Departament de Llenguatges i Sistemes Informatics
UNIVERSITAT POLITÈCNICA DE CATALUNYA

**Master in Computing**

## Master of Science Thesis

# DEVELOPMENT OF A HYBRID METAHEURISTIC FOR THE EFFICIENT SOLUTION OF STRATEGIC SUPPLY CHAIN MANAGEMENT PROBLEMS: APPLICATION TO THE ENERGY SECTOR

Pedro Jesús Copado Méndez

Advisor: Christian Blum (UPC, Barcelona)
Co-Advisor: Gonzalo Guillén Gosálbez (URV, Tarragona)

January, 14th 2011

In memory of my uncle
Rodolfo Méndez García,
who died on
November $17^{th}$, 2010.

## Acknowledgments

I would like to acknowledge Gonzalo Guillén Gosálbez, Christian Blum and Laureano Jiménez for their support and infinite patience.

I also wish to acknowledge support of this research work from the Spanish Ministry of Education and Science (DPI2008-04099/DPI).

# Contents

# Chapter 1

# Introduction

In this chapter we provide an introduction to supply chain management (SCM) with emphasis on two types of methods for solving optimization problems in SCM: mathematical programming and metaheuristics.

## 1.1 Introduction to Supply Chain Management

Supply chain management (SCM) addresses the strategic, tactical, and operational decision making that optimizes the supply chain performance. The strategic level defines the supply chain configuration: the selection of suppliers, transportation routes, manufacturing facilities, production levels, technologies. The tactical level plans and schedules the supply chain to meet actual demand. The operational level executes plans. Tactical and operational level decision-making functions are distributed across the supply chain.

To increase or optimize performance, supply-chain functions must be perfectly coordinated. But the cycles of the enterprise and the market make this difficult: raw material does not arrive on time, production facilities fail, workers are ill, customers change or cancel orders, therefore, causing deviations from the plan. In some cases, these situations may be dealt with locally. In other cases, the problem cannot be "locally contained" and modifications across many functions are required. Consequently, the supply chain management system must coordinate the revision of plans or schedules. The ability to better understand an algorithm is important to focus on the following variables: tactical and operational levels of the supply chain so that the timely dissemination of information, accurate coordination of decisions, and management of actions among people and systems is achieved ultimately

determines the efficient, coordinated achievement of enterprise goals.

## 1.2 Solution Methods

Several solution methods have been proposed so far for dealing with the complexity associated with the SCM problem. We next provide an overview of these strategies, with emphasis on optimization tools based on mathematical programming an metaheuristics.

### 1.2.1 Mathematical Programming

Mathematical programming or optimization is a technique for finding an optimal solution of a given problem with constraints. Optimization problems can be classified in terms of continuous and discrete variables. Solution methods for solving problems that only include continuous variables are linear programming (LP) and non-linear programming (NLP). On the other side, problems on discrete variables are modelled as mixed-integer linear programming (MILP) and mixed-integer non-linear programming (MINLP) problems. Other than these major problem classes, other types of optimization problems are considered: dynamic optimization (including differential equations), stochastic programming (optimization under uncertainty), etc. These problems can be solved by different techniques, in the past most of these where based on trial and error, however in the last decades research in this area has produced systematic approaches to do this task.

In this general introduction, we will first consider a general constrained optimization problem (Bazaraa et al., 2006):

$$\begin{array}{ll} min & f(x,y) \\ s.t & h(x,y) = 0 \\ & g(x,y) \leq 0 \\ & x \in X \\ & y \in Y \end{array}$$

Hereby, $f(x,y)$ is the objective function, $x$ is the set continuous variables, and y are the integer variables, $h(x,y) = 0$ are the equality constraints and $g(x,y) \leq 0$ are the inequality constraints. Any optimization problem can be represented in this form. Note that maximizing function $f(x,y)$ is equivalent to minimizing $-f(x,y)$. Moreover, if we have inequalities grater than zero, these may be transformed by multiplying the two terms by minus one.

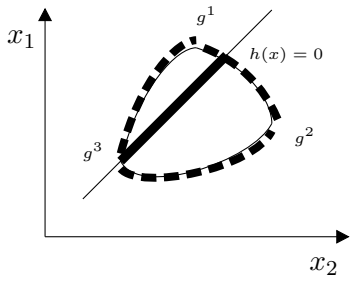


Figure 1.1: Feasible region for three inequalities and one equation.

#### 1.2.1.1 Linear Programming (LP)

In the case of only linear functions and continuous decision variables, the corresponding problem may be modelled as a linear programming (LP) problem.

$$\begin{array}{ll} min & Z = c^T \cdot x \\ s.t & A \cdot x = b \\ & C\dot{x} \leq d \\ & x \geq 0 \end{array}$$

The standard solution method to solve LP problems is the simplex method (Dantzig, 1963), although in the last decades interior point methods (Wu et al., 2002) have been extensively used for highly constrained LP problems (e.g., problems with about 100,000 constraints and variables). The major commercial solvers developed for LP optimization problems are CPLEX and OSL, which are based on the simplex method, and XPRESS which makes use of a Newton barrier interior point method.

#### 1.2.1.2 Non-linear programming (NLP)

In the case that some of the functions are non-linear and the problem does not include discrete decision variables, the problem can be modelled as a non-linear programming (NLP) model.

$$\begin{array}{ll} min & f(x) \\ s.t & h(x) = 0 \\ & g(x) \leq 0 \\ & x \in X \end{array}$$

Two major methods for NLP optimization are used; successive quadratic programming (SQP) (Powell, 1978) and the reduced gradient method (Murtagh and Saunders, 1978). Concerning the SQP algorithm, the basic idea is to solve a quadratic programming subproblem at each iteration. In contrast, the reduced gradient method solves a sequence of subproblems with linearized constraints. Comparing SQP with the reduced gradient method, SQP generally requires less iterations. However, for large-scale problems the reduced gradient method is more robust.

The main SQP solvers are SNOPT, KNITOR, IPOPT and rSQP, whereas reduced gradient method solvers are GRG2, CONOPT and MINOS.

#### 1.2.1.3 Mixed integer-linear programming (MILP)

This type of problem model is an extension of LP where some of the variables may adopt an integer value. The general form of a MILP problem is the following one:

$$\begin{array}{ll} min & Z = c^T \cdot x + b^T \cdot y \\ s.t & A \cdot x + B \cdot y \leq d \\ & x \geq 0 \\ & y \in \{0,1\}^m \end{array}$$

The principal method for solving MILPs is the LP-based branch and bound (Nemhauser and Wolsey, 1988a). This method consists of a tree enumeration where at each node a relaxed LP subproblem is solved. Another technique for MILP optimization makes use of cutting planes, a method based on generating cuts from the LP relaxation. Currently, most of the commercial solvers combine both techniques. These computer packages are, among others, CPLEX, OSL, LINDO and ZOOM.

#### 1.2.1.4 Mixed integer-nonlinear programming (MINLP)

This type of problem model has the characteristics of both NLP and MILP. The variables may adopt either real or integer values and some of the constrains or the objective function may be non-linear. The general form of a MINLP problem is the following one:

$$\begin{array}{ll} min & Z = c^T \cdot x + b^T \cdot y \\ s.t & A \cdot x + B \cdot y \leq d \\ & x \in X, y \in Y \\ & X = \{x \mid x \in R^n, x^L \leq x \leq x^U, B \cdot x \leq b\} \\ & Y = \{y \mid y \in \{0,1\}^m, A \cdot y \leq a\} \end{array}$$

There are several resolution algorithms: branch and bound (Gupta and Ravindran, 1985), branch and cut (Stubbs and Mehrotra, 1999), generalized benders decomposition (Geoffrion, 1972) and outer-approximation (Duran and Grossmann, 1986).

### 1.2.2 Incomplete Algorithms: Heuristics and Metaheuristics

In this section we start talking about heuristics, then we delve into algorithms that are known as metaheuristics. In fact, the hybridization of metaheuristics and integer linear programming is an important aspect of this work. Therefore, at the end of this section we will give a brief description of the hybridization of metaheuristics with other techniques for optimization.

#### 1.2.2.1 Combinatorial Optimization

A combinatorial optimization problem (COP), following the definition of Papadimitriou and Steiglitz, is the pair $P = (S, f)$ consisting by the finite set of objects $S$ and an objective function $f : S \to \mathbb{R}^+$, which assigns a positive value to each of the objects of $S$. The goal is to find an $s \in S$ which has a cost value $f(s)$ that is lower than (or equal to) the cost value of any other object in $S$. A well-known example of a COP is the Traveling Salesman Problem (TSP) (Lawler et al., 1985), which is defined as follows.

**Definition 1** *In the Traveling Salesman Problem (TSP) is given a complete graph $G = (V, E)$ with a weight $w_e \in \mathbb{R}^+$ for each edge $e \in E$. The goal consists in finding the minimum Hamiltonian cycle in $G$. The objective function value $f(s)$ is calculated as the sum of the weights of the edges that form the Hamiltonian cycle $s$, and hence the search space $S$ consists of all possible Hamiltonian cycles that exist in $G$.*

Note that each COP can be modelled as a MILP in various different ways. The resolution of any COP, (or any computer problem) is through an algorithm. Algorithms may be c̃omplete, *probabilistic* or *approximate*: complete methods guarantee to find an optimal solution. However, when the size of the problem instance increases, the computation time required by complete methods may be impractically high. In the case of COP problems that are $NP$-hard, no polynomial algorithm exists to solve them. Therefore, when rather large instances of $NP$-hard problems are concerned, approximate algorithms are often the only alternative. While these methods produce good

solutions in a reasonable amount of computation time, they do not guarantee to find optimal solutions. There are two basic types of approximate algorithms, constructive heuristics and local search.

#### 1.2.2.2 Constructive Heuristics

Constructive heuristics are the most typical approximate algorithms for solving COP. These algorithms build solutions from scratch, starting from an empty initial solution. They employ a *construction mechanism* which will add solution components at each step, according to a cost function, until the solution is complete or the process is stopped by another criterion.

The schema of a constructive heuristic is shown in algorithm 1. The algorithm is initialized with an empty partial solution $s^p$. Given a partial solution $s^p$, a set of solution components $cc(s^p)$ can be derived for the extension of $s^p$. To each component $c \in cc(s^p)$ is assigned a Greedy value $\eta(c)$ which serves as a selection criterion: at each step we choose the component $c \in cc(s^p)$ with the maximal Greedy value and extend the partial solution $s^p$ by adding $c$. This process is repeated until set $cc(s^p)$ is empty, or until some other criterion indicates that the solutions construction process should be stopped.

**Algorithm 1** Constructive Heuristic

$s^p = \emptyset$
Generate($cc(s^p)$)
**while** $cc(s^p) \neq \emptyset$ **do**
  c = Selection($Cc(s^p)$)
  $s^p$ = extend $s^p$ by adding the component c
  Generate($cc(s^p)$)
**end while**

#### 1.2.2.3 Local Search Methods

These algorithms start from some initial solution and iteratively try to replace the current solution by a better one, looking in a neighborhood formally defined as follows:

**Definition 2** *A neighborhood is a function $N : S \rightarrow 2^S$ that assigns to each $s \in S$ a set of neighboring solutions $N(s) \subseteq S$. Hereby, each solution $s' \in N(s)$ is obtained by applying an operator to $s$, which applies a rather small*

*change to $s$ and hereby creates a new solution $s'$. Operators or movements are applied in a particular order.*

A neighborhood with an instance of the problem define a search space, which can be represented by a graph where the vertices are solutions that are labeled by the value of the objective function, and the arcs represent the neighborhood relationship of the solutions. A solution $s^* \in S$ is called *globally minimal solution* if for all $s \in S$ it holds that $S : (f(s^*) \leq f(s))$. The introduction of a neighborhood structure enables us to additionally define the concept of *locally minimal solutions*.

**Definition 3** *A local minimum with respect to a neighborhood $N$ is a solution $\widehat{s}$ such that $\forall s \in N(\widehat{s}) : f(\widehat{s}) \leq f(s)$. And $\widehat{s}$ is a strict local minimum if $\forall s : s \in N(\widehat{s}) : (f(\widehat{s}) < f(s))$.*

The simplest method of local search is the *iterative improvement local search* where at each step a neighbor is chosen which is better than the current solution. This method is outlined in algorithm 2.

**Algorithm 2** Iterative Improvement Local Search

```
s = generate_initial_solution()
while ∃s' ∈ N(s) such as f(s') < f(s) do
    s = choose_improving_neighbor(N(s))
end while
```

There are mainly two ways to implement function *choose_improving neighbor*$(N(s))$: searching the neighborhood in a pre-defined order returning the first neighbor which is better than the current solution, or performing an exhaustive search through the neighborhood and returning the best neighbor.

### 1.2.3 Metaheuristics

Metaheuristics (Glover, 1986a; Reeves, 1993) were introduced in the 70's. They are approximate algorithms that combine basic heuristics to explore the search space more effectively and efficiently than constructive heuristics and local search. The term metaheuristic is a Greek compound word. *Heuristic* comes from *heuriskein* which means "search", while the suffix *meta* means " beyond" referring to a higher level. There are different definitions of metaheuristics according to the respective authors. Here is a representative one:

"A metaheuristic is an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a construction method." (Voss et al., 1999)

Here we give an excerpt of some of the fundamental properties taken from different authors, such as for example Blum (2005a). Metaheuristics are characterized by:

- Metaheuristics are strategies that "guide" the search process.
- The goal is to efficiently explore the search space in order to find (near-)optimal solutions.
- Techniques which constitute metaheuristic algorithms range from simple local search procedures to complex learning processes.
- Metaheuristic algorithms are approximate and usually non-deterministic.
- They may incorporate mechanisms to avoid getting trapped in confined areas of the search space.
- The basic concepts of metaheuristics can be described on an abstract level (i.e., not tied to a specific problem)
- Metaheuristics are not problem-specific.
- Metaheuristics may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategy.
- Today more advanced metaheuristics use search experience (embodied in some form of memory) to guide the search.

### 1.2.4 Classification of Metaheuristics

Metaheuristics may be classified in different ways. In the following we outline some of these possible classifications:

**Nature-inspired vs. non-nature inspired.** Probably the most intuitive way of classifying metaheuristics refers to their origins. There are nature-inspired algorithms, such as evolutionary algorithms and ant colony optimization, and non nature-inspired ones such as tabu search and iterated local search. However, this classification may not be very meaningful.

Many recent hybrid algorithms can not be assigned to any of the two classes. Moreover, it is sometimes difficult to clearly attribute an algorithm to one of the two classes.

