation explained in section 6.2, and the CLP-MILP label refers to the formulation presented in section 6.3. For each set of H, ne, np_o, and $np_{\bar{o}}$ values, the optimization structure is run, and a specific cost is attained (objective function value). The *Auxiliary Routine* and the *Tank Bound* simulation data are neglected. These structures required a computational time lower than one second, for all illustrative instances. No uncertainties are considered. The model is solved for fixed product demands. The columns named "Number of Variables" and "Number of Constraints" indicate the number of variables and constraints presented by the MILP, CLP, and CLP-MILP models. The column "Optimality proved in (s)" indicates the processing time (in seconds) spent by each optimization approach to prove the solution optimality. A dash (-) indicates that the *Main Model* was not run to optimality to the specific problem instance, and the computational data are neglected. In fact, the simulation was aborted after 72 hours of running (259200 seconds). The column "Average time" indicates the average computational time, displayed in format hours:minutes:seconds (hh:mm:ss). This time is calculated based on the seven different instances that are run when the $\sum_{o \in O} np_o$ remains constant. As an example, for $\sum_{o \in O} np_o = 8$, $H = 96$, 97, 98, 99, 100, 101, and 102 h, the "Optimality proved in (s)" column indicates that the CLP-MILP times were, respectively, 5.2, 7.3, 7.1, 14, 12, 12, and 23 s. Thus, the average CLP-MILP time for $\sum_{o \in O} np_o = 8$ can be calculated as $\frac{5.2+7.3+7.1+14+12+12+23}{7} \approx 12$ s. The column "Cost" indicates objective function values, according to different problem instances.

In addition to results provided by table 6.3, another fact regarding the number of demanded *plugs* was observed during simulations: the optimization structure determined pumping sequences (for each set of H, ne, np_o, and $np_{\bar{o}}$ values) where two (2) *plugs* were included. For instance, at $H = 96$ h, $ne = np_o = np_{\bar{o}} = 4$, the determined pumping sequence is $p6$, $p8$, $p5$, and $p7$ (*reflow* procedure); $p1$, $p3$, $p2$, and $p4$ (*flow* procedure). According to statement (v) on page 138, this pumping sequence implies the use of *plugs* between $p8/p5$ and between $p3/p2$. Therefore, the total number of demanded *plugs* equals two[11].

[11]This fact is going to be exploited in section 6.4.3 – see assumption (b) on page 142.

Table 6.3: Main Model Illustrative Instances II – MILP, CLP, and CLP-MILP Formulations.

H (h)	ne	np_o	$np_{\bar{o}}$	$\sum_{o \in O} np_o$	Number of Variables			Number of Constraints			Optimality proved in (s)			Average time (hh:mm:ss)			Cost ($)
					MILP	CLP	CLP-MILP	MILP	CLP	CLP-MILP	MILP	CLP	CLP-MILP	MILP	CLP	CLP-MILP	
96	4	4	4	8	361	273	417	1132	571	1620	23	0.7	5.2				158
97	4	4	4	8	361	273	417	1132	571	1620	41	6.7	7.3				154
98	4	4	4	8	361	273	417	1132	571	1620	27	46	7.1				150
99	4	4	4	8	361	273	417	1132	571	1620	61	231	14	00:00:50	00:26:50	00:00:12	146
100	4	4	4	8	361	273	417	1132	571	1620	52	397	12				143
101	4	4	4	8	361	273	417	1132	571	1620	53	635	12				140
102	4	4	4	8	361	273	417	1132	571	1620	91	9952	23				137
102	4	5	4	9	418	319	481	1302	679	1891	256	3.2	25				164
103	4	5	4	9	418	319	481	1302	679	1891	386	42	36				160
104	4	5	4	9	418	319	481	1302	679	1891	548	332	35				156
105	4	5	4	9	418	319	481	1302	679	1891	498	1570	73	00:10:19	03:38:45	00:01:05	152
106	4	5	4	9	418	319	481	1302	679	1891	717	2789	64				149
107	4	5	4	9	418	319	481	1302	679	1891	721	4514	64				146
108	4	5	4	9	418	319	481	1302	679	1891	1206	82626	157				143
108	4	5	5	10	476	366	546	1479	792	2174	4908	23	166				170
109	4	5	5	10	476	366	546	1479	792	2174	5834	245	222				166
110	4	5	5	10	476	366	546	1479	792	2174	9306	2124	224				162
111	4	5	5	10	476	366	546	1479	792	2174	16752	10819	431	03:17:32	11:20:09	00:06:24	158
112	4	5	5	10	476	366	546	1479	792	2174	16890	19848	378				155
113	4	5	5	10	476	366	546	1479	792	2174	11069	32491	379				152
114	4	5	5	10	476	366	546	1479	792	2174	18202	220110	891				149
114	4	6	5	11	538	417	615	1664	917	2477	103243	356	2186				176
115	5	6	5	11	626	461	703	1925	998	2804	133381	29477	2934				177
116	5	6	5	11	626	461	703	1925	998	2804	111275	186635	2877				178
117	5	6	5	11	626	461	703	1925	998	2804	125112	-	5916	39:33:31	-	01:28:34	179
118	5	6	5	11	626	461	703	1925	998	2804	169996	-	5907				176
119	5	6	5	11	626	461	703	1925	998	2804	176697	-	5768				173
120	5	6	5	11	626	461	703	1925	998	2804	177174	-	11607				170
120	5	6	6	12	697	517	781	2140	1135	3148	-	-	21324				197
121	5	6	6	12	697	517	781	2140	1135	3148	-	-	24808				193
122	5	6	6	12	697	517	781	2140	1135	3148	-	-	24213				189
123	5	6	6	12	697	517	781	2140	1135	3148	-	-	49254	-	-	12:42:49	185
124	5	6	6	12	697	517	781	2140	1135	3148	-	-	50233				182
125	5	6	6	12	697	517	781	2140	1135	3148	-	-	50020				179
126	5	6	6	12	697	517	781	2140	1135	3148	-	-	100530				176

6.4.2.1 Results: Commentaries on Table 6.3

In essence, the experiments conducted in section 6.4.2 are aimed at comparing the computational effort demanded by methodologies MILP, CLP, and CLP-MILP in different problem instances. Table 6.3 reports the results of such computational comparisons. In particular, the "Average time" column of table 6.3 indicates the MILP, CLP, and CLP-MILP computational effort trend, according to the considered problem instances. Figure 6.1 is plotted in order to enable the results of such trend to be visualized, and it shows the average computational time as a function of the total number of demanded products ($\sum_{o \in O} np_o$). In figure 6.1 the CLP, MILP, and CLP-MILP tendency curves are plotted. These curves are approximated by exponential functions of the form $y = ce^{bx}$, where c and b are constants, and e is the natural logarithm base ($e \approx 2.7182818\ldots$).