**Single point vs. population-based search.** Another characteristic that can be used for the classification of metaheuristics is the number of solutions that a metaheuristic works on at the same time: Does the algorithm make use of a population or only of a single solution at any time? Algorithms that work on single solutions are generally referred to as *trajectory methods*. They comprise all metaheuristics that are based on local search. This is because their search process describes a trajectory in the search space. Population-based metaheuristics, on the contrary, either perform search processes which can be described as the evolution of a set of points in the search space, or they perform search processes which can be described as the evolution of a probability distribution over the search space.

**Dynamic vs. static objective function.** Metaheuristics can also be classified regarding the way in which they use the objective function. While some algorithms use static objective functions during run-time, some others, such as guided local search, modify the objective function during the search. The idea behind this approach is to escape from local minima by modifying the search landscape.

**One vs. various neighborhood structures.** Many metaheuristic algorithms only use one single neighborhood structure. In other words, the search landscape topology does not change in the course of the algorithm. Other metaheuristics, such as variable neighborhood search, use more than one neighborhood structure. This opens the possibility to diversify the search by swapping between different search landscapes.

**Memory-based vs. memory-less methods.** A very important feature to classify metaheuristics is based on the fact whether they use memory or not. Memory-less algorithms perform a Markov process, as the information they exclusively use to determine the next action is the current state of the search process. Usually we differentiate between the use of short term and long term memory. The former usually refers to recently performed moves, visited solutions or, in general, decisions taken. The latter is usually an accumulation of synthetic parameters about the search. The use of memory is nowadays recognized as one of the fundamental elements of a powerful metaheuristic.

### 1.2.5 Trajectory Methods

As mentioned before, trajectory methods are characterized by the fact that the work on a single solutions at any time. Their search process describes a trajectory in the search space.

#### 1.2.5.1 Simulated Annealing

The idea of Simulated Annealing (SA) (Metropolis et al., 1953) is taken from the metallurgical industry. It is based on the process of cooling down metal and glass, slow enough in order to get (nearly) perfect crystal structures. In each iteration, the SA selects a solution $s' \in N(s)$ at random, where $s$ is the current solution. If $f(s') < f(s)$ then we replace $s$ by $s'$. Otherwise, $s$ is still replaced by $s'$ with probability $p(s' \mid T_k, s)$ (equation 1.1) which follows a Boltzmann distribution:

$$p(s' \mid T_k, s) = e^{-\frac{f(s')-f(s)}{T_k}} \quad , \tag{1.1}$$

where $T_k$ is a so-called temperature parameter. This version of SA works without memory, but the use of memory for storing the history can be beneficial. The framework of the method is outlined in Algorithm 3.

**Algorithm 3** Simulated annealing (SA)

$s = generate\_initial\_solution()$
$k = 0$
$T_k = set\_initial\_temperature()$
**while** end condition not found **do**
  $s' = choose\_neighbor\_at\_random(N(s))$
  **if** $f(s') < f(s)$ **then**
    $s = s'$
  **else**
    Accept $s'$ as new solution with probability $p(s' \mid T_k, s)$ (see 1.1)
  **end if**
  $update\_temperature(T_k, s)$
**end while**

The different functions that are used in the SA algorithm are explained in more detail in the following:

- *generate_initial_solution*()**:** The algorithm starts by generating a solution that can be random solution or the result of a constructive

heuristic.

- *set_initial_temperature*(): The initial temperature has to be one that allows to move to considerably worse solutions at the beginning of the algorithm. It's setting is crucial for the success of the algorithm.

- *choose_neighbor_at_random*($N(s)$): Randomly select a neighbor.

- *update_temperature*($T_k$): The temperature parameter needs to be updated at each iteration. The idea is to reduce the value of this parameter step by step in order to gradually decrease the probability to move to worse solutions. An example is the following scheme:

$$T_k = T_{k-1} \cdot \alpha, \ \alpha \in (0, 1) \tag{1.2}$$

The temperature planning method is crucial for obtaining good results.

#### 1.2.5.2 Tabu Search

The simple version of tabu search algorithm (TS) (Glover, 1986b) (see Algorithm 4) is based on the (*best-improvement*) version of local search and uses short term memory to avoid local minima and cycling. The short-term memory is implemented by a *tabu list*, denoted as $TL$, which keeps track of the latest solutions visited. $N_a(s)$ is called the *allowed set*. This set is a subset of the neighborhood $N(s)$, generated by eliminating the solutions stored in $TL$. In each iteration we choose the best solution of all the ones in $N_a(s)$ as new current solution. This solution is also stored in $TL$ by function $update(TL, s, s')$. If $TL$ exceeds its maximum capacity, then the oldest solution is removed from $TL$. This means that $TL$ is managed following a FIFO policy. As we can see the algorithm terminates when the end condition is true or when $N_a(s)$ is empty.

The implementation of a short-term memory using a list that stores complete solutions is not practical because keeping a list of solutions is very inefficient. Therefore, rather than storing solutions, we keep the *solution components* involved in the movements. So we need a $TL$ for each type of solution component. Each tabu list $TL$ will be involved in defining a *tabu condition* that serves to filter the neighborhood of the current solution.

**Algorithm 4** Simple tabu search (TS)

$s = generate\_initial\_solution()$
$TL = \emptyset$
**while** end condition not true **do**
  $N_a(s) = N(s) \setminus TL$
  $s' = argmin\{f(s'') \mid s'' \in N_a\}$
  $update(TL, s, s')$
  $s = s'$
**end while**

However, note that storing solution components instead of complete solutions, even though it is much more efficient, it comes with a potential loss of information. This is because when forbidding to visit all solutions that contain this component, solutions may be forbidden that were not visited before. To solve this problem, *aspiration criteria* are used. They may allow to include otherwise forbidden solutions into set $N_a$. The most common aspiration criterion used is when a solution is better than the best solution found. Such a solution should, of course, not be forbidden. Algorithm 5 shows the framework of this more practical version of TS.

**Algorithm 5** Tabu search (TS)

$s = generate\_initial\_solution()$
$init\_tabu\_lists(TL_1, \ldots, TL_r)$
**while** end condition not found **do**
  $N_a(s) = \{s' \in N(s) \mid s'$ is not forbidden,
  or aspiration conditions are satisfied$\}$
  $s' = argmin\{f(s'') \mid s'' \in N_a\}$
  $update\_tabu\_lists(TL_1, \ldots, TL_r, s, s')$
  $s = s'$
**end while**

#### 1.2.5.3 Explorative Local Search Methods

In this section we present some other trajectory methods, which are *greedy randomized adaptive search procedures (GRASP)*, *variable neighborhood search (VNS)*, *guided local search (GLS)* , and *iterate local search (ILS)*.

1. **Greedy Randomized Adaptive Search Procedures** The greedy randomized adaptive search procedure Feo and Resende (1995) is a

metaheuristic that combines a constructive metaheuristic with local search (see Algorithm 6). GRASP is an iterative procedure that consists of two phases: the construction of a solution (see Algorithm 7) and the improvement of the solution built.

**Algorithm 6** Greedy randomized adaptive search procedure (GRASP)

**while** end condition not found **do**
  $s = build\_greedy\_random\_solution()$
  $apply\_local\_search(s)$
**end while**

The construction method is a method that randomly constructs a solution $s^p$ step by step, adding components of a finite list called restricted candidate list ($RCL$). The $RCL$ is made up of the first $\alpha$ components from $cc(s^p)$, assuming that the elements of $cc(s^p)$ are ordered by a Greedy function $\eta$. Note that $\alpha$ is an important parameter.

**Algorithm 7** Greedy randomized solution construction

$s^p = \langle\rangle$
$\alpha = determine\_length\_of\_candidate\_list()$
**while** end condition not found **do**
  $RCL = generate\_candidate\_list(\eta, cc(s^p), \alpha)$
  $c = choose\_at\_random(RCL)$
  $s^p =$ extend $s^p$ by adding solution component $c$
**end while**

As local search it is possible to use any of the available local search algorithms, such as the simple search algorithms, or more advanced methods such as SA or TS. To be effective, GRASP has to satisfy (at least) two conditions:

- The constructive heuristic should explore the best areas of the search space.
- Built solutions should belong to different local minima of the utilized local search.

In order to satisfy these two conditions both algorithm components have to be properly chosen. Moreover, the length of the candidate list must be adequately chosen.

2. **Variable Neighborhood Search**

   Variable neighborhood search (Hansen and Mladenovi, 2001) applies strategies to switch between different neighborhoods of a finite set of predefined neighborhoods (see Algorithm 8):

   VNS is initialized with a set of neighborhoods that are required to meet the following condition: $\forall s : s \in S : (|N_1(s)| < |N_2(s)| < \ldots < |N_{kmax}(s)|)$. Second, an initial solution $s$ is generated. Then the outer loop of the algorithm is iterated until the stopping conditions are reached. Within the outer loop, the neighborhood index $k$ is initialized to 1. Each iteration of the inner loop has three phases: *shaking*, *local search* and *acceptance* of a new current solution. In the shaking phase a solution $s'$ of the $k-$th neighborhood $N_k(s)$ of $s$ is chosen, and is then subject to the local search phase which results in a local minimum $s''$. If $f(s'') < f(s)$ then we replace $s$ by $s''$, and initialize the neighborhood index $k$ to 1. However, if $f(s'') \geq f(s)$ then we increase the neighborhood index $k$ by one in order to diversify the search process.

**Algorithm 8** Variable neighborhood search (VNS)

```
define neighborhoods N_k, k = 1, ..., k_max
s = generate_initial_solution()
while end condition not reached do
  k = 1
  while k < k_max do
    s' = choose_at_random(N_k(s))
    s'' = local_search(s')
    if f(s'') < f(s) then
      s = s''
      k = 1
    else
      k = k + 1
    end if
  end while
end while
```

3. **Guided local search**

   Guided local search (GLS) (Voudouris and Tsang, 1999) is a meta-

heuristic that uses a dynamic objective function to escape from local minima. The dynamic objective function $f'$ is obtained from an adaptive change of the original objective function $f$. This change is based on a set of, in general, $m$ characteristics of a solution: $sf_i, i = 1, \ldots, m$. These features can be used to differentiate solutions. $I(i, s)$ tells us if the property $sf_i$ is present in the solution $s$.

$$I(i,s) = \begin{cases} 1: \text{ if the feature } sf_i \text{ is in the solution } s \\ 0: \text{ otherwise} \end{cases} \tag{1.3}$$

During the execution of the algorithm, the original function $f(\cdot)$ is replaced by $f'(\cdot)$, which is obtained from $f(\cdot)$ by adding the penalty $p_i : i = 1, \ldots, m$, where $\lambda > 0$ is the influence of $p_i$ in $f'(\cdot)$:

$$f'(s) = f(s) + \lambda \sum_{i=1}^{m} p_i \cdot I(i,s) \tag{1.4}$$

The algorithm (see Algorithm 9) works as follows. First, an initial solution $s$ is generated and the vector of penalties $\mathbf{p}$ is initialized to all zeros. Then, at each iteration local search is applied to the current solution $s$ based on the changed objective function $f'$. This results in a solution $s'$. Depending on $s'$ the penalty vector $p = (p_1, \ldots, p_m)$ is modified in the function $update\_vector\_penalty(p, s')$ calculating the *utility* $U(i, s')$ for each property:

$$U(i,s') = I(i,s') \cdot \frac{c_i}{1 + p_i} \tag{1.5}$$

The elements $p_i$ of the penalty vector $\mathbf{p}$ may be modified as follows:

$$p_i = p_i + 1 \tag{1.6}$$

However, there are also other ways for the penalty vector update.

4. **Iterated local search**

In each iteration of iterated local search (ILS) (Stützle, 1999) (see Algorithm 10) the current solution $s'$, which is a local minimum, is

**Algorithm 9** Guided local search (GLS)

$s = generate\_initial\_solution()$
$\mathbf{p} = (0, \ldots, 0)$
**while** end condition not found **do**
  $s' = local\_search(s, f')$
  $update\_penalty\_vector(\mathbf{p}, s')$
  $s = s'$
**end while**

subject to the perturbation function, which returns a perturbed solution $s''$. Afterwards, solution $s''$ is subject to local search which provides a new local minimum $s'''$. Finally, the algorithm must decide between solutions $s'$ and $s'''$ for a new current solution. The *perturbation* prevents the algorithm from being trapped in local minima, whereas the *acceptance criteria* has an influence on the diversification and intensification behaviour of the algorithm.

**Algorithm 10** Iterated local search (ILS)

$s = generate\_initial\_solution()$
$s' = local\_search(s)$
**while** end condition not found **do**
  $s'' = perturbation(s', memory)$
  $s''' = local\_search(s'')$
  $s' = apply\_acceptance\_criteria(s''', s', memory)$
**end while**

- The term *memory* (see Algorithm 10) refers, for example, to the fact that solutions found during the search process may be stored and used for different purposes.
- $generate\_initial\_solution()$: This function constructs the initial solution. The most important requirement is to be fast, such as, for example, the generation of a random solution or the use of a simple greedy heuristic.

- $perturbation(s', memory)$: The perturbation usually tends to be non-deterministic. The most important feature of the perturbation is the *strength*, defined as the "damage" inflicted on the

current solution. This feature can be fixed or variable. In the first case the distance between $s'$ and $s''$ will always be the same regardless of the state of the search process. However, a *variable* strength is usually more effective and must be experimentally found, depending on the current state of the search process.

- $apply\_acceptance\_criteria(s''', s', memory)$: The two extreme cases for the acceptance criterion are as follows. We may only accept $s'''$ as new current solution in case it is better than $s'$, or we may always accept the new local minimum regardless of its quality. In between these two cases we have several possibilities to adopt, for example, acceptance criteria similar to the one used in SA.

### 1.2.6 Population-Based Methods

The most well known metaheuristics based on populations for the application to combinatorial optimization problems are evolutionary algorithms (EAs), or (*evolutionary computation*(EC) algorithms), and *ant colony optimization (ACO)* algorithms.

#### 1.2.6.1 Evolutionary Algorithms

Evolutionary algorithms (EC) are inspired by nature's ability to adapt to the environment, that is, the natural evolution of species. In each iteration the individuals that make up the current population are subject to operations such as recombination, from which arise the individuals of the next generation (iteration). These next generation individuals are selected based on their fitness, which is determined by their objective function value.

The family of evolutionary algorithms can be divided into three categories of independent development: Firstly, evolutionary programming (EP) was introduced by Fogel (Fogel et al., 1966). Then, evolutionary strategies (ES) were proposed by Rechenberg (Rechenberg and Eigen, 1973) and finally genetic algorithms (GA) as proposed by Holland (Holland, 1992).

Algorithm 11 shows the basic structure of EC algorithms. In the algorithm, $P$ denotes the population. New individuals are produced by applying recombination and mutation operators to the individuals of $P$. Then the new population $P''$ is selected from $P$ and from the newly generated individuals. The main features of the EC algorithm are as follows:

- **Representation of the individuals**: Commonly used solution representations are bit-strings or permutations of integers. Note that a

**Algorithm 11** Evolutionary Computation (EC)

```
P = generate_initial_population()
evaluate(P)
while end condition not found do
  P' = recombination(P)
  P'' = mutation(P')
  evaluate(P'')
  P = choose(P'', P)
end while
```

solution representation is generally called *genotype* whereas the solution which is represented is called *phenotype*.

- **Process of evolution**: In each iteration, individuals are chosen to constitute the next generation. In some cases the new population is exclusively composed of new individuals. In other cases, the new population is chosen as the best set of solutions from the old population and the newly generated individuals. Many times the number of individuals is constant from iteration to iteration.

- **Neighborhood function**: In the context of EC algorithms, this concept refers to a function $N_{\varepsilon c} : I \longrightarrow 2^R$ that assigns to each individual $i \in I$ a set of individuals $N_{\varepsilon c}(i) \subseteq I$, which refers to the set of solutions with which individual $i$ can be recombined.

- **Information source**: This refers to the input parameters of a recombination operator. In the standard case, a pair of parent individuals are recombined to generate one or two new individuals (children). But it could also be that more than two individuals are recombined to create new individuals.

- **Unfeasible solutions**: An important aspect of EC algorithms is the way of dealing with individuals who are not feasible. This problem is often encountered, because many genetic operators may generate unfeasible individuals. There are three ways to address this: the first is to discard the infeasible individuals, the second way is to penalize infeasible individuals based on decreasing their quality, and the third way is to try to fix the corresponding individual.

#### 1.2.6.2 Ant Colony Optimization

In order to solve a given CO problem, the metaheuristic ant colony optimization (ACO) has first to derive a finite set $C$ of solution components, which is used to assemble solutions (Dorigo and Blum, 2005; Blum and Roli, 2003). Then we have to define a set $T$ of pheromone values, which is generally known as the *pheromone model*. Together with a mechanism for constructing solutions, these values define a probability distribution over the search space. The pheromone model is the heart of each ACO metaheuristic. Generally, each solution component from $C$ has an associated pheromone value $\tau_i \in T$, so that solutions can be generated by assembling components probabilistically.

The ACO in each iteration has two phases:

- Solutions are built on the basis of the pheromone model.
- The values of the pheromone model are updated depending on the quality of the solutions built.

The general idea is that the pheromone model can guide the search process to those parts of the search space containing high quality solutions. This is because the pheromone value update increases the value of the pheromone components depending on the quality of the corresponding solutions. A precondition for ACO to work is that good solutions contain good solution components.

**Algorithm 12** Ant Colony Optimization (ACO)

```
while end condition not found do
  build_solution()
  update_pheromones()
  daemon_actions()
end while
```

In the following we give a more detailed description of ACO (see Algorithm 12). ACO is an iterative algorithm which consists of three stages or procedures *build_solution*(), *update_pheromones*(), and *daemon_actions*(). These procedures are explained in more detail in the following:

*build_solution*(): (see Algorithm 13) The exploration of the ants is simulated with a probabilistic constructive heuristic that assembles solution

components from a finite set $C = \{c_1, \ldots, c_n\}$. Each component $c_i$ of this set $C$ has an associated pheromone value $\tau_i$. A complete solution is obtained as a sequence of solution components $s$. A solution construction start with the empty sequence $s = \langle\rangle$. At each step, the current sequence $s$ is extended by adding a solution component from $N(s) \subseteq C \setminus s$. The selection of component $c_i \in N(s)$ is done in a probabilistic way: to each component we assign a probability $p(c_i \mid s)$ which depends on greedy information ($\eta$) and pheromone information:

$$p(c_i \mid s) = \frac{[\tau_i]^\alpha \cdot [\eta(c_i)]^\beta}{\sum_{c_j \in N(s)} [\eta_j]^\alpha \cdot [\eta(c_j)]^\beta}, \forall c_i \in N(s), \tag{1.7}$$

Hereby, $\eta$ is a greedy function which is also known as the *heuristic information*. This greedy function assigns to each component $c_i \subseteq N(s)$ a value $\eta(s)$. Moreover, the positive exponents $\alpha$ and $\beta$ are parameters that scale the weight of the heuristic information in relation to the weight of the pheromone information.

**Algorithm 13** *build_solution*()

```
s = ⟨⟩
compute(cc(s))
while Cc(s) ≠ ∅ do
  c = choose(cc(s))
  s = add component c to s
  compute(cc(s))
end while
```

*update_pheromones*(): The pheromone updating process has two phases: the first phase consists of the so-called pheromone evaporation, which decreases the value of the pheromones uniformly. This function is necessary in order to avoid the rapid convergence of the algorithm and explore different regions of the solution space. Second, pheromones are increased in each iteration as follows:

$$\tau_i = (1 - \rho) \cdot \tau_i + \rho \cdot \sum_{\{s \in S_{upd} | c_i \in s\}} (w_s \cdot F(s)) \ , \tag{1.8}$$

for $i = 1, \ldots, n$. Hereby, $S_{upd}$ the set of solutions that are used to update the pheromones, and $F(s)$ is a function $F : S \longmapsto \mathbb{R}^+$ such that

$f(s) < f(s') \Longrightarrow F(s) \geqslant F(s'), \forall s, s' \in S$, and $w_s \in \mathbb{R}^+$ denotes the weight of the solution $s$. In most cases, $S_{upd}$ is composed of the best solutions constructed in the current iteration.

*daemon_action*(): Here we apply those actions that require a global vision.

### 1.2.7 Hybrid Metaheuristics

The concept of hybrid metaheuristics is only known from recent years, even if the idea of combining different metaheuristic strategies and algorithms dates back to the 1980s. Today, we can observe a generalized common agreement on the advantage of combining components from different search techniques and the tendency of designing hybrid techniques is widespread in the fields of operations research and artificial intelligence. The consolidated interest around hybrid metaheuristics is also demonstrated by publications on classifications, taxonomies and overviews on the subject (Raidl, 2006; Talbi, 2002).

In general, a hybrid metaheuristic is obtained by combining a metaheuristic with algorithmic components originating from other techniques for optimization (possibly another metaheuristic). We may distinguish between two categories: the first consists in designing a solver including components from a metaheuristic into another one, while the second combines metaheuristics with other techniques typical of fields such as operations research and artificial intelligence. A prominent representative of the first category is the introduction of trajectory methods into population based techniques or the use of a specific local search method into a more general trajectory method such as ILS. The second category includes hybrids resulting from the combination of metaheuristics with constraint programming (CP), integer programming (IP), tree-based search methods, data mining techniques, etc. Both categories contain numerous instances and an exhaustive description is not possible.

#### 1.2.7.1 Component Exchange Among Metaheuristics

One of the most popular ways of metaheuristic hybridization is the use of trajectory methods inside population-based methods. Indeed, most of the successful applications of EC and ACO make use of local search procedures. The reason for that becomes apparent when analyzing the respective strengths of trajectory methods and population-based methods.

The power of population-based methods is certainly their capability of recombining solutions to obtain new ones. In EC algorithms explicit recombinations are implemented by one or more recombination operators. In ACO, for example, recombination is implicit, because new solutions are generated by using a probability distribution over the search space which is a function of earlier populations. This enables the search process to perform a guided sampling of the search space, usually resulting in a coarse grained exploration. Therefore, these techniques can effectively find promising areas of the search space.

The strength of trajectory methods is rather their way in which they explore a promising region of the search space. As in those methods local search is the driving component, a promising area in the search space is searched in a more structured way than in population-based methods. Therefore, the danger of being close to good solutions but "missing" them is not as high as in population-based methods. More formally, local search techniques efficiently drive the search toward the attractors, i.e., local optima or confined areas of the space in which many local optima are condensed.

In summary, population-based methods are better in identifying promising areas in the search space from which trajectory methods can quickly reach good local minima. Therefore, hybrid metaheuristics that can effectively combine the strengths of both population-based methods and trajectory methods are often very successful.

#### 1.2.7.2 Integration of Metaheuristics With AI and OR Techniques

One of the most prominent research directions is the integration of metaheuristics with more classical artificial intelligence (AI) and operations research (OR) methods, such as constraint programming (CP) and branch & bound or other tree search techniques. In the following we outline some of the possible ways of integration.

Metaheuristics and tree search methods can be sequentially applied or they can also be interleaved. For instance, a tree search method can be applied to generate a partial solution which will then be completed by a metaheuristic approach. Alternatively, metaheuristics can be applied to improve a solution generated by a tree-search method.

CP techniques can be used to reduce the search space or the neighborhood to be explored by a local search method. In CP, combinatorial optimization problems are modelled by means of variables, domains and constraints, which can be mathematical (as for example in linear program-

ming) or symbolic. Constraints encapsulate well-defined parts of the problem into sub-problems, thus making it possible to design specialized solving algorithms for sub-problems that occur frequently. Every constraint is associated to a *filtering* algorithm that deletes values from a variable domain that do not contribute to feasible solutions. Metaheuristics (especially trajectory methods) may use CP to efficiently explore the neighborhood of the current solution, instead of simply enumerating the neighbors or randomly sampling the neighborhood. A prominent example of such a kind of integration is Large Neighborhood Search (Shaw, 1998a), which is the technique that we developed in this thesis in the context of integer programming. These approaches are effective mainly when the neighborhood to explore is very large, or when problems (such as many real-world problems) have additional constraints (usually called *side constraints*). A detailed overview of the possible ways of integration of CP and metaheuristics can be found in (Focacci et al., 2002).

Another possible combination consists in introducing concepts or strategies from either class of algorithms into the other. For example, the concepts of tabu list and aspiration criteria—known from tabu search—can be used to manage the list of open nodes (i.e., the ones whose child nodes are not yet explored) in a tree search algorithm. An example of such an approach can be found in (Della Croce and T'kindt, 2002). Tree-based search is also successfully integrated into ACO in (Blum, 2005b), where beam search (Ow and Morton, 1988) is used for solution construction.

Integer and linear programming can be also effectively combined with metaheuristics. For instance, linear programming is often used either to solve a sub-problem or to provide dual information to a metaheuristic in order to select the most promising candidate solution or solution component (Ibaraki and Nakamura, 2006; Blum, 2005b).

The kinds of integration we shortly mentioned belong to the class of *integrative combinations*. The other possible way of integration, called either *collaborative combinations* or also *cooperative search* consists in a loose form of hybridization, in that search is performed by possibly different algorithms that exchange information about states, models, entire sub-problems, solutions or search space characteristics. Typically, cooperative search algorithms consist of the parallel execution of search algorithms with a varying level of communication. The algorithms can be different or they can be instances of the same algorithm working on different models or running with different parameter settings. The algorithms composing a cooperative search

system can be all approximate, all complete, or a mix of approximate and complete approaches. This area of research shares many issues with the design of parallel algorithms and we forward the interested reader to the specific literature on the subject (Alba, 2005).

### 1.2.8 Intensification and Diversification

As mentioned before, (hybrid) metaheuristics are intelligent strategies for exploring a search space. Crucial for the success of such an algorithm is a well-adjusted (dynamic) balance between *diversification* and *intensification*. The term diversification generally refers to the exploration of the search space, while the term intensification refers to the exploitation of the accumulated search experience. The balance between diversification and intensification is important because, first, we would like to quickly identify areas of search space with high quality solutions, and second, we would like to avoid spending too much time in areas of the search space that are already well explored or that only consist of poor-quality solutions. For more to see (Blum, 2004).

### 1.2.9 Objectives of the master thesis:

The main goal of this master thesis is to propose new algorithms for SCM, with special emphasis on problems encountered in energy applications. Particularly, we will focus on two engineering problems that have attracted an increasing interest in the recent past: the design of hydrogen supply chains for vehicle use and the strategic planning of ethanol supply chains.

We will first formally state the problems of interest, and then present detailed mathematical formulations based on MILP that will be expedited through the use of tailored algorithms. It should be mentioned that our methods could be easily extended to other SCM applications, since the models addressed in this work show general common features with standard supply chain formulations.

# Chapter 2

# Hydrogen

This first example addresses the design of supply chains for Hydrogen production (Sabio et al., 2010). In this case, uncertainties are not included in the model for the sake of simplicity.

## 2.1 Problem Statement

The first SCM problem addressed in this work has as objective to determine the configuration of a three-echelon hydrogen network for vehicle use (production-storage-market) with the goal of minimizing the expected total discounted cost and financial risk. The structure of the three-echelon SC taken as reference in this work is depicted in Fig. 2.1. This network includes a set of plants, where hydrogen can be produced (hexagons), and a set of storage facilities (circles), where hydrogen is stored before being delivered to the final customers (rectangles). We consider a given region (e.g., a country, a continent, etc.) that can be divided into a set of potential locations that correspond to different sub-regions of the original region of interest characterized by a given hydrogen demand. The set of potential locations of the problem along with the associated geographical distribution of the demand are input data to the problem.

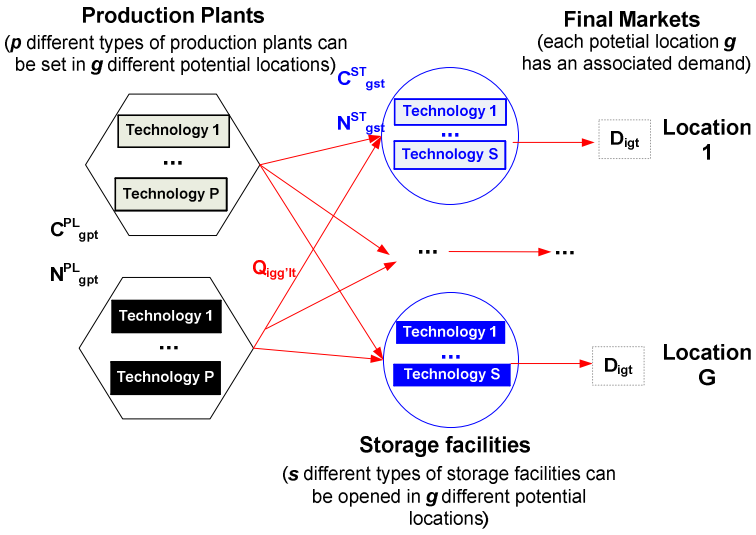


Figure 2.1: The structure of the three-echelon SC (HYDROGEN)

The network design problem can therefore be formally stated as follows. Given are a fixed time horizon and number of time periods, the set of available production, storage and transportation technologies, the capacity limitations of plants and storage facilities, the costs associated with the network operation (production, transportation and inventory costs), the investment cost, the probabilistic information that describe the uncertain parameters (i.e., type of probability distribution, mean and variance) and interest rate. The final goal is to determine (1) the SC design, including the number, type, location and capacity of plants and storage facilities; (2) the number and type of transportation units (e.g. tanker trucks, railway tube cars, etc.) and transportation links to be established between the potential locations; and (3) the associated planning decisions, including the production rates at the plants, inventory levels at the storage facilities and flows of hydrogen between plants and storage facilities; in order to minimize the total cost.