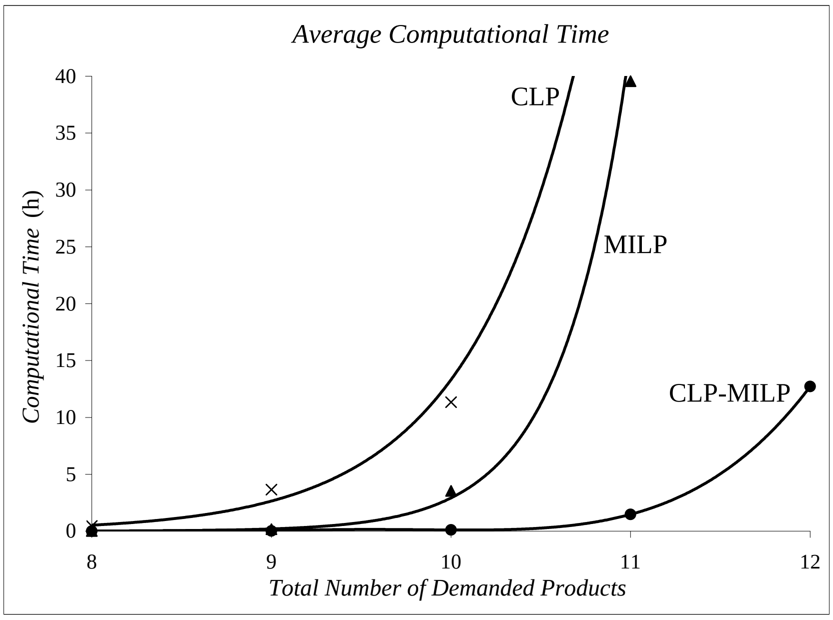


Figure 6.1: Average Computational Time for MILP, CLP, and CLP-MILP Models.

Figure 6.1 clearly indicates that, in average, the combined CLP-MILP model tends to solve this specific problem faster than the MILP and the CLP models, as the total number of demanded products increased. Obviously, one might argue that even the CLP-MILP model has an exponential-time computational behavior, but, for sure, the combined model could "go a step further" than both, the MILP and the CLP models. In addition, the search component applied by the CLP and the CLP-MILP approaches is essentially the same. However, in the CLP-MILP search mechanism, the auxiliary bounds provided by the MILP formulation are indeed used (section 6.3 further explains this fact). Therefore, when comparing the CLP and the CLP-MILP models, one can infer that the latter perform better due to the aid of linear programming bounds on the search mechanism.

6.4.3 Third Computational Experiment

Figure 6.1 indicates the average computational time presented by the CLP, MILP, and CLP-MILP models herewith developed. Such figure highlights the CLP-MILP computational gain over the root techniques, as the problem instances ($\sum_{o \in O} np_o$) are progressively increased. Nevertheless, even the combined approach presents an exponential-time tendency. Therefore, one might argue whether there is an alternative procedure to reduce the computational burden. In particular, this section (section 6.4.3) investigates the use of "heuristic" information to aid the search process. Such heuristic is based in two main assumptions:

(a) The pipeline is maintained pressurized and idle during high electric cost intervals;

(b) The number of *plugs* used during pumping procedures is known a priori.

The computational data of section 6.4.2 are also used within this experiment. So that, the parameters (i) to (vi) stated on page 138 should be observed. In particular, parameter (iii) establishes that for each different set of demanded products ($\sum_{o \in O} np_o = 8, 9, \dots, 12$), the scheduling horizon varies in the interval $H^{min} \leq H \leq H^{min} + 6\,\text{h}$. According to the experiments of section 5.5.2 (page 103), the optimization structure indeed maintains the pipeline pressurized and idle during on-peak demand intervals, as far as possible. Therefore, experiments of section 5.5.2 help understand assumption (a). Moreover, the second computational experiment (page 137) indicated that, for the hypothetical data of section 6.4.2, the demanded number of *plugs* equals two. Therefore, the number of demanded *plugs* is known a priori, as stated by assumption (b). The three models are informed about assumptions (a) and (b) as follows:

- Constraints 6.85 and 6.86 are added to the MILP formulation;
- The search component is modified in the CLP model, as indicated in formulation 6.87. Therefore, the search tree is created based on the OPL's built-in command `generate`, which receives the discrete variable $seq_{p,o}$ and generates values for the specified variables, respecting the domain conditions. However, just the generated sequences that verify equation 6.86 are accepted. Furthermore, within the accepted sequences, the *dpres* value is set to $H - H^{min}$. Obviously, the search tree defined by formulation 6.87 differs from the one generated by formulation 6.24. In the CLP model, constraints 6.85 and 6.86 are not explicitly added to the formulation, but are indeed tested during the search tree, according to 6.87;
- The CLP-MILP approach inherits constraints 6.85, 6.86, and the search component defined by formulation 6.87.

$$dpres = H - H^{min} \tag{6.85}$$

$$\begin{aligned}&\sum_{o\in O}\sum_{p\in P_o}\sum_{\substack{pa\in P_o,\\ pa\neq p}} t_{p,pa,o}\cdot plug_{p,pa} \quad + \quad \sum_{\substack{o=d,\\ p\in P_o}}\sum_{\substack{\bar{o}=\bar{d},\\ pa\in P_{\bar{o}}}} tw_{p,pa}\cdot plug_{p,pa} \quad + \\ &\sum_{\substack{p=pin,\\ o=d,\\ pa\in P_o}} first_{p,o}\cdot plug_{p,pa} \quad = \quad 2\end{aligned} \tag{6.86}$$

```
generate($seq_{p,o}$)   $\forall o \in O,\ p \in P_o$
  If   $equation$ 6.86 $verifies$
  Then
      $dpres = H - H^{min};$
      $generate\ the\ node\ to\ be\ exploited$
  Else   $do\ not\ generate\ the\ node$
```
(6.87)

Table 6.4 provides information about the optimization structure simulation on a Pentium 4, 2.4 GHz, 1 Gbyte RAM. In table 6.4, H is the scheduling horizon, ne is the number of on-peak electric time intervals, np_o and $np_{\bar{o}}$ indicate the number of demanded products in origins o and $\bar{o}$, while $\sum_{o \in O} np_o$ represents the total number of demanded products. The set of parameters H, ne, np_o, and $np_{\bar{o}}$ represents a specific problem instance. The MILP label refers to the formulation presented in section 5.4, the CLP label refers to the formulation explained in section 6.2, and the CLP-MILP label refers to the formulation presented in section 6.3. For each set of H, ne, np_o, and $np_{\bar{o}}$ values, the optimization structure is run, and a specific cost is attained (objective function value). The *Auxiliary Routine* and the *Tank Bound* simulation data are neglected. These structures required a computational time lower than one second, for all illustrative instances. No uncertainties are considered. The model is solved for fixed product demands.

The columns named "Number of Variables" and "Number of Constraints" indicate the number of variables and constraints presented by the MILP, CLP, and CLP-MILP models. The column "Optimality proved in (s)" indicates the processing time (in seconds) spent by each optimization approach to prove the solution optimality. A dash (-) indicates that the *Main Model* was not run to optimality to the specific problem instance, and the computational data are neglected. In fact, the simulation was aborted after 72 hours of running (259200 seconds). The column "Average time" indicates the average computational time, displayed in format hours:minutes:seconds (hh:mm:ss). This time is calculated based on the seven different instances that are run when the $\sum_{o \in O} np_o$ remains constant. As an example, for $\sum_{o \in O} np_o = 8$, $H = 96$, 97, 98, 99, 100, 101, and 102 h, the "Optimality proved in (s)" column indicates that the CLP-MILP times were, respectively, 1.4, 1.6, 1.6, 2.1, 2.1, 2.3, and 3.2 s. Thus, the average CLP-MILP time for $\sum_{o \in O} np_o = 8$ can be calculated as $\frac{1.4+1.6+1.6+2.1+2.1+2.3+3.2}{7} \approx 2$ s. In addition, the "Average time" column informs (between parenthesis) the average processing time value attained in the second computational experiment (page 137), and reported in table 6.3. The column "Cost" indicates objective function values, according to different problem instances.