## 2.2 Mathematical Model

Our mathematical formulation, which is based on the superstructure depicted in Fig. 1, is similar to that introduced by Guillén-Gosálbez et al. (2010). Particularly, the model considers the possibility of establishing dif-

ferent production and storage facilities in a set of potential locations with known demand and uncertain economic parameters. For the sake of completeness of our work, we next provide a detailed description of the MILP model notation, constraints, and objective function equations.

### 2.2.1 Mass balance constraints

The mass balance must be satisfied in each potential location $g$ and time period $t$. Thus, for every hydrogen form $i$,liquid or gas, the initial inventory kept in a location $S_{igst-1}$ plus the amount produced ($PR_{igpt}$) and the input flow rate ($Q_{ilg'gt}$) must equal the final inventory ($S_{igst}$)plus the amount delivered to the customers ($D_{igt}$)and the output flow rate ($Q_{ilgg't}$):

$$\begin{aligned}&\sum_{s\in SI(i)} S_{igst-1} + \sum_{p} PR_{igpt} + \sum_{g'\neq g}\sum_{l} Q_{ilg'glt} \\ &= \sum_{s\in SI(i)} S_{igst} + D_{igt} + \sum_{g'\neq g}\sum_{l} Q_{ilgg'lt} \forall i,g,t\end{aligned} \tag{2.1}$$

In this equation, $SI(i)$ represents the set of technologies that can be used to store product form $i$. Furthermore, the total amount of hydrogen consumed ($D_{igt}$) is restricted to be lower than the hydrogen demand ($\overline{D_{gt}}$) and higher than a minimum demand satisfaction level ($dsat$):

$$\overline{D_{gt}} dsat \leq \sum_{i} D_{igt} \leq \overline{D_{gt}} \forall g,t \tag{2.2}$$

### 2.2.2 Capacity constraints

#### 2.2.2.1 Plants

The capacity of each production technology $p$ of product form $i$ at location $g$ in period $t$is represented by a continuous variable denoted by $C^{PL}_{gpt}$. Eq. (2.3) constraints the total production rate ($PR_{igp}$) to be lower than the existing capacity and higher than a minimum desired percentage, $\tau$, of the capacity installed.

$$\tau C^{PL}_{gpt} \leq \sum_{i} PR_{igpt} \leq C^{PL}_{gpt} \forall g,p,t \tag{2.3}$$

The capacity of each technology $p$ in any time period $t$ is calculated from the existing capacity at the end of the previous period plus the expansion in capacity ($CE^{PL}_{gpt}$) executed in period $t$ and location $g$:

$$C^{PL}_{gpt} = C^{PL}_{gpt-1} + CE^{PL}_{gpt} \quad \forall g,p,t \tag{2.4}$$

Eq. (2.4) is applied to limit capacity expansions within lower and upper bounds. These limits are calculated from the number of plants installed $N_{gpt}^{PL}$ and the minimum and maximum capacities associated with each technology $p$ ($\underline{PC_p^{PL}}$ and $\overline{PC_p^{PL}}$, respectively).

$$\underline{PC_p^{PL}} N_{gpt}^{PL} \leq CE_{gpt}^{PL} \leq \overline{PC_p^{PL}} N_{gpt}^{PL} \forall g, p, t \tag{2.5}$$

#### 2.2.2.2 Storage facilities

The storage capacity of product form $i$ during period $t$ in location $g$ associated with storage technology $s$ is represented by the continuous variable $C_{gst}^{ST}$. The total inventory of product in form $i$ kept at the end of period $t$ in the storage facilities of type $s$ installed in location $g$ ($S_{igst}$), is enforced to be lower than the available capacity by means of Eq. (2.6).

$$\sum_{i \in IS(s)} S_{igst} \leq C_{gst}^{ST} \forall g, s, t \tag{2.6}$$

Here, $IS(s)$ denotes the set of product forms $i$ that can be stored by technology $s$. Moreover, the amount of hydrogen delivered from the storage facility to the customers is constrained by its capacity. In steady-state operation, the average inventory of a product form $i$ in location $g$, is determined from the amount delivered to customers ($D_{igt}$) and the storage period $\theta$. This storage period is introduced to cover fluctuations in both supply and demand as well as plant interruptions (Almansoori and Shah, 2006):

$$2(\theta D_{igt}) \leq \sum_{s \in SI(i)} C_{gst}^{ST} \forall i, g, t \tag{2.7}$$

In Eq. (2.7) $SI(s)$ denotes the set of storage technologies $s$ that can handle product forms $i$. Finally, the capacity of the storage technology at any time period is determined from the previous one and the expansion in capacity executed in the same period:

$$C_{gst}^{ST} = C_{gst-1}^{ST} + CE_{gst}^{ST} \forall g, s, t \tag{2.8}$$

In a similar manner as occurred with the manufacturing plants, the value of $CE_{gst}^{ST}$ is bounded within lower and upper limits.

$$\underline{SC_s^{ST}} N_{gst}^{ST} \leq CE_{gst}^{ST} \leq \overline{SC_s^{ST}} N_{gst}^{ST} \forall g, s, t \tag{2.9}$$

#### 2.2.2.3 Transportation constraints

In this block of equations, we introduce a binary variable $X_{gg'lt}$, which takes a value of one if a transportation link of type $l$ (i.e., tanker trucks, railway tube cars, etc.) is established between locations $g$ and $g'$ in time period $t$, and zero otherwise. Eq. (2.10) enforces the definition of this variable.

$$\begin{aligned} &\underline{QC_{lgg'}} X_{gg'lt} \leq \sum_i Q_{ilgg't} \leq \overline{QC_{lgg'}} X_{gg'lt} \\ &\forall g, g'(g \neq g'), l \in LI(i) \cup NPL, t \end{aligned} \tag{2.10}$$

Note that a zero value of the aforementioned binary variable prevents the flow of materials that can be transported via transportation technology $l$, from taking place, whereas a value of one allows the transport flows within some lower $\underline{QC_{lgg'}}$and upper limits $\overline{QC_{lgg'}}$. In this equation, $LI(i)$ denotes the set of technologies $l$ that can transport $i$, whereas $NPL$ is the set of transportation technologies that involve the use of either railway, trucks or ships.

Eq. (2.11) is similar to Eq. (2.10), but applies only to pipelines. Specifically, we assume that if a pipeline is constructed, then the associated transportation link will remain opened during the entire time horizon:

$$\begin{aligned} &\sum_{t' \leq t+1} \underline{QC_{lgg'}} X_{gg'lt'} \leq \sum_i Q_{ilgg't} \leq \sum_{t' \leq t+1} \overline{QC_{lgg'}} X_{gg'lt} \\ &\forall g, g'(g \neq g'), l = pipeline, t \end{aligned} \tag{2.11}$$

Furthermore, only one transportation link involving pipelines can be constructed at most during the entire time horizon:

$$\sum_{t' \leq t+1} X_{gg'lt'} \leq 1 \qquad \forall g, g'(g \neq g'), l = pipeline, t \tag{2.12}$$

We assume that a location can either import or export hydrogen, but not both at the same time. This is because if a location can only satisfy its needs by importing from other locations, it would not make sense to export to other locations:

$$X_{gg'lt} + X_{g'glt} \leq 1 \qquad \forall g, g'(g \neq g'), l \in LI(i, t \tag{2.13}$$

Some specific constraints are appended to the model formulation to handle the specific case of maritime transportation devices. Hence, some binary variables $X_{lgg't}$ denoting the existence of transportation links must be forced

to take a zero value in some particular cases to prevent ships from transporting materials between grids without harbors (Eq. 2.14) and also from avoiding the use of road transportation devices, excluding pipelines, between those grids that can only be connected via a maritime link (Eq. 2.15)

$$\begin{aligned} &X_{lgg't} = 0 \qquad \forall l, g, g' \in LG' \\ &LG' = \{l, g, g' : (l = ship) \wedge ((g, g') \notin SGG(gg'))\} \end{aligned} \tag{2.14}$$

$$\begin{aligned} &X_{lgg't} = 0 \qquad \forall l, g, g' \in LG \\ &LG = \{l, g, g' : (l \neq ship) \wedge ((g, g') \in SGG'(gg'))\} \end{aligned} \tag{2.15}$$

In these constraints, $SGG(g, g')$ is the subset of allowable maritime links, whereas $SGG(g, g')$ is the subset of maritime links (i.e., $SGG'(g, g') \subset SGG(g, g')$) that cannot be connected through road transportation units.

Finally, Eq. (2.16) is introduced to avoid transportation tasks within the same locations:

$$X_{lgg't} = 0 \qquad \forall l, g = g' \tag{2.16}$$

### 2.2.3 Objective function equations

The model considers that the coefficients of the objective function (e.g. facility investment and variable costs and transportation capital costs) are uncertain and that their variability can be described through a set of scenarios with given probability of occurrence. As a result, the cost associated with the establishment and operation of the SC is not a single nominal value, instead it is a stochastic variable that follows a discrete probability function. In this context, the optimization method must identify the set of solutions (i.e., strategic SC decisions) that simultaneously minimize the expected value of the cost distribution as well as its risk level. The main advantage of the scenario-based approach is that it allows to deal with any type of probability function. Furthermore, this approach avoids the nonlinearities associated with the reformulation of the probabilistic constraints used in robust optimization.

#### 2.2.3.1 Expected cost

The expected total cost is given by the mean value of the cost discrete distribution described by the scenario realizations:

$$E[TDC] = \sum_{e} prob_e TDC_e \tag{2.17}$$

The total discounted cost attained in each particular scenario realization($TDC_e$) is calculated as the summation of the discounted costs associated with each time period:

$$TDC_e = \sum_t \frac{TC_{te}}{(1+ir)^{t-1}} \qquad \forall e \tag{2.18}$$

In the aforementioned expressions, $e$ is a subscript that represents a particular scenario $e$ and $prob_e$ is the probability of occurrence associated to each scenario. In Eq. (2.18), $ir$ represents the interest rate and $TC_{te}$ is the total amount of money spent in period $t$ and scenario $e$, which includes the capital ($FCC_t$, $TCC_t$) as well as operating costs ($FOC_{te}$, $TOC_{te}$) given by the production, storage and transportation facilities of the network:

$$TC_{te} = FCC_t + TCC_t + FOC_{te} + TOC_{te} \qquad \forall t, e \tag{2.19}$$

In general, it will be possible to know accurately the capital cost at the design stage, since it is usually agreed before the establishment of a new facility. On the other hand, the value of the operating cost will fluctuate according to the market trends. Hence, in Eq. (2.18) it seems convenient to assume that $FCC$ and $TCC$ are non scenario dependent, whereas $FOC$ and $TOC$ will depend on the specific scenario realization.

The facility operating cost term is obtained with multiplying the unit production and storage costs ($upc_{igpte}$ and $usc_{igste}$, respectively), which are regarded as uncertain parameters, with the corresponding production rates and average inventory levels:

$$\begin{aligned} FOC_{te} = & \sum_i \sum_g \sum_p upc_{igpte} PR_{igpt} \\ & + \sum_i \sum_g \sum_s \in SI(i) usc_{igste} \left(\theta D_{igt}\right) \qquad \forall t, e \end{aligned} \tag{2.20}$$

#### 2.2.3.2 Facility capital cost

The facility capital cost in period $t$ ($FCC_t$) is determined from the capacity expansions made in the manufacturing plants and storage facilities during that period:

$$\begin{aligned} FCC_t = & \sum_g \sum_p \left(\alpha^{PL}_{gpt} N^{PL}_{gpt} + \beta^{PL}_{gpt} CE^{PL}_{gpt}\right) \\ & + \sum_g \sum_s \left(\alpha^{ST}_{gst} N^{ST}_{gst} + \beta^{ST}_{gst} CE^{ST}_{gst}\right) \qquad \forall t \end{aligned} \tag{2.21}$$

The parameters,$\alpha_{gpt}^{PL}$, $\beta_{gpt}^{PL}$, $\alpha_{gst}^{ST}$ and $\beta_{gst}^{ST}$ are the fixed and variable investment terms corresponding to plants and storage facilities, respectively. These parameters reflect the concept of economies of scale.

#### 2.2.3.3 Transportation capital cost

The transportation capital cost, which includes the cost of the trucks and railcars required to satisfy the demand, is calculated via Eq. (2.22):

$$TCC_t = \sum_{l \neq ship,pipeline} N_{lt}^{TR} \cdot cc_{lt} + PCC_t \tag{2.22}$$

Here, $PCC_t$ is the pipeline capital costs, $cc_{lt}$ represents the capital cost associated with transport mode $l$ in period $t$, and $N_{lt}^{TR}$ is an integer variable that denotes the total number of transportation units of type $l$ purchased in period $t$ that can transport product $i$ (i.e., $l \in LI(i)$). Note that ships and pipelines are excluded from the first summation term of the equation. This is because the model assumes that ships are hired for carrying out the specific transportation tasks (i.e., outsourcing), whereas the capital cost of pipelines is calculated via the following equation:

$$PCC(t) = \sum_{g} \sum_{g' \neq g} \sum_{l \in LI(i)} upcc_t X_{lgg't} distance_{gg'} \qquad \forall t \tag{2.23}$$

where $upcc_t$ is the unit capital cost of the pipeline per unit of length built, and $distance_{gg'}$ denotes the distance between grids $g$ and $g'$.

The average number of trucks and/or railcars required to satisfy a certain flow between different locations is computed from the flow rate of products between the locations ($Q_{igg'lt}$), the transportation mode availability ($av_l$), the capacity of a transport container ($tcap_l$), the average distance traveled between the locations ($distance_{gg'}$), the average speed ($speed_l$) and the loading/unloading time ($lutime_l$), as stated in Eq. (2.24):

$$\sum_{t' \leq t+1} N_{lt'}^{TR} \geq \sum_{i \in IL(l)} \sum_{g} \sum_{g' \neq g} \sum_{t} \frac{Q_{igg'lt}}{av_l tcap_l} \left( \frac{2distance_{gg'}}{speed_l} + lutime_l \right) \tag{2.24}$$
$$\forall l \neq ship, pipeline$$

The total number of transportation units available in any period $t$ includes the ones purchased in the same period $t$ as well as those acquired in previous periods $t'$. Therefore, the left hand side of the inequality in Eq. 2.24 represents the summation of all the transportation units purchased in all

the time periods $t'$ up to the actual period $t$ (i.e., $t' = t$). In this equation, $IL(l)$ denotes the set of product forms $i$ that can be transported by transport mode $l$. For the sake of simplicity, this work assumes that each transportation facility can only operate between two predefined locations. Thus, in constraint 23, the distance between locations $g$ and $g'$ ($distance_{gg'}$) is multiplied by two to account for the return journey of the trucks/railcars.

#### 2.2.3.4 Transportation operating cost

The total operating cost associated with the transportation tasks carried out in scenario $e$ in period $t$ ($TOC_{te}$) is determined from Eq. 2.25:

$$TOC_{te} = ROC_{te} + POC_{te} + SOC_{te} \qquad \forall t, e \tag{2.25}$$

where $ROC_{te}$, $POC_{te}$ and $SOC_{te}$ are the operating costs associated with road transportation technologies and railway, pipelines and ships, respectively. The first term includes the fuel ($FC_{te}$), labor ($LC_{te}$), maintenance ($MC_{te}$) and general costs ($GC_{te}$):

$$ROC_{te} = FC_{te} + LC_{te} + MC_{te} + GC_{te} \qquad \forall t, e \tag{2.26}$$

The fuel cost is a function of the fuel price ($fuelp_{lte}$) and fuel usage:

$$FC_{te} = \sum_{i} \sum_{g} \sum_{g' \neq g} \sum_{l \in LI(i)} fuelp_{lte} \frac{2 distance_{gg'} Q_{ilgg't}}{fuelc_l tcap_l} \qquad \forall t, e \tag{2.27}$$

Note that the main source of uncertainty here is the fuel price, since it cannot be perfectly known in advance at the design stage. In Eq. (2.27), the fractional term represents the fuel usage, and it is determined from the total distance traveled in a trip (2 $distance_{gg'}$), the fuel consumption of transport mode $l$ ($fuelc_l$) and the number of trips made per period of time ($\frac{Q_{ilgg't}}{tcap_l}$). The labor transportation cost is described as a function of the driver wage in scenario $e$ ($wage_{le}$) and total delivery time (term inside the brackets):

$$\begin{aligned} LC_{te} = & \sum_{i} \sum_{g} \sum_{g' \neq g} \sum_{l \in LI(i)} wage_{lte} \\ & \times \left[ \frac{Q_{ilgg't}}{tcap_l} \left( \frac{2 distance_{gg'}}{speed_l} + lutime_l \right) \right] \qquad \forall t, e \end{aligned} \tag{2.28}$$

The maintenance cost, which accounts for the general maintenance of the transportation systems, is a function of the cost per unit of distance traveled

in scenario $e$ ($cud_{le}$) and total distance driven:

$$MC_{te} = \sum_{i} \sum_{g} \sum_{g' \neq g} \sum_{l \in LI(i)} cud_{le} \frac{2distance_{gg'} Q_{ilgg't}}{tcap_l} \qquad \forall t, e \qquad (2.29)$$

Finally, the general cost includes the transportation insurance, license and registration, and outstanding finances. It can be determined from the unit general expenses in scenario $e$ ($ge_{lte}$) and number of transportation units as follows:

$$GC_{te} = \sum_{l} \sum_{t' \leq t} ge_{lte} N^{TR}_{lt'} \qquad \forall t, e \qquad (2.30)$$

Eq. (2.31) determines the pipeline operating costs from the unit operating cost of the pipelines in scenario $e$ ($upoc_{te}$) and the freight to be delivered.

$$POC(t, e) = \sum_{i} \sum_{g} \sum_{g' \neq g} \sum_{l \in LI(i)} upoc_{te} Q_{ilgg't} \qquad \forall t, e \qquad (2.31)$$

Finally, Eq. (2.32) calculates the ship operating costs from the unit operating costs for maritime transportation in scenario $e$ ($usoc_{te}$), the time required to deliver the hydrogen and the load transported:

$$SOC_{t,e} = \sum_{i} \sum_{g} \sum_{g' \neq g} \sum_{l \in LI(i)} usoc_{te} \left( \frac{distance_{gg'}}{speed_l} \right) Q_{ilgg't} \qquad \forall t, e \quad (2.32)$$

#### 2.2.3.5 Financial Risk

The traditional approach to address optimization under uncertainty relies on formulating a single-objective optimization problem where the expected performance of the system is the objective to be optimized. This strategy does not allow controlling the variability of the objective function in the uncertain space. In other words, optimizing the expected economic performance of a SC does no imply that the process will yield better results at a certain level considering the whole cost distribution. The underlying idea in risk management is to incorporate the trade-off between financial risk and expected cost within the decision-making procedure. This gives rise to a multi-objective optimization problem in which the expected performance and a specific risk measure are the objectives considered. The solution of such a problem is given by a set of Pareto solutions that represent the optimal trade-off between expected performance and risk level.

In this work, the probability of meeting unfavorable scenarios is controlled by adding the worst case cost as an additional objective to be minimized. This metric is easy to implement and leads to a good numerical performance in stochastic models, as shown by Bonfill et al. (2004). The worst case can be easily determined from the maximum cost attained over all the scenarios:

$$\mathrm{WC} \geq \mathrm{TDC_e} \qquad \forall \mathrm{e} \tag{2.33}$$

The inclusion of the worst case as an alternative objective to be minimized along with the expected total cost leads to the following bi-criterion MILP formulation:

$$\begin{aligned} \text{(MOP)} \qquad & \min_{x,y,z} && (E[TDC](x,y,z), WC(x,y,z)) \\ & \text{s.t.} && \text{equations 2.1 to 2.33} \\ & && x \in \Re,\ y \in \{0,1\},\ z \in N \end{aligned}$$

where $x$, $y$ and $z$ denote the continuous, binary and integer variables of the problem, respectively. The aforementioned multi-objective problem can be solved by standard algorithms for multi-objective optimization such as the $\epsilon$-constraint or the weighted-sum method (Ehrgott, 2005). The weighted-sum method is only rigorous for the case of Pareto sets, whereas the $\epsilon$-constraint method is also rigorous for the non-convex case, which turns out to be our case. This method entails solving a set of instances of problem (P) corresponding to different values of the auxiliary parameter $\epsilon$:

$$\begin{aligned} \text{(P)} \qquad & \min_{x,y,z} && E[TDC](x,y,z) \\ & \text{s.t.} && \text{equations 2.1 to 2.33} \\ & && WC(x,y,z) \leq \epsilon \\ & && \underline{\epsilon} \leq \epsilon \leq \overline{\epsilon} \\ & && x \in \Re,\ y \in \{0,1\},\ z \in N \end{aligned}$$

where the lower and upper bounds within which the epsilon parameter must fall (i.e.$\epsilon \in [\underline{\epsilon}, \overline{\epsilon}]$) are obtained from the optimization of each separate scalar objective.

# Chapter 3

# Ethanol

This second example addresses the design of supply chains for ethanol production (Kostina et al., 2010). In this case, uncertainties are not included in the model for the sake simplicity.

## 3.1 Problem Statement

To formally state the SC design problem, we consider a generic three-echelon SC (production-storage-market) like the one depicted in Figure 3.1. This network includes a set of production and storage facilities, and final markets. We assume that we are given a specific region of interest that is divided into a set of sub-regions in which the facilities of the SC can be established in order to produce and deliver final products to the customers. In general, these sub-regions, which are regarded as potential locations for the SC entities, will be defined according to the administrative division of a country. The SC design problem can then be formally stated as follows:

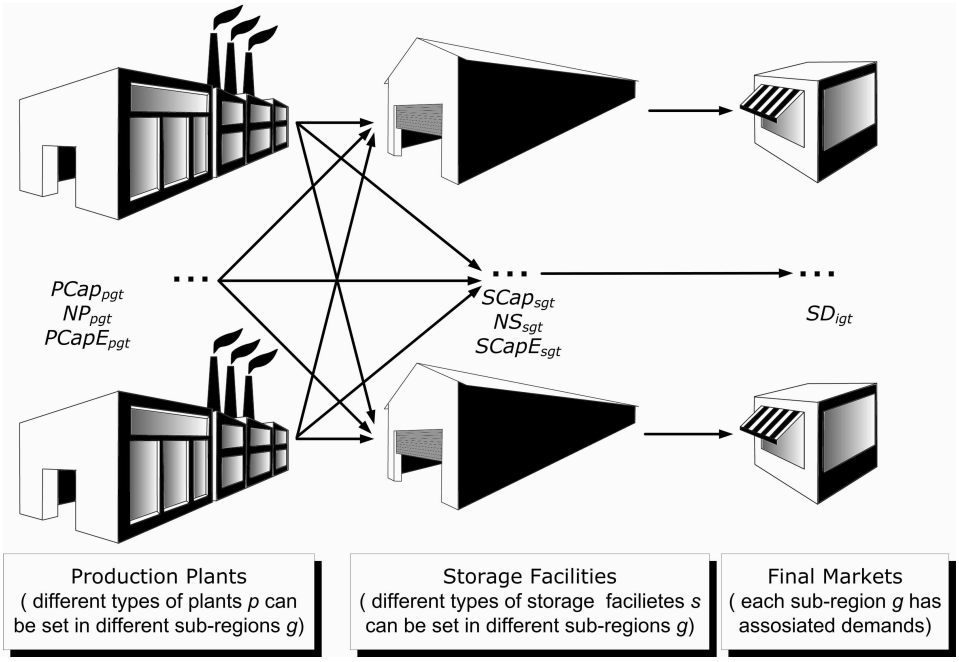


Figure 3.1: The structure of the three-echelon SC (Ethanol)

*Given are a fixed time horizon, product prices, cost parameters for production, storage and transportation of materials, demand forecast, the tax rate, capacity data for plants, storages and transportation links, fixed capital investment data, interest rate, storage holding period and landfill tax.*

The goal is to determine the configuration of a three-echelon bioethanol network and the associated planning decisions with the goal of maximizing the economic performance calculated over the entire useful life of the SC. Decisions to be made include the number, location and capacity of production plants and warehouses to be set up in each sub-region, their capacity expansion policy for a given forecast of prices and demand over the planning horizon, the transportation links and transportation modes of the network, and the production rates and flows of feed stocks, wastes and final products.

## 3.2 Mathematical Model

In this section, we present a mathematical model that considers the specific features of the sugar cane industry, while still being general enough to be easily adapted to any other industrial SC. Particularly, our model is based on the MILP formulation introduced by Almansoori and Shah (2006), and Guillén-Gosálbez et al. (2010), which addresses the design of hydrogen SCs. Furthermore, the model follows the SC formulation developed by Guillén-Gosálbez and Grossmann for the case of petrochemical SCs (Guillén-Gosálbez and Grossmann, 2009, 2010), in the way in which the mass balances are handled.

### 3.2.1 Production plants

Sugar cane is the leading feedstock for bioethanol production in Argentina as well as in most of the tropical regions all over the world (e.g., Brazil, India, China, etc.). The juice is extracted from sugar cane mainly by milling. From this step sugar cane juice can be treated in different ways.
Sugar factories can use this juice to produce white sugar and raw sugar. There are two technologies realizing "sugar cane-to-sugar" pathway: one of them generates molasses (T1) as a byproduct, whereas the other one provides a secondary honey (T2) in addition to sugars.
These two kinds of byproducts are distinguished by their sucrose content. Molasses is a viscous dark honey whose low sucrose content cannot be separated by crystallization, while secondary honey is a honey with a larger amount of sucrose that leaves the sugar mill before being exhausted by crystallization. Anhydrous ethanol can be produced by fermentation and following dehydration of different process streams: molasses (T3), honey (T4) and sugar cane juice (T5). According to this, the model considers five different technologies, two for sugar production and three types of distilleries.
The details of each technology, including the mass balance coefficients, are shown in Figure 3.2. We assume that bagasse is completely utilized for internal purposes, so the model includes a set of nine materials: sugar cane, ethanol, molasses, honey, white sugar, raw sugar, vinasse type 1, vinasse type 2 and vinasse type 3. Each plant type incurs fixed capital and operating costs and may be expanded in capacity over time in order to follow a specific demand pattern. The establishment of a plant type is determined from the demand of the sub-region, the capacity that the sub-region has to fulfill its internal needs and the cost data.

### 3.2.2 Storage facilities

The model includes two different types of storage facilities: warehouses for liquid products and warehouses for solid materials. Each storage facility type has fixed capital and unit storage costs, and lower and upper limits for capacity expansions. The storage capacity might be expanded in order to follow changes in the demand as well as in the supply.

### 3.2.3 Transportation modes

Transportation links allow to deliver final products to customers, supply the plants with raw materials and dispose the process wastes. The model assumes that the transportation tasks can be performed by three types of trucks: heavy trucks with open-box bed for sugar cane, lorries for sugar and tank trucks for liquid products. Each type of transportation mode has fixed capital and unit transportation costs and lower and upper limits for its capacity. The number and capacity of the transportation links can also vary over time in order to follow a given demand pattern.

### 3.2.4 General constraints

We next describe the main mathematical constraints of the model, which have been derived bearing in mind the particular features of the sugar cane industry in Argentina.