Table 6.4: Main Model Illustrative Instances III – MILP, CLP, and CLP-MILP Formulations.

H	ne	np_o	$np_{\bar{o}}$	$\sum_{o \in O} np_o$	Number of Variables			Number of Constraints			Optimality proved in (s)			Average time (hh:mm:ss)			Cost
(h)					MILP	CLP	CLP-MILP	MILP	CLP	CLP-MILP	MILP	CLP	CLP-MILP	MILP	CLP	CLP-MILP	($)
96	4	4	4	8	361	273	417	1134	571	1622	7.3	0.2	1.4				158
97	4	4	4	8	361	273	417	1134	571	1622	9.3	0.8	1.6				154
98	4	4	4	8	361	273	417	1134	571	1622	13	4.9	1.6				150
99	4	4	4	8	361	273	417	1134	571	1622	20	23	2.1	00:00:17	00:02:35	00:00:02	146
100	4	4	4	8	361	273	417	1134	571	1622	25	37	2.1	(00:00:50)*	(00:26:50)	(00:00:12)	143
101	4	4	4	8	361	273	417	1134	571	1622	24	56	2.3				140
102	4	4	4	8	361	273	417	1134	571	1622	23	961	3.2				137
102	4	5	4	9	418	319	481	1304	679	1893	109	0.6	5.5				164
103	4	5	4	9	418	319	481	1304	679	1893	154	3.2	6.2				160
104	4	5	4	9	418	319	481	1304	679	1893	121	22	6.2				156
105	4	5	4	9	418	319	481	1304	679	1893	136	102	8.6	00:03:11	00:14:09	00:00:08	152
106	4	5	4	9	418	319	481	1304	679	1893	364	166	8.2	(00:10:19)	(03:38:45)	(00:01:05)	149
107	4	5	4	9	418	319	481	1304	679	1893	162	255	8.2				146
108	4	5	4	9	418	319	481	1304	679	1893	290	5393	13				143
108	4	5	5	10	476	366	546	1481	792	2176	950	2.0	19				170
109	4	5	5	10	476	366	546	1481	792	2176	1041	11	22				166
110	4	5	5	10	476	366	546	1481	792	2176	803	88	21				162
111	4	5	5	10	476	366	546	1481	792	2176	1807	419	30	00:21:49	01:06:42	00:00:28	158
112	4	5	5	10	476	366	546	1481	792	2176	1260	721	27	(03:17:32)	(11:20:09)	(00:06:24)	155
113	4	5	5	10	476	366	546	1481	792	2176	1946	1119	28				152
114	4	5	5	10	476	366	546	1481	792	2176	1356	25653	47				149
114	4	6	5	11	538	417	615	1666	917	2479	23737	13	115				176
115	5	6	5	11	626	461	703	1927	998	2806	25853	662	146				177
116	5	6	5	11	626	461	703	1927	998	2806	58996	3928	143				178
117	5	6	5	11	626	461	703	1927	998	2806	21507	4898	204	07:47:42	07:58:46	00:03:14	179
118	5	6	5	11	626	461	703	1927	998	2806	24142	7176	203	(39:33:31)	(-)	(01:28:34)	176
119	5	6	5	11	626	461	703	1927	998	2806	12978	10768	205				173
120	5	6	5	11	626	461	703	1927	998	2806	29323	173634	345				170
120	5	6	6	12	697	517	781	2142	1135	3150	-	63	530				197
121	5	6	6	12	697	517	781	2142	1135	3150	-	249	563				193
122	5	6	6	12	697	517	781	2142	1135	3150	-	2732	557				189
123	5	6	6	12	697	517	781	2142	1135	3150	-	18483	799	-	-	00:12:45	185
124	5	6	6	12	697	517	781	2142	1135	3150	-	-	820	(-)	(-)	(12:42:49)	182
125	5	6	6	12	697	517	781	2142	1135	3150	-	-	816				179
126	5	6	6	12	697	517	781	2142	1135	3150	-	-	1270				176

*The values between parenthesis indicate the average time for the second computational experiment (see table 6.3).

6.4.3.1 Results: Commentaries on Table 6.4

In essence, the experiments conducted in section 6.4.3 are aimed at verifying the computational effect of adding heuristic information - assumptions (a) and (b) - into MILP, CLP, and CLP-MILP developed models. Therefore, comparing the simulations of section 6.4.3 with section 6.4.2, where no heuristic information is used, is a must. In a simplified way, table 6.4 reports such computational comparisons.

The "Average time" column of table 6.4 indicates the MILP, CLP, and CLP-MILP computational effort trend, according to the considered problem instances. In addition, the same column informs (between parenthesis) the section 6.4.2 correspondent computational results. For all cases, section 6.4.3 presented smaller running times than the equivalent simulation of section 6.4.2. Therefore, table 6.4 indicates that, in average, there existed a computational time reduction when comparing simulations of sections 6.4.3 and 6.4.2. Figure 6.2 is plotted in order to enable the results of such computational time reduction to be visualized. For example, if one takes the sub-column CLP-MILP, with $\sum_{o \in O} np_o = 12$, then, the average time reduction can be calculated as $\frac{12:42:49}{00:12:45} = \frac{45769\,s}{765\,s} \approx 60$. Figure 6.2 shows the average time reduction as a function of the total number of demanded products ($\sum_{o \in O} np_o$).

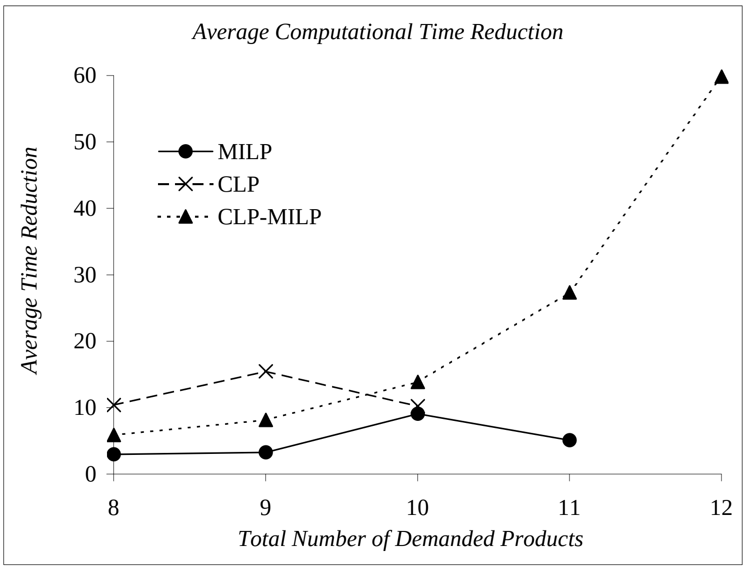


Figure 6.2: Average Computational Time Reduction when Comparing the Third Experiment with the Second Experiment.