#### 3.2.4.1 Materials balance

The starting point for all design is the material balance. Particularly, the law of conservation of mass must be satisfied in every sub-region. The overall mass balance for each sub-region is represented by Eq.(3.1). In accordance with it, for every material form $i$, the initial inventory kept in sub-region $g$ from previous period ($ST_{isgt-1}$) plus the amount produced ($PT_{igt}$), the amount of raw materials purchased ($PU_{igt}$) and the input flow rate from other facilities in the SC ($Q_{ilg'gt}$) must equal the final inventory ($ST_{isgt}$) plus the amount delivered to customers ($DTS_{igt}$) plus the output flow to

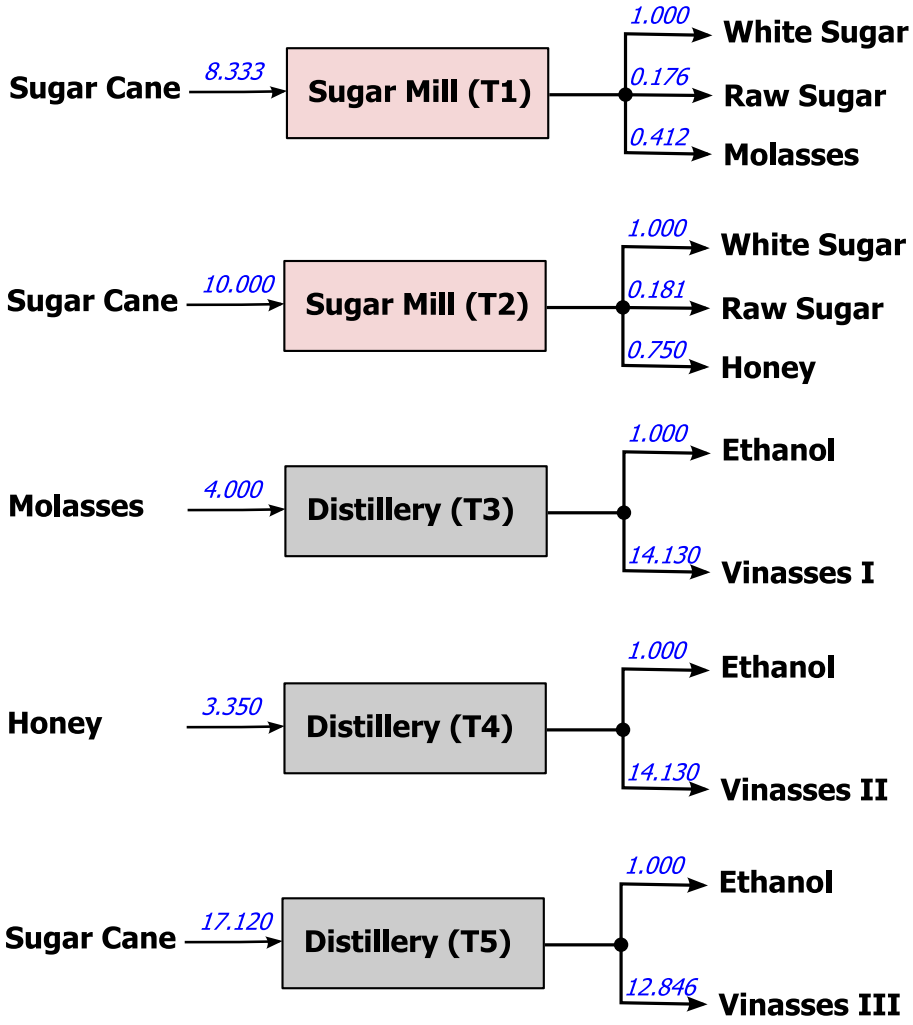


Figure 3.2: The technology with mass balance coefficients

other sub-regions ($Q_{ilgg't}$) and the amount of waste ($W_{igt}$).

$$\sum_{s\in SI(i)} ST_{isgt-1} + PT_{igt} + PU_{igt} + \sum_{l\in LI(i)}\sum_{g'\neq g} Q_{ilg'gt} = \sum_{s\in SI(i)} ST_{isgt} + DTS_{igt} + \sum_{l\in LI(i)}\sum_{g'\neq g} Q_{ilgg't} + W_{igt} \qquad \forall i,g,t \tag{3.1}$$

In this equation, $SI(i)$ represents the set of technologies that can be used to store product $i$, whereas $LI(i)$ are the set of transportation modes that can transport product $i$. Furthermore, the amount of products delivered to the final markets should be less than or equal to the actual demand ($SD_{igt}$):

$$DTS_{igt} \leq SD_{igt} \qquad \forall i,g,t \tag{3.2}$$

#### 3.2.4.2 Production

The total production rate of material $i$ in sub-region $g$ is determined from the particular production rates ($PE_{ipgt}$) of each technology $p$ installed in the sub-region:

$$PT_{igt} = \sum_{p} PE_{ipgt} \qquad \forall i,g,t \tag{3.3}$$

As can be observed in Figure 3.2,the material balance coefficients of the main products (white sugar and ethanol) can be normalized to 1. The production rates of byproducts and raw materials for each technology can then be calculated from the material balance coefficients, $\rho_{pi}$, and the production rates of the main products:

$$PE_{ipgt} = \rho_{pi} PE_{i'pgt} \qquad \forall i,p,g,t \qquad \forall i' \in IM(p) \tag{3.4}$$

In this equation, $IM(p)$ represents the set of main products associated with each technology. The production rate of each technology $p$ in sub-region $g$ is limited by the minimum desired percentage of the available technology that must be utilized, $\tau$, multiplied by the existing capacity (represented by the continuous variable $PCap_{pgt}$) and the maximum capacity:

$$\tau PCap_{pgt} \leq PE_{ipgt} \leq PCap_{pgt} \qquad \forall i,p,g,t \tag{3.5}$$

The capacity of technology $p$ in any time period $t$ is calculated adding the existing capacity at the end of the previous period to the expansion in capacity, $PCapE_{pgt}$, carried out in period $t$:

$$PCap_{pgt} = PCap_{pgt-1} + PCapE_{pgt} \qquad \forall p,g,t \tag{3.6}$$

Eq.(3.7) bounds the capacity expansion $PCapE_{pgt}$ between upper and lower limits, which are calculated from the number of plants installed in the sub-region ($NP_{gpt}$) and the minimum and maximum capacities associated with each technology $p$ ($\underline{PCap_p}$ and $\overline{PCap_p}$, respectively).

$$\underline{PCap_p} NP_{pgt} \leq PCapE_{pgt} \leq \overline{PCap_p} NP_{pgt} \qquad \forall p, g, t \tag{3.7}$$

The purchases of sugar cane are limited by the capacity of the existing sugar cane plantation in sub-region $g$ and time interval $t$:

$$PU_{igt} \leq CapCrop_{gt} \qquad \forall i = Sugar\ cane, g, t \tag{3.8}$$

#### 3.2.4.3 Storage

As occurs with plants, the storage capacity is limited by lower and upper bounds, which are given by the number of storage facilities installed in sub-region $g$ ($NS_{sgt}$) and the minimum and maximum storage capacities ($\underline{SCap_s}$ and $\overline{SCap_s}$, respectively) associated with each storage technology:

$$\underline{SCap_s} NS_{sgt} \leq SCapE_{sgt} \leq \overline{SCap_s} NS_{sgt} \qquad \forall s, g, t \tag{3.9}$$

The capacity of a storage technology $s$ in any time period $t$ is determined from the existing capacity at the end of the previous period and the expansion in capacity in the current period ($SCapE_{sgt}$):

$$SCap_{sgt} = SCap_{sgt-1} + SCapE_{sgt} \qquad \forall s, g, t \tag{3.10}$$

The storage capacity should be enough to store the total inventory ($ST_{isgt}$) of product $i$ during time interval $t$:

$$\sum_{i \in IS(s)} ST_{isgt} \leq SCap_{sgt} \qquad \forall s, g, t \tag{3.11}$$

In this equation, $IS(s)$ denote the set of products that can be stored by technology $s$. During steady-state operation, the average inventory ($AIL_{igt}$) is a function of the amount delivered to customers and the storage period $\beta$:

$$AIL_{igt} = \beta DTS_{igt} \qquad \forall i, g, t \tag{3.12}$$

The storage capacity ($SCap_{sgt}$) that should be established in a sub-region in order to cope with fluctuations in both supply and demand, is twice the summation of the average inventory levels of products $i$ (Simchi-Levi et al., 2003).

$$2AIL_{igt} \leq \sum_{s \in SI(i)} SCap_{sgt} \qquad \forall i, g, t \tag{3.13}$$

#### 3.2.4.4 Transportation

The existence of a transportation link between two sub-regions $g$ and $g'$ is represented by a binary variable $X_{lgg't}$ which equals 1 if a transportation link is established between the two sub-regions and 0 otherwise. The definition of this variable is enforced via Eq.(3.14), which constraints the materials flow between minimum and maximum allowable capacity limits ($\underline{Q_l}$ and $\overline{Q_l}$, respectively):

$$\underline{Q_l} X_{lgg't} \leq \sum_{i \in IL(l)} Q_{ilgg't} \leq \overline{Q_l} X_{lgg't} \qquad \forall l, t, g, g'(g' \neq g) \tag{3.14}$$

In this equation, $IL(l)$ represents the set of materials that can be transported via transportation mode $l$. Furthermore, a sub-region can either import or export material $i$, but not both at the same time:

$$X_{lgg't} + X_{lg'gt} = 1 \qquad \forall l, t, g, g'(g' \neq g) \tag{3.15}$$

### 3.2.5 Objective function

Usage of $NPV$ as an objective function is a widely-spread approach in investment planning. In most cases it results in a linear model, which can be effectively solved by standard branch-and-bound methods. However, the $NVP$ measure does not account appropriately for the rate at which the investment is recovered because it tends to add investment that has marginal or meaningless returns. Bagajewicz (2008) pointed out that additional procedures and measures are needed in planning problems. Particularly, the return of investment ($ROI$) is a more appropriate key performance indicator when there are other investment alternatives competing for the same capital. In the context of a SC design problem like the one addressed in this article, this metric can be determined as the ratio between the average cash flows ($CF_t$) and the fixed capital investment $FCI$:

$$ROI = \frac{(\sum_t CF_t)/T}{FCI} \tag{3.16}$$

As observed, the introduction of the $ROI$ as the economic indicator to be maximized gives rise to a MINLP formulation with a nonconvex objective function. Given that the linear NPV-based approach already has computational issues that this paper attempts to ameliorate, following Bagajewicz (2008) we resort to approximate the solution of this problem by applying a heuristic procedure that relies on solving a series of MILPs that maximize the $NPV$ for

different upper bounds on $FCI$. As discussed in Bagajewicz (2008), from these results one can identify solutions close to the maximum $ROI$ one.

The $NPV$ can be determined from the discounted cash flows generated in each of the time intervals $t$ in which the total time horizon is divided:

$$NPV = \sum_{t} \frac{CF_t}{(1+ir)^{t-1}} \tag{3.17}$$

In this equation, $ir$ represents the interest rate. The cash flow that appears in Eq.(3.17) in each time period is computed from the net earnings $NE_t$ (i.e., profit after taxes), and the fraction of the total depreciable capital ($FTDC_t$) that corresponds to that period as follows:

$$CF_t = NE_t - FTDC_t \qquad t = 1, ..., T-1 \tag{3.18}$$

In the calculation of the cash flow of the last time period ($t = T$), it is necessary to take into account the fact that part of the total fixed capital investment may be recovered at the end of the time horizon. This amount, which represents the salvage value of the network ($sv$), may vary from one type of industry to another.

$$CF_t = NE_t - FTDC_t + svFCI \qquad t = T \tag{3.19}$$

The net earnings are given by the difference between the incomes ($Rev_t$) and the facility operating ($FOC_t$), and transportation cost ($TOC_t$), as it is stated in Eq.(3.20):

$$NE_t = (1-\varphi)(Rev_t - FOC_t - TOC_t) + \varphi DEP_t \qquad \forall t \tag{3.20}$$

In this equation, $\varphi$ denotes the tax rate. The revenues are determined from the sales of final products and the corresponding prices ($PR_{igt}$):

$$Rev_t = \sum_{i \in SEP} \sum_{g} DTS_{igt} PR_{igt} \qquad \forall t \tag{3.21}$$

In this equation $SEP$ represents the set of materials $i$ that can be sold. The facility operating cost is obtained by multiplying the unit production and storage costs ($UPC_{ipgt}$ and $USC_{isgt}$, respectively) by the corresponding production rates and average inventory levels, respectively. This term includes also the disposal cost ($DC_t$):

$$\begin{aligned} FOC_t = & \sum_{i} \sum_{g} \sum_{p \in IM(p)} UPC_{ipgt} PE_{ipgt} + \\ & \sum_{i} \sum_{g} \sum_{s \in IS(s)} USC_{isgt} AIL_{igt} + \\ & DC_t \qquad \forall t \end{aligned} \tag{3.22}$$

The disposal cost is a function of the amount of waste and landfill tax ($LT_{ig}$):

$$DC_t = \sum_i \sum_g W_{igt} LT_{ig} \qquad \forall t \tag{3.23}$$

The transportation cost includes the fuel ($FC_t$), labour ($LC_t$), maintenance ($MC_t$) and general ($GC_t$) costs:

$$TOC_t = FC_t + LC_t + MC_t + GC_t \qquad \forall t \tag{3.24}$$

The fuel cost is a function of the fuel price ($FP_{lt}$) and fuel usage:

$$FC_t = \sum_g \sum_{g' \neq g} \sum_l \sum_{i \in IL(l)} \left[ \frac{2EL_{gg'}\ Q_{ilgg't}}{FE_l\ TCap_l} \right] FP_{lt} \qquad \forall t \tag{3.25}$$

In Eq.(3.25), the fractional term represents the fuel usage, and is determined from the total distance traveled in a trip ($2EL_{gg'}$ ), the fuel consumption of transport mode $l$ ($FE_l$) and the number of trips made per period of time ($\frac{Q_{ilgg't}}{TCap_l}$). Note that this equation assumes that the transportation units operate only between two predefined sub-regions. Furthermore, as shown in Eq.(3.26), the labor transportation cost is a function of the driver wage ($DW_{lt}$) and total delivery time (term inside the brackets):

$$LC_t = \sum_g \sum_{g' \neq g} \sum_l DW_{lt} \sum_{i \in IL(l)} \left[ \frac{Q_{ilgg't}}{TCap_l} \left( \frac{2EL_{gg'}}{SP_l} + LUT_l \right) \right] \qquad \forall t \tag{3.26}$$

The maintenance cost accounts for the general maintenance of the transportation systems and is a function of the cost per unit of distance traveled ($ME_l$) and total distance driven:

$$MC_t = \sum_g \sum_{g' \neq g} \sum_l \sum_{i \in IL(l)} ME_l \frac{2EL_{gg'} Q_{ilgg't}}{TCap_l} \qquad \forall t \tag{3.27}$$

Finally, the general cost includes the transportation insurance, license and registration, and outstanding finances. It can be determined from the unit general expenses ($GE_{lt}$) and number of transportation units ($NT_{lt}$), as follows:

$$GC_t = \sum_l \sum_{t' \leq t} GE_{lt} NT_{lt'} \qquad \forall t \tag{3.28}$$

The depreciation term is calculated with the straight-line method:

$$DEP_t = \frac{(1 - sv)FCI}{T} \qquad \forall t \tag{3.29}$$

where $FCI$ denotes the total fixed cost investment, which is determined from the capacity expansions made in plants and warehouses as well as the purchases of transportation units during the entire time horizon as follows:

$$\begin{aligned} FCI = & \sum_{p}\sum_{g}\sum_{t}(\alpha^{PL}_{pgt}NP_{pgt} + \beta^{PL}_{pgt}PCapE_{pgt}) + \\ & \sum_{s}\sum_{g}\sum_{t}(\alpha^{S}_{sgt}NS_{sgt} + \beta^{S}_{sgt}SCapE_{sgt}) + \\ & \sum_{l}\sum_{t}(NT_{lt}TMC_{lt}) \end{aligned} \tag{3.30}$$

Here, the parameters $\alpha^{PL}_{pgt}$ , $\beta^{PL}_{pgt}$ and $\alpha^{S}_{sgt}$ , $\beta^{S}_{sgt}$ are the fixed and variable investment terms corresponding to plants and warehouses, respectively. On the other hand, $TMC_{lt}$ is the investment cost associated with transportation mode $l$. The average number of trucks required to satisfy a certain flow between different sub-regions is computed from the flow rate of products between the sub-regions, the transportation mode availability ($avl_l$), the capacity of a transport container, the average distance traveled between the sub-regions, the average speed, and the loading/unloading time, as stated in Eq.(3.31):

$$\sum_{t\leq T}NT_{lt} \geq \sum_{i\in IL(l)}\sum_{g}\sum_{g'\neq g}\sum_{t}\frac{Q_{ilgg't}}{avl_l TCap_l}\left(\frac{2EL_{gg'}}{SP_l} + LUT_l\right) \qquad \forall l \tag{3.31}$$

The total amount of capital investment can be constrained to be lower than an upper limit, as stated in Eq.(3.32):

$$FCI \leq \overline{FCI} \tag{3.32}$$

Finally, the model assumes that the depreciation is linear over the time horizon. Thus, the depreciation term ($FTDC_t$) is calculated as follows:

$$FTDC_t = \frac{FCI}{T} \qquad \forall t \tag{3.33}$$

Finally, the overall MILP formulation is stated in compact form as follows:

$$\begin{aligned} & \max_{x,X,N} NPV(x,X,N) && \text{(P)} \\ & \text{s.t. constraints 1-32} \\ & x \subset \mathbb{R}, \quad X \subset \{0,1\}, \quad N \subset \mathbb{Z}^{+} \end{aligned}$$

Here, $x$ denotes the continuous variables of the problem (capacity expansions, production rates, inventory levels and materials flows), $X$ represents the binary variables (i.e., establishment of transportation links), and $N$ is the set of integer variables denoting the number of plants, storage facilities and transportation units of each type selected.

# Chapter 4

# Solution Method: LNS

The models presented before lead to large scale MILPs that are hard to be solved in short CPU times, specially as the number of grids and time periods increase. Hence, tailored solution methods are required to expedite their solutions, so decision-makers can analyze a wide range of potential alternatives in short CPU times.

Particularly, this chapter we will go into more detail for what concerns the description of a customized Large Neighborhood Search algorithm, that exploits the particular features of the SCM models described before. First, we make a description of the algorithm in general. Second, we describe the implemented version of LNS.

## 4.1 Large Neighborhood Search

A crucial decision when dealing with neighborhood search is the choice of a neighborhood function. In general, when the employed neighborhood function generates rather small neighborhoods, the corresponding local search method is fast. On the downside, however, the average quality of the local minima is rather low. If, in contrast, the generated neighborhoods are rather large, local search is slower. However, the average quality of the local minima is rather high. Therefore, the general aim is to develop neighborhoods for which local search is not too slow, and the average quality of the minima is reasonably high. In other words, neighborhoods define a balance between the speed of local search and the quality of the obtained solution.

A special type of neighborhood is employed in so-called *very large-scale neighborhood search (VLSN)* algorithms (Ahuja et al., 2002). In particular, these algorithms make use of exponential-size neighborhoods. Let $I$ be an

instance of a problem, and $x \in X(I)$ a solution to this problem, where $X(I)$ is the set of all feasible solutions of instance $I$. Given a neighborhood function $N(\cdot)$, then the size of the neighborhood concerning instances of size $n$ can be defined as follows:

$$f(n) = max\{|N(x)| \mid x \in X(I) \text{ and } I \in F(n)\} \ ,$$

where $F(n)$ is the set of all instances of size $n$ of the problem under study. A neighborhood search algorithm is considered as belonging to the class of VSLN algorithms if $f(n)$ is of the order $\bigcirc(2^n)$. According to (Ahuja et al., 2002) VLSN algorithms are divided into three main categories: (1) depth variable methods, (2) network flow based improvement methods, and (3) methods based on restriction to subclasses solvable in polynomial time.

For many combinatorial optimization problems the field of mathematical programming and (mixed) integer linear programming (MIP) in particular provides powerful tools (see, for example, (Nemhauser and Wolsey, 1988b; Wolsey, 1998)). MIP-solvers are, in general, based on some sort of tree search, but further include the solution of linear programming relaxations of a given MIP model for the problem at hand (besides primal heuristics) in order to obtain lower and upper bounds. Frequently, such MIP approaches are highly effective for small to medium sized instances of hard problems; however, they often do not scale well enough to large instances relevant in practice. MIP-solvers might therefore be very useful for searching large neighborhoods within a metaheuristic framework. Especially the availability of effective general purpose MIP-solvers and their relatively easy applicability makes this approach particularly interesting in practice, providing the problem at hand can be expressed by a MIP model.

Such an approach, labelled as the *LNS metaheuristic*, was proposed in (Shaw, 1998b). Hereby, the definition of the large neighborhoods may be done in different ways. In the simplest case, an appropriate portion of the decision variables is fixed to the values they have in the current solution, and only the remaining ("free") variables are optimized by the MIP-solver. If the MIP-solver finds an improved solution, it becomes the new current solution, a new large neighborhood is defined around it, and the process is iterated. Obviously, the selection of the variables that remain fixed and the ones that are subject to optimization, respectively, plays a crucial role: The number of free variables directly implies the size of the neighborhood. Too restricted neighborhoods—that is, subproblems—are unlikely to yield

**Algorithm 14** The framework of the LNS metaheuristic

**Require:** a feasible solution $x$
**Ensure:** $x^b$
  $x^b = x$
  **while** stopping conditions do not apply **do**
    $x' = repair(destroy(x))$
    **if** $accept(x', x)$ **then**
      $x = x'$
    **end if**
    **if** $cost(x') < cost(x^b)$ **then**
      $x^b = x'$
    **end if**
  **end while**

improved solutions, while too large neighborhoods might result in excessive running times for solving the subproblem by the MIP-solver. Therefore, a strategy for dynamically adapting the number of free variables is sometimes used. Furthermore, the variables to be optimized might be selected either purely at random or in a more sophisticated, guided way by considering the variables' potential impact on the objective function and their relatedness.

The pseudo-code of LNS is presented in algorithm 4.1. It works roughly as follows: First, an initial solution $x$ is generated. This may be done randomly, or by means of an appropriate heuristic. Then the initial solution is stored in variable $x^b$, which—at all times—contains the best solution found so far by the algorithm. Then, at each iteration of the algorithm, method $destroy(\cdot)$ is applied to the current solution $x$. This method frees some of the decision variables of $x$. Afterwards, method $repair(\cdot)$ is applied to the resulting partial solution in order to find the best setting for the decision variables that have no value. This process results in a (possibly) new solution $x'$. Finally, method $accept(x', x)$ chooses among $x$ and $x'$ the current solution for the next iteration. In case a new best solution has been found, it is stored in $x^b$.

In the following we explain the characteristics of the involved functions in more detail:

- Function $accept(x', x)$: In the original algorithm as proposed in (Shaw, 1998b) the only possibility that was considered concerns the acceptance of the best solution among $x$ and $x'$ as the new current solution. However, in later versions of the LNS metaheuristic (see, for exam-

ple, (Ropke and Pisinger, 2006)) acceptance criteria similar to the one used in simulated annealing were applied. That is, if the newly produced solution $x'$ satisfies that $cost(x') > cost(x)$ then it still may be accepted with probability:

$$e^{-(cost(x')-cost(x))/T}$$

Hereby, $T > 0$ is the current *temperature*, which is initialized to some positive value at the start of the algorithm and then gradually decreased.

- Function $repair(\cdot)$: There are several possible ways of implementing this method: Apart from using a MIP-solver, as mentioned above, a partial solution might also be completed by means of a heuristic. This option is, in general, the fastest one. On the other side, exact methods such as dynamic programming or constraint programming techniques might be employed for completing the partial solution in the best possible way. However, notice that in this work we have used a general purpose MIP-solver.

- Function $destroy(\cdot)$: Together with the repair method, this function is the most important aspect of the LNS metaheuristic. As mentioned above, a very important choice when implementing this method is the *degree of destruction*: if only a very small part of the current solution is destroyed then LNS may have trouble exploring the whole search space. If a very large part of solution is destroyed then the LNS metaheuristic almost degrades into a repeated exact optimization algorithm. Shaw, in (Shaw, 1998b), proposed to gradually increase the degree of destruction, while in (Ropke and Pisinger, 2006) the authors chose the degree of destruction randomly. In order for the entire space to be reached, the destruction method should not only focus on a particular component of the solution.

## 4.2 Large Neighborhood Search Implemented

In this section we will show all the details of the LNS implementation for case studied and data structures necessary:

- **Solution:** As explained in chapter 1 a mathematical model is a set of values that are defined by constraints. To define these constraints

variables are used. In this context when we speak about variables, we refers to a mathematical variables in the mathematical programming field. Our algorithm works with models manipulating their variables. Therefore the models have been encapsulated in a data structure so our algorithm can work with them. The data structure that represents the solution is a hash table of variables called $s$ (see fig 4.1). The $lab$ variable is denoted by $s[lab]$, where $lab$ is a label that represents the name of $lab$ variable. A variable is a triplet $\langle v, lb, ub \rangle$, where $v$ is the value of the variable, $lb$ is the lower bound and $ub$ is the upper bound. The value of a variable could be a integer, real or Boolean. It always holds $lb \leq v \leq ub$. A variable is fixed if $v = lb = ub$. A variable is released when we return to the initial values of the variables bounds $lb = LB$ and $ub = UB$.

- **Model:** The model is a mathematical programming model. It consists of a set of equations and inequalities that define the search hyperspace, which must be enclosed.
- **Objective Function:** The objective function is a variable of type real.

The algorithm requires the following input:

- The model $mdl$.
- A maximum time of algorithm execution $t_{max}$.
- A maximum number of iterations $it_{max}$.
- A maximum number of variables to be released $n_{max}$.
- A maximum number of attempts $m_{max}$.

The algorithm works as follows:

1. First of all we generate the initial solution and the value of the objective function. The initial solution is a feasible solution with all the variables fixed, ie $\forall lab : s[lab].lb = s[lab].v = s[lab].ub$.
2. In the main loop while $end$ not equal to $true$, for each trial $m$, we do the following: First, we choose a set of $n$ random variables $V$ to release. Second, we copy the solution $s$ to $s'$, and release the $n$ variables of the solution $s'$. Third, we invoke the solve. The solver tries to improve

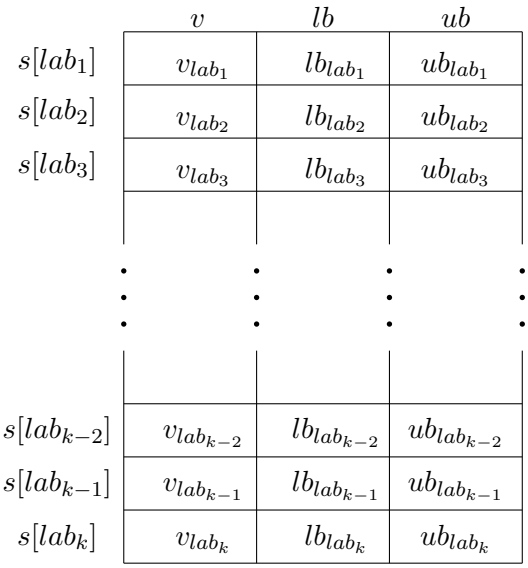


Figure 4.1: Data Structure of the Solution is a hash table

the solution changing the value of the variables released. The solve invocation gets the run time $t$ of solver, the new value of the objective function $fo'$, and a new solution. If the objective function value is better than the current value, then we update the best solution, the objective function, and variable *improved* assign to *true*. Finally, we increase the number of iterations $it$, the current time $ct$ and if the number of iterations $it$ and the time $ct$ is greater than their maximum, then the variable *end* assign to *true*.

Now go into more detail on each of the functions of the algorithm. To better understand our algorithm is important to differentiate between the case of HYDROGEN or the case of Ethanol:

### 4.2.1 Hydrogen

In the case of HYDROGEN, we focus on this set of variables (see chapter 2):

- $PR_{igpt}$: Number of factories of Hydrogen form $i$ with the manufacturing technology $p$ in place $g$ at period $t$.

**Algorithm 15** LNS for Supply Chain

**Require:** $mdl$ is the model **and** $t_{max} > 0$ **and** $it_{max} > 0$ **and** $m_{max} > 0$ **and** $n_{max} > 0$
**Ensure:** $s$ { The best solution found }

```
1:  ⟨s, fo⟩ := initial_solution(mdl);
2:  end := false ; it := 0 ; ct := 0
3:  while not end do
4:     m := 1; improved := false;
5:     while m ≤ m_max and not improved do
6:        n := 1;
7:        while n ≤ n_max and not improved do
8:           V := choose_random_vars_to_release(n); { V is a set of vars}
9:           s' := release_vars(s, V);
10:          ⟨t, fo', s'⟩ := solve(mdl, s');
11:          if better(fo, fo') then
12:             fo' := fo ; s := s' ; improved := true;
13:          end if
14:          ct := ct + t ; it := it + 1;
15:          if ct ≥ t_max or it ≥ it_max then
16:             end := true;
17:          end if
18:          n := n + 1;
19:       end while
20:       m := m + 1;
21:    end while
22: end while
```

- $N_{gst}^{ST}$: Number of stores of Hydrogen with storage technology $s$ in place $g$ at period $t$.

- $X_{gg'lt}$: Equals 1 if there is a link between $g$ and $g'$ with the transport $l$ at the period $t$, otherwise 0.

In the case of HYDROGEN, the implementation of the methods is as follow:

- $initial\_solution(mdl)$: The generation of the initial solution in HYDROGEN is as follow:

    1. We obtain the average demand for each location $g$.

2. Solve the problem for period $t = 1$, then obtain the value of $PR_{igpt}$, $N^{ST}_{gst}$ and $X_{gg'lt}$ for that period.
3. Fix the values of $PR_{igpt}$, $N^{ST}_{gst}$, $X_{gg'lt}$ (make them equal to those associated to period $t = 1$) and solve considering all the periods, in order to obtain the initial solution.

- *choose_random_vars_to_release*$(n)$: This function selects $n$ places. In the case of HYDROGEN are the autonomous communities of Spain, which are numbered from 1 to 19 (see 4.2).

- *release_vars*$(s, V)$: This method returns a copy of the solution $s$ with some variables released. The variables are released as follows:
For every variable $PR_{igpt}$ or $N^{ST}_{gst}$ or $X_{gg'lt}$ where $g \in v$ put the lower bounds

$$s[PR_{igpt}].lb = LB$$
$$s[N^{ST}_{gst}].lb = LB$$
$$s[X_{gg'lt}].lb = 0$$

and put the upper bound

$$s[PR_{igpt}].ub = UB$$
$$s[N^{ST}_{gst}].ub = UB$$
$$s[X_{gg'lt}].ub = 1$$

.

- *solve*$(mdl, s')$: Given a model of type $mdl$ MILP, and a solution $s'$, this function solves the model. The Solver tries to improve the solution changing the value of variables released function implements a branch & bound, this project has used a commercial software called CPLEX.

- *better*$(fo, fo')$: Given two objective functions $fo$ y $fo'$ determines which is the best. The best solution is the smallest.

### 4.2.2 Ethanol

In the case of ETHANOL, we focus on this set of variables (see chapter 3):

- $NP_{gpt}$: Number of factories of Ethanol with the manufacturing technology $p$ in place $g$ at period $t$.