In the MILP model, constraints 6.85 and 6.86 were added, and helped prune the traditional MILP search tree. By table 6.4, the MILP average time reduction can be calculated as $(\frac{00:00:50}{00:00:17} + \frac{00:10:19}{00:03:11} + \frac{03:17:32}{00:21:49} + \frac{39:33:31}{07:47:42})/4 \approx 5.1$. Thus, the MILP model informed about assumptions (a) and (b) was, in average, 5.1 times faster than the MILP model with no heuristic information added. In the CLP model, the search component was altered, and indeed helped reduce the processing time by an average factor of $(\frac{00:26:50}{00:02:35} + \frac{03:38:45}{00:14:09} + \frac{11:20:09}{01:06:42})/3 \approx 12$. In the CLP-MILP model, the average processing time reduction can be calculated as $(\frac{00:00:12}{00:00:02} + \frac{00:01:05}{00:00:08} + \frac{00:06:24}{00:00:28} + \frac{01:28:34}{00:03:14} + \frac{12:42:49}{00:12:45})/5 \approx 23$. However, a different time reduction behavior can be observed in the CLP-MILP model. In theory, the three models were informed about assumptions (a) and (b), but, as indicated in figure 6.2, the CLP-MILP approach presented a computational time reduction more expressive than the root techniques, as the total number of demanded products increased. For theoretic more time-consuming instances (e.g. $\sum_{o \in O} np_o = 12$), the CLP-MILP model tends to present an average computational time reduction greater than the average time reduction presented in theoretic less time-consuming instances (e.g. $\sum_{o \in O} np_o = 8$). In fact, this is a very promising characteristic because more difficult instances tend to have a more expressive computational time reduction. Thus, using the problem knowledge (heuristic information) in order to reduce the computational burden seems to be a valid procedure, mainly in the CLP-MILP model.

6.5 Remarks on Chapter 6

The main goal of chapter 6 was to demonstrate the creation of a combined CLP-MILP modeling approach where both parts of the combined model are generated through the same basis: a set of high-level MILP modeling structures with equivalent CLP modeling devices. The importance of building a combined model instead of just creating either MILP or CLP models was indeed evidenced by the computational results (section 6.4). The CLP-MILP model presented, in average, smaller running times than the root techniques. Numerical details are given in tables 6.1 (page 134), 6.3 (page 140), and 6.4 (page 145).

Chapter 7

Conclusions/Contributions and Future Research

7.1 Conclusions/Contributions

The struggle to model and solve Combinatorial Optimization Problems (COPs) has challenged the development of new approaches to deal with them. In one of the front lines of such approaches, Operational Research (OR) and Constraint Programming (CP) optimization techniques are converging, despite their very different origins. More specifically Mixed Integer Linear Programming (MILP) and Constraint Logic Programming (CLP) are at the confluence of the OR and the CP fields. It is conceivable that the union of CLP and MILP in a unique CLP-MILP framework can bring an alternative approach to solve problems that could not be addressed by none of the root techniques (Hooker, 2002). In particular, this thesis considers the integration of CLP and MILP in a modeling standpoint: it conveys the fundamentals of both techniques, and the modeling features that help establish a combined CLP-MILP approach.

Chapters 1 to 3 rise a quite large body of evidences that MILP and CLP are merging. In addition, the literature surveyed in these chapters indicates that the combined CLP-MILP approach is a promising alternative to deal with hard combinatorial optimization problems. Nevertheless, chapters 1 to 3 are not aimed at being a comprehensive review of neither CLP nor MILP, but rather an introductory discussion of opportunities for the integration of these techniques in a combined CLP-MILP approach.

> ***Contribution 1: The development of a literature survey about opportunities for the integration of CLP and MILP techniques in a combined CLP-MILP approach.***

CLP is well-known by its rich modeling framework, mainly global constraints. On the other hand, the MILP modeling vocabulary is based on inequalities, which makes the modeling process hard and error-prone. Chapter 4 addresses this issue, and it presents high-level MILP modeling structures based on logical inference paradigms. Some of these structures are literature-based and others are herein proposed. Such high-level structures help the formulation of MILP models, providing an approach to narrow the gap between CLP and MILP modeling devices. However, chapter 4 just addressed the formulation of MILP structures, and the development of algorithms to encapsulate such high-level structures is a completely different matter.

> ***Contribution 2: The development of a set of high-level MILP modeling structures based on logical inference paradigms.***

Chapter 5 presents the development of an MILP formulation addressing a combinatorial problem. This model was focused on issues regarding the oil industry, more specifically, involving the scheduling of operational activities in a multi-product pipeline. The main goal was to demonstrate the applicability of the high-level MILP modeling structures developed in chapter 4 rather than to fully exploit the problem scheduling details. Problems concerning pipeline-management were already addressed by the author of this thesis, as it can be seen, for instance, in Magatão (2001). Chapter 5 evidenced two important issues concerning the high-level structures proposed in chapter 4, in particular the if-then and the if-then-else statements: (i) *they indeed worked to model a real-world problem*; (ii) *they decisively aided the MILP formulation of such model.*

Contribution 3: The development and simulation of a novel continuous time approach MILP model addressing a specific pipeline-scheduling problem.

Chapter 6 presents a CLP and a combined CLP-MILP formulation addressing the same problem exploited in chapter 5. The main goal is to demonstrate the applicability of the high-level MILP modeling structures developed in chapter 4 in an integrated CLP-MILP modeling framework. The functionalities of the commercial tool ILOG OPL Studio 3.6.1 (ILOG, 2002b) were used in order to implement and solve the root MILP[1] and CLP models and also the combined CLP-MILP approach.

Contribution 4: The development of a CLP model addressing a specific pipeline-scheduling problem.

Contribution 5: The development of a combined CLP-MILP model addressing a specific pipeline-scheduling problem.

Chapter 6 exploited some computational characteristics of the combined approach over the root techniques. Section 6.4 brings a discussion about the computational results presented by the particular studied models. In general lines, the combined CLP-MILP formulation presented, in most simulation instances, smaller running times than either the CLP model or the MILP model. Numerical details are given in tables tables 6.1 (page 134), 6.3 (page 140), and 6.4 (page 145).

Contribution 6: The test of MILP, CLP, and CLP-MILP models in a series of specific problem instances, providing numerical comparisons amongst the three models.

Section 4.5 stated that the high-level MILP modeling structures, which were widely used in the CLP-MILP formulation, do not necessarily achieve the most computationally efficient model, that is, they do not provide the sharpest MILP formulation. However, the numerical results (section 6.4) demonstrate that, at this

[1]The MILP formulation was previously presented in chapter 5.

CLP-MILP formulation, the non-sharp MILP linear relaxation played a significant role, and indeed contributed to reduce the computational expense. Chapter 6 also evidenced one of the remarks previously acclaimed in section 4.5 (page 73), which is herewith quoted:

> *"Chapters 1 to 3 stated that the CLP modeling features overcome the MILP modeling features, which are essentially based on inequalities. However, in an integrated CLP-MILP approach, the MILP modeling, which is a hard task (see section 4.1), is indispensable. Sections 4.2 to 4.3 present connections between logical inference and MILP, which can be used to facilitate the MILP modeling task. Thus, the high-level MILP structures presented in chapter 4 can be seamlessly used in an integrated CLP-MILP modeling framework. The overall CLP-MILP modeling approach gains versatility, since CLP is well-known by its rich modeling framework, and the high-level MILP structures also facilitate the model builder task."*

Therefore, based on a practical problem, the modeling techniques presented in chapter 4 could be used in a CLP-MILP formulation. In addition, the practical use of such high-level structures evidenced that the resulting model tends to be simpler to understand and modify than the traditional MILP inequality-based model[2]. This issue was already exploited in an illustrative example of flexible storage, presented in section 4.4. Therefore, it can be stated that, somehow, the high-level structures bring "robustness" to the MILP modeling process.

> ***Contribution 7: The development of a "Logical Mixed Integer Programming (L-MILP)" framework, that is, an MILP formulation based on a set of high-level MILP modeling structures demonstrated in chapter 4.***

Another important issue of creating a combined CLP-MILP formulation concerns the compatibility between CLP and MILP modeling structures. In fact, the

[2]The referred inequality-based MILP model is presented by Magatão et al. (2004).

model builder should be aware that, in order to provide a better support for the CLP-MILP formulation to be created, one should try to use CLP structures with correspondent high-level MILP statements. Without the high-level MILP statements, the gap between CLP and MILP modeling devices becomes too large: it ranges from CLP global constraints to MILP bare inequalities.

> ***Contribution 8: The introduction of a combined CLP-MILP modeling approach where both parts of the combined model can be created through the same basis: a set of high-level MILP modeling structures with equivalent CLP modeling devices.***

The previous chapters evidenced that the combined CLP-MILP approach is a promising alternative to deal with combinatorial problems. However, the reader must be aware about some particular negative features of this combined approach:

- The model builder must address both formulations, or, at least, a hybrid model with CLP and MILP expressions;
- Just a few commercial solvers that deal with CLP and MILP modeling devices in an intertwined kernel are available (details are given in chapters 2 and 3);
- User-written applications for the hybrid technology tend to be hard to implement, because they demand algorithm knowledge about CLP and MILP structures, besides the additional effort to coordinate the different search mechanisms (Heipcke, 1999).

7.2 Future Research

This thesis focuses on modeling, and how a rethinking on modeling traditions can aid the formulation of either MILP or CLP-MILP approaches. The development of algorithms to improve the communication between CLP and MILP devices was beyond the research scope. This technical feature was circumvented by the use of an optimization tool (ILOG, 2002b) that lets the user state integrated CLP-MILP applications without the complexities of ordinary programming languages. Therefore, incoming work tends to be geared towards the directions herewith presented:

- In a practical point of view, the pipeline-scheduling formulation developed in chapters 5 and 6 can be improved. According to operational necessities, other scheduling details may be addressed. In fact, a formulation that incorporates origin and destination features in an intertwined arrangement, mass balance, due-dates (or time-windows) for sending/receiving products would have been desirable.

- The high-level MILP modeling structures of chapter 4 can be also applied to other (combinatorial) optimization problems, giving rise to "Logical Mixed Integer Linear Programming (L-MILP)" models. These L-MILP formulations should resemble the model created in chapter 5. In addition, L-MILP models can be integrated in a CLP framework, as indicated in chapter 6. In particular, decision problems open a wide range of candidate applications for the high-level MILP structures.

- In a theoretical point of view, the study of the high-level MILP modeling structures of chapter 4 can be extended. This is a promising research field, even more if one considers that, in the near future, a single optimization statement (e.g. *alldifferent*) may be part of an integrated modeling environment, and different techniques, such as CLP and MILP, should interpret this "high-level" form of problem representation. In this case, the model does not imply a solving technology, but it is just a form of problem description. Thus, the extension of the high-level MILP structures presented in chapter 4 is an interesting alternative towards a unifying modeling framework for a combined CLP-MILP approach.

- The algorithm integration of CLP and MILP devices can be further investigated. Perhaps, a user-written CLP-MILP algorithm integration can be designed.

Bibliography

Applequist, G., Samikoglu, O., Pekny, J. F. and Reklaitis, G. V. (1997). Issues in the use, design and evolution of process scheduling and planning systems, *ISA Transactions* **36**(2): 81–121.

Baptiste, P., LePape, C. and Nuijten, W. (2001). *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*, Kluwer Academic Publishers, Massachusetts, USA.

Barták, R. (1999). Constraint programming: In pursuit of the holy grail, *Proceedings of Week of Doctoral Students (WDS99), part IV*, MatFyzPress, Charles University in Prague, Faculty of Mathematics and Physics, Department of Theoretical Computer Science, pp. 555–564.

Barták, R. (2002). Constraint-based scheduling: An introduction to newcomers, *Technical Report TR 2002/2*, Charles University in Prague, Faculty of Mathematics and Physics, Department of Theoretical Computer Science.

Barth, P. and Bockmayr, A. (1995). Modeling mixed-integer optimisation problems in constraint logic programming, *Technical Report MPI-I-95-2-011*, Max-Planck-Institut für Informatik, Saarbrücken, Germany.

Beale, E. M. L. and Tomlin, J. A. (1970). Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables, *in* J. Lawrence (ed.), *Proceedings of the Fifth International Conference on Operational Research*, Tavistock Publications, pp. 447–454.

Beldicenau, N. and Contejean, E. (1994). Introducing global constraints in CHIP, *Journal of Mathematical and Computer Modeling* **20**(12): 97–123.

Bixby, R. E. (2002). Solving real-world linear programs: A decade and more of progress, *Operations Research* **50**(1): 3–15.

Bixby, R., Fenelon, M., Gu, Z., Rothberg, E. and Wunderling, R. (2000). MIP: Theory and practice — closing the gap, *in* M. Powell and S. Scholtes (eds), *Systems Modelling and Optimization: Methods, Theory, and Applications*, Kluwer Academic Publishers, pp. 19–49.

Bockmayr, A. and Kasper, T. (1997). A unifying framework for integer and finite domain constraint programming, *Technical Report MPI-I-97-2-008*, Max-Planck-Institut für Informatik, Saarbrücken, Germany.

Bollapragada, S., Ghattas, O. and Hooker, J. N. (2001). Optimal design of truss structures by mixed logical linear programming, *Operations Research* **49**: 42–51.

Brailsford, S. C., Hubbard, P. M., Smith, B. M. and Williams, H. P. (1996). Organizing a social event — A difficult problem of combinatorial optimization, *Computers and Operations Research* **23**(9): 845–856.

Brailsford, S. C., Potts, C. N. and Smith, B. M. (1999). Constraint satisfaction problems: Algorithms and applications, *European Journal of Operational Research* **119**: 557–581.

Brearley, A. L., Mitra, G. and Williams, H. P. (1975). Analysis of mathematical programming problems prior to applying the simplex algorithm, *Mathematical Programming* **8**: 54–83.

Brooke, A. and Meeraus, A. (1982). On the development of a general algebraic modeling system in a strategic planning environment, *Mathematical Programming Study* **20**: 1–29.

Carlsson, M. and Ottosson, G. (1999). A comparison of CP, IP and hybrids for configuration problems, *Technical Report 99/04*, Uppsala University, Swedish Institute of Computer Science (SICS).

Chandru, V. and Hooker, J. N. (1999). *Optimization Methods for Logical Inference*, Wiley-Interscience Series in Discrete Mathematics and Optimization, New York, USA.

Chandru, V. and Rao, M. R. (1996). Combinatorial optimization: An integer programming perspective, *ACM Computing Surveys* **28**(1): 55–58.

Codognet, P. and Diaz, D. (1996). Compiling constraints in clp(fd), *Journal of Logic Programming* **27**(3): 185–226.

Colmerauer, A. (1987). Opening the Prolog-III universe, *BYTE magazine* **12**(9): 177–182.

Colombani, Y. and Heipcke, S. (1997). The constraint solver SchedEns. Tutorial and documentation, *Technical Report 241*, Université Aix-Marseille II, Laboratoire d'Informatique de Marseille (LIM), Marseille, France.

COPEL (2005). Companhia Paranaense de Energia Elétrica – Tarifas de energia elétrica (on-line information available at http://www.copelsolucoes.com/downloads/downloads.html in May 2005).

COSYTEC (1995). *CHIP C Library*, COSYTEC editor, Orsay, France.

COSYTEC (2005). CHIP V5: Second generation constraint programming technology (on-line information available at http://www.cosytec.com/ in May 2005).

Dantzig, G. B. (1963). *Linear Programming and Extensions*, Princeton University Press, Princeton, New Jersey, USA.

Darby-Dowman, K. and Little, J. (1998). Properties of some combinatorial optimization problems and their effect on the performance of integer programming and constraint logic programming, *INFORMS Journal on Computing* **10**(3): 276–286.

Darby-Dowman, K., Little, J., Mitra, G. and Zaffalon, M. (1997). Constraint logic programming and integer programming approaches and their collaboration in solving assignment scheduling problem, *Constraints* **1**(3): 245–264.

De-Baker, B., Furnon, V., Prosser, P., Kilby, P. and Shaw, P. (2000). Solving vehicle routing problems using constraint programming and metaheuristics, *Journal of Heuristics Special Issue on Constraint Programming* **6**(4): 501–524.

Floudas, C. A. (1995). *Nonlinear and Mixed Integer Optimization: Fundamentals and Applications*, Oxford University Press, New York, USA.

Focacci, F. (2000). *Solving Combinatorial Optimization Problems in Constraint Programming*, PhD thesis, Università degli Studi di Ferrara, Ferrara, Italia.

Fourer, R. (1998). Extending a general-purpose algebraic modeling language to combinatorial optimization, *in* D. L. Woodruff (ed.), *Advances in Computational and Stochastic Optimization, Logic Programming and Heuristic Search: Interfaces in Computer Science and Operations Research*, Kluwer Academic Publishers, Dordrecht, The Netherlands, pp. 31–74.

Fourer, R., Gay, D. M. and Kernighan, B. W. (1990). A modeling language for mathematical programming, *Management Science* **36**: 519–554.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability – A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, Bell Laboratories, Murray Hill, New Jersey, USA.

Glover, F. and Laguna, M. (1993). *Modern Heuristic Techniques for Combinatorial Optimization*, Blackwell Scientific Publications, Oxford.

Goldbarg, M. C. and Luna, H. P. L. (2000). *Otimização Combinatória e Programação Linear – Modelos e Algoritmos*, Editora Campus, Rio de Janeiro, RJ, Brazil. (in portuguese).

Grossmann, I. E., Hooker, J. N., Raman, R. and Yan, H. (1994). Logic cuts for processing networks with fixed charges, *Computers & Operations Research* **21**: 265–279.

Guéret, C., Prins, C. and Sevaux, M. (2002). *Applications of Optimization with Xpress–MP*, Dash Optimization editor, France. Translated and revised by S. Heipcke.

Hadjiconstantinou, E. and Mitra, G. (1994). A linear and discrete programming framework for representing qualitative knowledge, *Journal of Economic Dynamics and Control* **18**: 273–297.

Hajian, M. T. (1996). Dis-equality constraints in linear/integer programming, *Technical report*, IC-Parc, Imperial College, London, UK.

Hajian, M. T., El-Sakkout, H., Wallace, M., Lever, J. M. and Richards, B. (1995). Towards a closer integration of finite domain propagation and simplex-based algorithms, *Technical report*, IC-Parc, Imperial College, London, UK.

Harjunkoski, I. and Grossmann, I. E. (2002). Decomposition techniques for multistage scheduling problems using mixed-integer and constraint programming methods, *Computers & Chemical Engineering* **26**: 1533–1552.

Harjunkoski, I., Jain, V. and Grossmann, I. E. (2000). Hybrid mixed-integer/constraint logic programming strategies for solving scheduling and combinatorial optimization problems, *Computers & Chemical Engineering* **24**: 337–343.

Heipcke, S. (1999). *Combined Modelling and Problem Solving in Mathematical Programming and Constraint Programming*, PhD thesis, University of Buckingham, UK.

Hentenryck, P. V. (1989). *Constraint Satisfaction in Logic Programming*, MIT Press.

Hentenryck, P. V., Perron, L. and Puget, J. F. (2000). Search and strategies in OPL, *ACM Transactions on Computational Logic* **1**(2): 285–320.

Hermenegildo, M. (1997). Some challenges for constraint programming, *Constraints, Special Issue on Strategic Directions in Constraint Programming.* **2**(1): 63–69.

Hooker, J. N. (2000). *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction*, Wiley-Interscience Series in Discrete Mathematics and Optimization, New York, USA.

Hooker, J. N. (2002). Logic, optimization and constraint programming, *INFORMS Journal on Computing* **14**: 295–321.

Hooker, J. N. (2003). A framework for integrating solution methods, *in* H. K. Bhargava and M. Ye (eds), *Proceedings of ICS2003: Computational Modeling and Problem Solving in the Networked World*, Kluwer Academic Publishers, pp. 3–30.

Hooker, J. N. and Osorio, M. A. (1999). Mixed/logical linear programming, *Discrete Applied Mathematics* **96–97**(1–3): 395–442.

Hooker, J. N., Ottosson, G. and Thorsteinsson, E. S. (2000). A scheme for unifying optimization and constraint satisfaction methods, *Knowledge Engineering Review* **15**: 11–30.

Hürlimann, T. (1998). An efficient logic-to-IP translation procedure, *APMOD98 International Conference, Applied Mathematical Programming and Modeling*, Lymassol, Cyprus, pp. 1–26.

Hürlimann, T. (2000). Reference manual for the LPL modeling language, version 4.40, *Technical Report 00-5*, University of Fribourg, Institute of Informatics, Fribourg, Switzerland.

Ierapetritou, M. G. and Floudas, C. A. (1998). Short-term scheduling: New mathematical models vs algorithmic improvements, *Computers & Chemical Engineering* **22**(supplement): S419–S426.

ILOG (2001a). *Constraint Programming with ILOG Solver*, ILOG Corporation, Gentilly, France, pp. 33–44.

ILOG (2001b). *ILOG OPL Studio*, ILOG Corporation, Gentilly, France, pp. 14–19.

ILOG (2001c). *ILOG Optimization Suite – White Paper*, ILOG Corporation. (Available at http://www.ilog.com/ in May 2005).

ILOG (2001d). *Mathematical Programming with ILOG CPLEX*, ILOG Corporation, Gentilly, France, pp. 26–32.

ILOG (2002a). *ILOG OPL Studio 3.6.1 – Language Manual*, ILOG Corporation, France.

ILOG (2002b). *ILOG OPL Studio 3.6.1 – User's Manual*, ILOG Corporation, France.

Jaffar, J. and Maher, M. J. (1994). Constraint logic programming: A survey, *Journal of Logic Programming, Supplement 1* **19–20**: 503–581.

Jain, V. and Grossmann, I. E. (1999). Algorithms for hybrid MILP/CLP models for a class of optimization problems, *Technical report*, Carnegie Mellon University, Department of Chemical Engineering, Pittsburg, PA, USA.

Jeroslow, R. G. and Lowe, J. K. (1984). Modelling with integer variables, *Mathematical Programming Study* **22**: 167–184.

Kalvelagen, E. (2003). On solving the progressive party problem as a MIP, *Computers & Operations Research* **30**(11): 1713–1726.

Kay, P. (1997). COSYTEC white paper: CHIP example code (available at http://www.cosytec.com/ in May 2005).

Kennedy, J. L. (1993). *Oil and Gas Pipeline Fundamentals*, PennWell Publishing Company, Oklahoma, USA.

Kondili, E., Pantelides, C. C. and Sargent, R. W. H. (1993). A general algorithm for short-term scheduling of batch operations – I. MILP formulation, *Computers & Chemical Engineering* **17**(2): 211–227.

Land, A. H. and Doig, A. G. (1960). An automatic method for solving discrete programming problems, *Econometrica* **28**: 497–520.

Laurière, J. L. (1978). A language and a program for stating and solving combinatorial problems, *Artificial Intelligence* **10**: 29–127.

Lee, H., Pinto, J. M., Grossman, I. E. and Park, S. (1996). Mixed-integer linear programming model for refinery short-term scheduling of crude oil unloading with inventory management, *Industrial & Engineering Chemistry Research* **35**: 1630–1641.

LINDO (2002). *LINGO: The Modeling Language and Optimizer – User's Guide*, LINDO Systems Inc., Chicago, Illinois.

Lustig, I. J. and Puget, J. F. (2001). Program does note equal program: Constraint programming and its relationship to mathematical programming, *Interfaces* **31**(6): 29–53.

Magatão, L. (2001). *Programação matemática aplicada à otimização das operações de um poliduto*, Master's thesis, Centro Federal de Educação Tecnológica do Paraná (CEFET-PR). Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial (CPGEI), Curitiba, Paraná, Brazil. (in portuguese).

Magatão, L., Arruda, L. V. R. and Neves-Jr, F. (2001). Sequencing inputs to a multi-product pipeline, *Proceedings of the European Control Conference - ECC 2001*, Porto, Portugal, pp. 2152–2157.

Magatão, L., Arruda, L. V. R. and Neves-Jr, F. (2002). A methodology for scheduling commodities in a multi-product pipeline, *Proceedings of the XV IFAC World Congress on Automatic Control*, Barcelona, Spain.

Magatão, L., Arruda, L. V. R. and Neves-Jr, F. (2004). A mixed integer programming approach for scheduling commodities in a pipeline, *Computers & Chemical Engineering* **28**(1-2): 171–185.

Maximal (2005). Maximal software inc. – MPL modeling system (on-line information available at http://www.maximal-usa.com/ in May 2005).

McKinnon, K. I. M. and Williams, H. P. (1989). Constructing integer programming model by the predicate calculus, *Annals of Operational Research* **21**: 227–246.

Mitra, G., Lucas, C., Moody, S. and Hadjiconstantinou, E. (1994). Tools for reformulating logical forms into zero-one mixed integer programs, *European Journal of Operational Research* **72**: 262–276.

Murphy, F. H. and Panchanadam, V. (1997). Understanding linear programming modeling trough and examination of the early papers on model formulation, *Operations Research* **45**(3): 341–356.

Nemhauser, G. L. and Wolsey, L. A. (1988). *Integer and Combinatorial Optimization*, John Wiley & Sons, Inc., New York, USA.

Ottosson, G. and Carlsson, M. (1997). Using global constraints for frequency allocation, *Technical Report 97/07*, Uppsala University, Swedish Institute of Computer Science (SICS).

Ottosson, G., Thorsteinsson, E. S. and Hooker, J. N. (2001). Mixed global constraints and inference in hybrid CLP–IP solvers, *Annals of Mathematics and Artificial Intelligence, Special Issue on Large Scale Combinatorial Optimisation and Constraints* **34**(4): 271–290.

Ottosson, G., Thorsteinsson, E. S. and Hooker, J. N. (2002). Mixed global constraints and inference in hybrid CLP–IP solvers, *Annals of Mathematics and Artificial Intelligence, Special Issue on Large Scale Combinatorial Optimisation and Constraints* **34**(4): 271–290.

Pekny, J. F. and Reklaitis, G. V. (1998). Towards the convergence of theory and practice: A technology guide for scheduling/planning methodology, *American Institute of Chemical Engineering Symposium Series* **94**(320): 91–111.

Pesant, G. and Gendreau, M. (1996). A view of local search in constraint programming, *Lecture Notes in Computer Science* **1118**: 353–363.

Pinto, J. M. and Grossmann, I. E. (1997). A logic-based approach to scheduling problems with resource constraints, *Computers & Chemical Engineering* **21**: 801–818.

Pinto, J. M. and Grossmann, I. E. (1998). Assignment and sequencing models for the scheduling of chemical processes, *Annals of Operations Research* **81**: 433–466.

Proll, L. and Smith, B. (1998). Integer linear programming and constraint programming approaches to a template design problem, *INFORMS Journal on Computing* **10**(3): 265–275.

Puget, J. F. (1994). A C++ implementation of CLP, *Technical Report 94-01*, ILOG Corporation, Gentilly, France.

Raman, R. and Grossmann, I. E. (1991). Relation between MILP modeling and logical inference for chemical process synthesis, *Computers & Chemical Engineering* **15**(2): 73–84.

Raman, R. and Grossmann, I. E. (1992). Integration of logic and heuristic knowledge in MINLP optimization for process synthesis, *Computers & Chemical Engineering* **16**(3): 155–171.

Raman, R. and Grossmann, I. E. (1993). Symbolic integration of logic in mixed-integer linear programming techniques for process synthesis, *Computers & Chemical Engineering* **17**(9): 909–927.

Raman, R. and Grossmann, I. E. (1994). Modeling and computational techniques for logic based integer programming, *Computers & Chemical Engineering* **18**(7): 563–578.

Reklaitis, G. V. (1992). Overview of scheduling and planning of batch process operations, *Proceedings of the NATO Advanced Study Institute on Batch Processing Systems*, Antalya, Turkey, pp. 660–705.

Rodošek, R., Wallace, M. G. and Hajian, M. T. (1999). A new approach to integrating mixed integer programming and constraint logic programming, *Annals of Operations Research* **86**: 63–87.

Rossi, F. (1999). Constraint logic programming: A survey on research and applications., *in* K. R. Apt, A. C. Kakas, E. Monfrey and F. Rossi (eds), *New Trends in Constraints, Joint ERCIM/Compulog Net Workshop*, Vol. 1865 of *Lecture Notes in Computer Science*, Springer, pp. 40–74.

Schrage, L. (2000). *Optimization Modeling with LINGO*, LINDO Systems Inc., Chicago, Illinois.

Shah, N. (1998). Single and multisite planning and scheduling: Current status and future challenges, *American Institute of Chemical Engineering Symposium Series* **94**(320): 75–90.

Sherali, H. D. and Driscoll, P. J. (2000). Evolution and state-of-the-art in integer programming, *Journal of Computational and Applied Mathematics* **124**: 319–340.

Smith, B. M. (1995). A tutorial on constraint programming, *Technical Report 95.14*, University of Leeds, Division of Artificial Intelligence, Leeds, UK. School of Computer Studies, Research Report Series.

Smith, B. M., Brailsford, S. C., Hubbard, P. M. and Williams, H. P. (1996). The progressive party problem: Integer linear programming and constraint programming compared, *Constraints* **1**: 119–136.

Subrahmanyam, S., Bassett, M. H., Pekny, J. F. and Reklaitis, G. V. (1995). Issues in solving large scale planning, design and scheduling problems in batch chemical plants, *Computers & Chemical Engineering* **19**(supplement): S577–S582.

Thorsteinsson, E. S. (2001). *Hybrid Approaches to Combinatorial Optimisation*, PhD thesis, Carnegie Mellon University, Graduate School of Industrial Administration, Pittsburgh, Pennsylvania, USA.

Türkay, M. and Grossmann, I. E. (1996). Logic-based MINLP algorithms for the optimal synthesis of process network, *Computers & Chemical Engineering* **20**: 959–978.

Vecchietti, A. and Grossmann, I. E. (2000). Modeling issues and implementation of language for disjunctive programming, *Computers & Chemical Engineering* **24**: 2143–2155.

Wallace, M., Novello, S. and Schimpf, J. (1997). ECL^{i}PS^e: A platform for constraint logic programming, *Technical report*, IC-Parc, Imperial College, London, UK.

Williams, H. P. (1987). Linear and integer programming applied to the propositional calculus, *International Journal of Systems Research and Information Science* **2**: 81–100.

Williams, H. P. (1995). Logic applied to integer programming and integer programming applied to logic, *European Journal of Operational Research* **81**: 605–616.

Williams, H. P. (1999). *Model Building in Mathematical Programming*, John Wiley & Sons, Inc., UK.

Williams, H. P. and Wilson, J. M. (1998). Connections between integer linear programming and constraint logic programming — an overview and introduction to the cluster of articles, *INFORMS Journal on Computing* **10**(3): 261–264.

Wolsey, L. A. (1998). *Integer Programming*, John Wiley & Sons, Inc., New York, USA.

Wu, D. and Ierapetritou, M. G. (2003). Decomposition approaches for the efficient solution of short-term scheduling problems, *Computers & Chemical Engineering* **27**: 1261–1276.

Yeom, K. and Lee, J. K. (1998). Logical representation of integer programming models, *Decision Support Systems* **18**: 227–251.

Yunes, T. H., Moura, A. V. and Souza, C. C. (2000). Modeling and solving a crew rostering problem with constraint logic programming and integer programming, *Technical Report IC-00-04*, State University of Campinas, Campinas, SP, Brazil.

MINISTÉRIO DA EDUCAÇÃO
CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGª ELÉTRICA E INFORMÁTICA INDUSTRIAL

"Mixed Integer Linear Programming and Constraint Logic Programming: Towards a Unified Modeling Framework"

por

Leandro Magatão

Esta Tese de Doutorado foi apresentada no dia 12/05/2005, como requisito parcial para a obtenção do título de DOUTOR EM CIÊNCIAS – Área de Concentração: Informática Industrial. Aprovada pela Banca Examinadora composta pelos professores:

[illegible]
Prof. Dr. Lúcia Valéria Ramos de Arruda
(Orientador - CEFET-PR)

[illegible]
Prof. Dr. Celso Carnieri
(UFPR)

[illegible]
Prof. Dr. Paulo Morelato França
(UNICAMP)

[illegible]
Prof. Dr. Marcus Vinícius de Oliveira Magalhães
(PETROBRAS)

[illegible]
Prof. Dr. Flávio Neves Junior
(CPGEI/CEFET)

Visto e aprovado para impressão:

[illegible]
Prof. Dr. José Luís Fabris
(Coordenador do CPGEI)

ABSTRACT

The struggle to model and solve Combinatorial Optimization Problems (COPs) has challenged the development of new approaches to deal with COPs. In one of the front lines of such approaches, Operational Research (OR) and Constraint Programming (CP) optimization techniques are beginning to converge, despite their very different origins. More specifically, Mixed Integer Linear Programming (MILP) and Constraint Logic Programming (CLP) are at the confluence of the OR and the CP fields. This thesis summarizes and contrasts the essential characteristics of MILP and CLP, and the ways that they can be fruitfully combined. Chapters 1 to 3 sketch the intellectual background for recent efforts at integration and the main results achieved. In addition, these chapters highlight that CLP is known by its rich modeling framework, and the MILP modeling vocabulary is just based on inequalities, which makes the modeling process hard and error-prone. Therefore, a combined CLP-MILP approach suffers from this MILP inherited drawback. In chapter 4, this issue is addressed, and some “high-level” MILP modeling structures based on logical inference paradigms are proposed. These structures help the formulation of MILP models, and can be seen as a contribution towards a unifying modeling framework for a combined CLP-MILP approach. In addition, chapter 5 presents an MILP formulation addressing a combinatorial problem. This problem is focused on issues regarding the oil industry, more specifically, issues involving the scheduling of operational activities in a multi-product pipeline. Chapter 5 demonstrates the applicability of the high-level MILP modeling structures in a real-world scenario. Furthermore, chapter 6 presents a CLP-MILP formulation addressing the same scheduling problem previously exploited. This chapter demonstrates the applicability of the high-level MILP modeling structures in an integrated CLP-MILP modeling framework. The set of simulations conducted indicates that the combined CLP-MILP model was solved to optimality faster than either the MILP model or the CLP model. Thus, the CLP-MILP framework is a promising alternative to deal with the computational burden of this pipeline-scheduling problem. In essence, this thesis considers the integration of CLP and MILP in a modeling standpoint: it conveys the fundamentals of both techniques and the modeling features that help establish a combined CLP-MILP approach. Herein, the concentration is on the building of MILP and CLP-MILP models rather than on the solution process.

KEYWORDS
Mixed Integer Linear Programming (MILP), Constraint Logic Programming (CLP), Combinatorial Optimization Problems (COPs), Scheduling, Pipeline.

ÁREAS / SUB-ÁREAS DO CONHECIMENTO

3.08.02.00-8	Pesquisa Operacional
3.08.02.02-4	Programação Linear, Não-Linear, Mista e Dinâmica
3.06.03.16-1	Petróleo e Petroquímica

2005

Nº: 11