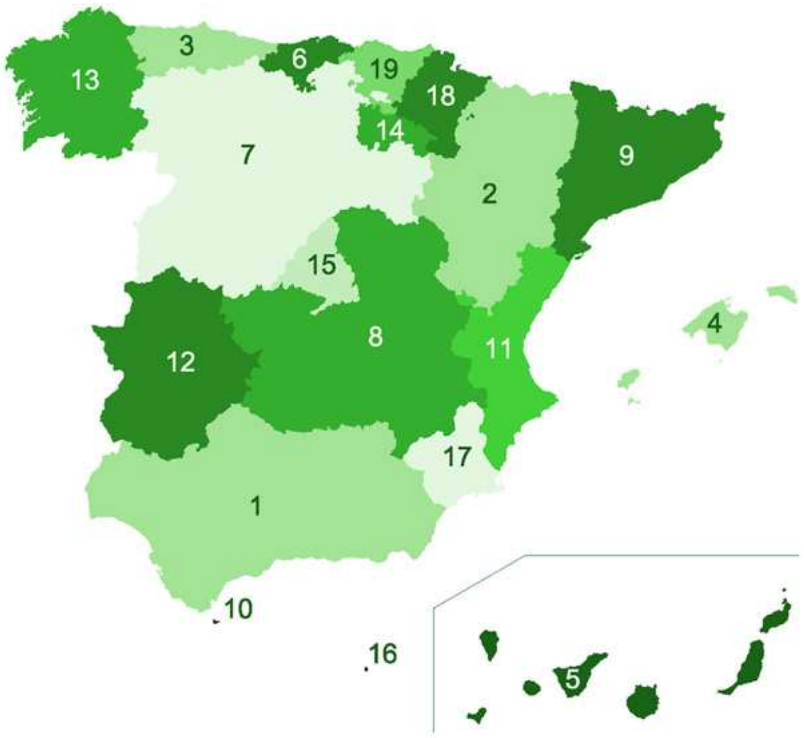


Figure 4.2: Autonomous communities of Spain.

- $NS_{sgt}$: Number of stores of Ethanol with storages technology $s$ in the place $g$ at the period $t$.
- $X_{lgg't}$: Equals 1 if there is a link between $g$ and $g'$ with the transport $l$ at period $t$, otherwise 0.

In the case of ETHANOL, the implementation of the methods is as follow:

- $initial\_solution(mdl)$: We obtain the initial solution solving the problem where integer variables are treated as continuous (relaxed model).
- $choose\_random\_vars\_to\_release(n)$: This function selects $n$ places. In the case of ETHANOL are the provinces of Argentine, which are numbered from 1 to 25 (see 4.3).
- $release\_vars(s, V)$: This method returns a copy of the solution $s$ with some variables released. The variables are released as follows:
  For every variable $NP_{gpt}$ or $NS_{sgt}$ or $X_{lgg't}$ where $g \in v$ put the lower bounds

$$s[NP_{gpt}].lb = LB$$
$$s[NS_{sgt}].lb = LB$$
$$s[X_{lgg't}].lb = 0$$

  and put the upper bound

$$s[NP_{gpt}].ub = UB$$
$$s[NS_{sgt}].ub = UB$$
$$s[X_{lgg't}].ub = 1$$

  .
- $solve(mdl, s')$: Is the same as HYDROGEN. This project has used a commercial software called CPLEX.
- $better(fo, fo')$: Given two objective functions $fo$ y $fo'$ determines which is the best. The best solution is the greatest.

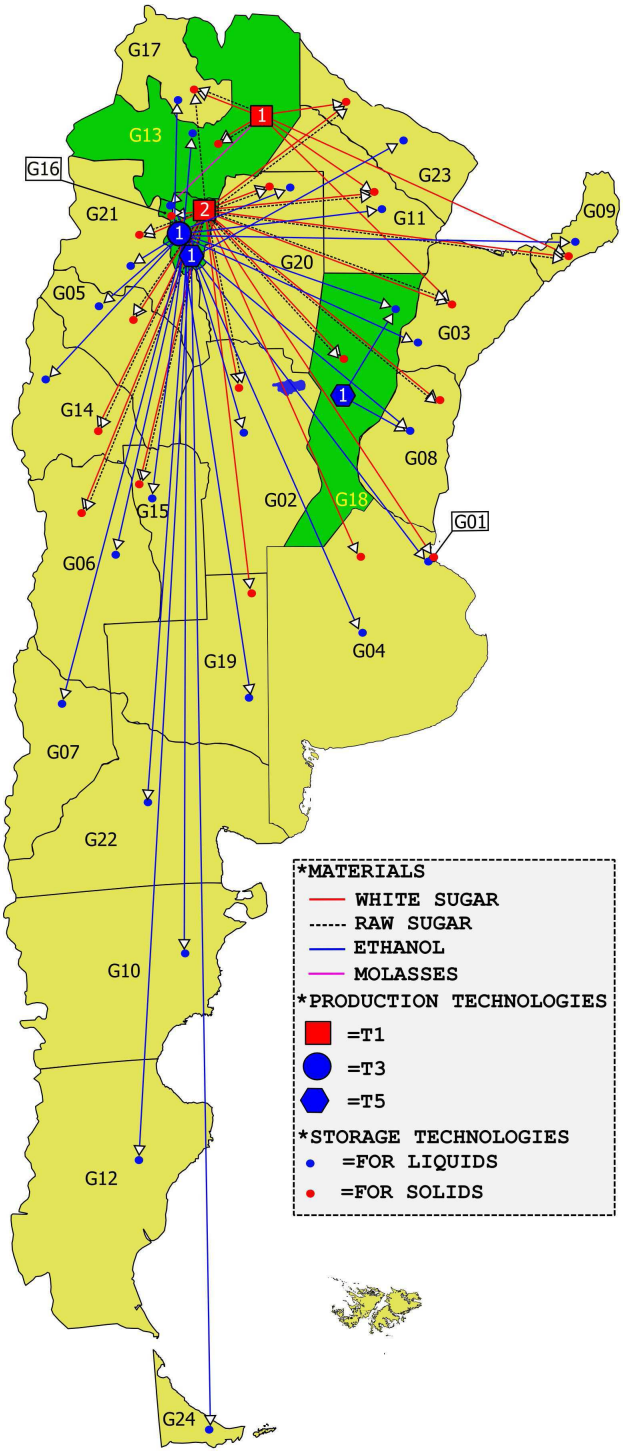


Figure 4.3: Provinces of Argentina.

# Chapter 5

# Numerical Results

In this chapter we describe the numerical results obtained using the algorithms described in the previous chapter, where different instances that differ in the number of time periods were solved. Particularly, we have compared the results obtained by our algorithm with those produced by the standard branch and cut program implemented in CPLEX, nowadays regarded as one the most efficient state of the art deterministic code for MILP.

For each case, we determine the values of parameters $m_{max}$ and $n_{max}$ that lead to the best performance of the algorithm for the case of 8 periods ($t = 8$), considering 10 runs of the algorithm.

## 5.1 Experiments:

In the following subsection we present the numerical results that illustrate the performance of our solution method compared with the commercial full space branch and cut code implemented in CPLEX. Recall that in the HYDROGEN SC problem, the Expected Total Cost (E[TDC]) is minimized, whereas in of the ETHANOL SC, the Net Present Value (NPV) is maximized. We have selected the numbers of periods shown in 5.1.

| $t$ | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 |
|---|---|---|---|---|---|---|---|---|

Table 5.1: Number of periods

All experiments were performed on PC Intel (R) Core (TM) Quad CPU Q9550@2.83 GHz 2.83 GHz 2.98GB RAM.

### 5.1.1 Hydrogen

First, we tune the algorithm, solving the problems for different values of $n$ and $m$. Consider that $n$ is the maximum number of variables released and $m$ is the maximum number of attempts. Figures 5.1 and 5.2 show the results obtained considering 10 runs of the algorithm. This figure is a boxplot, a convenient way of graphically depicting groups of numerical data through their five-number summaries: the smallest observation (sample minimum), lower quartile (Q1), median (Q2), upper quartile (Q3), and largest observation (sample maximum). A boxplot may also indicate which observations, if any, might be considered as outliers. That is, in red we show the median or Q2 quartile of the runs, whereas the blue boxes denote the Q1 and Q3 quartiles. The final values selected are highlighted in red color.

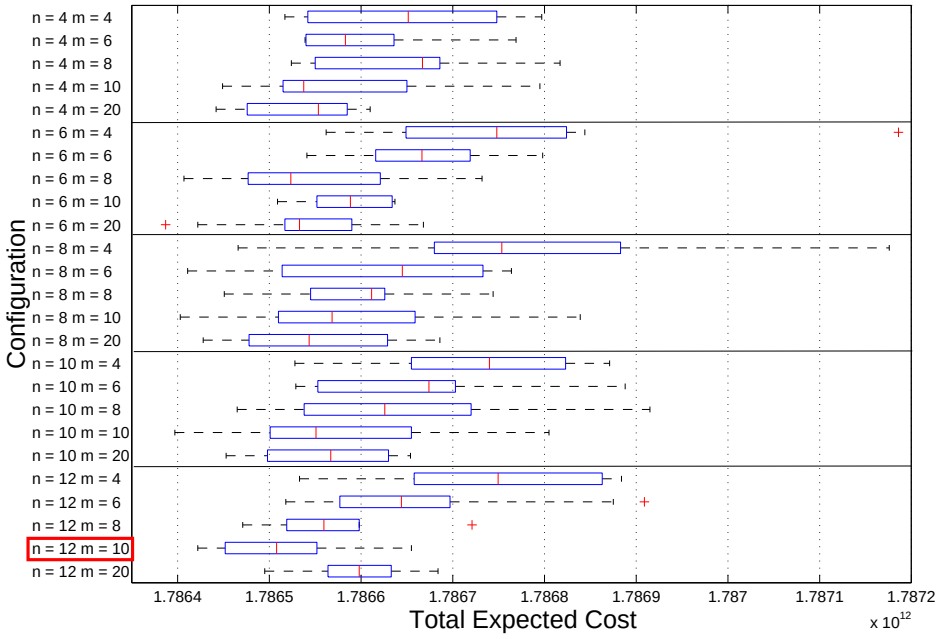


Figure 5.1: Tuning of HYDROGEN results sorted by $n$ and $m$

Once the algorithm is tuned, we compare its performance with the commercial branch and cut code implemented in CPLEX. Figures A.1 to A.8

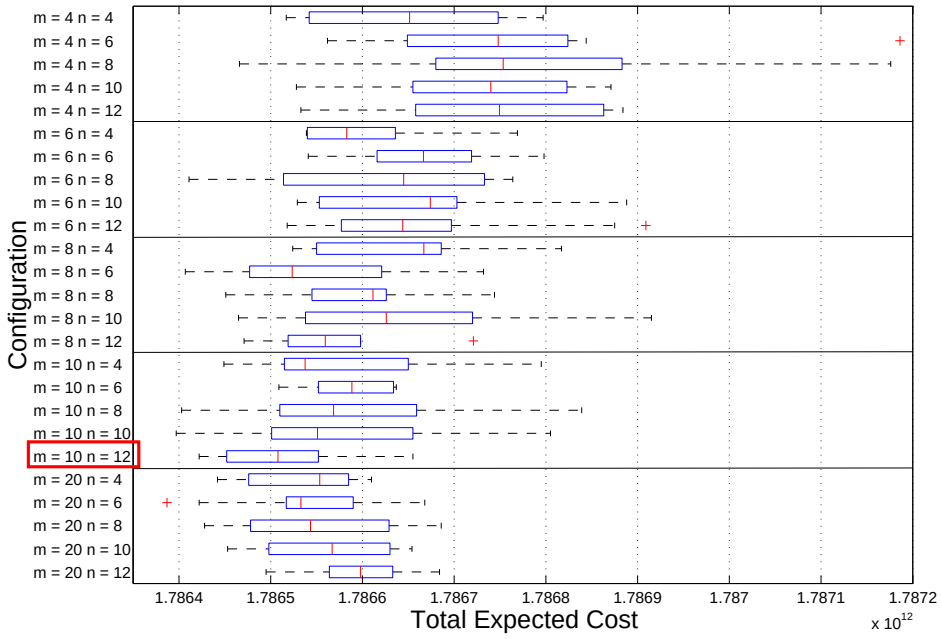


Figure 5.2: Tuning of HYDROGEN results sorted by $m$ and $n$

show for different time periods, the evolution of the lower and upper bounds found by CPLEX as a function of time, along with the values provided by the proposed LNS algorithm as iterations proceed. The time frame considered in the analysis is that for which no further improvement is observed in the LNS. As observed, for low time periods (less than or equal to 6), CPLEX outperforms the proposed algorithm, finding better solutions in shorter CPU times. For more than 6 time periods, CPLEX cannot find any solution, whereas the LNS is always able to provide at least one solution. Note that our algorithm does not show a high variability.

In Table 5.2, we provide for each instance being solved, the time at which the best solution calculated by the LNS has been found, the best objective function value in all the runs, the average objective function value and CPU time, and the standard deviation of the objective function and CPU time. Note that in periods 14 and 16 the standard deviation of the time is zero

because the algorithm convergence is very fast.

| Per | Time | Bst Time | Bst Cost | Avg Cost | Std Cost | Avg Time | Std Time |
|---|---|---|---|---|---|---|---|
| 2 | 1000 | 72.83 | 1050695000000 | 477.97 | 248.27 | 1050707800000 | 18860894.29 |
| 4 | 2000 | 139.12 | 1296608000000 | 748.38 | 581.41 | 1296693500000 | 37997806.95 |
| 6 | 3000 | 273.30 | 1543787000000 | 1156.19 | 784.40 | 1543836900000 | 42019704.37 |
| 8 | 4000 | 294.95 | 1786446000000 | 1651.62 | 1196.73 | 1786516900000 | 58333238.10 |
| 10 | 5000 | 295.90 | 2019579000000 | 2751.97 | 1657.26 | 2019793600000 | 111899955.32 |
| 12 | 6000 | 638.58 | 2239774000000 | 2973.20 | 1508.65 | 2239906300000 | 128670164.03 |
| 14 | 7000 | 58.05 | 2450000000000 | 195.28 | 125.80 | 2450000000000 | 0 |
| 16 | 8000 | 144.57 | 2640000000000 | 227.95 | 116.70 | 2640000000000 | 0 |

Table 5.2: Results of the LNS algorithm to the problem of HYDROGEN

Finally, Table 5.3 displays the optimality GAPs of the following solutions: the best solution calculated by CPLEX after 12 hours of CPU time and after the same CPU time provided to the LNS, the best solution found by the LNS and the average solution calculated by the LNS. The GAP is determined from the best solution calculated by CPLEX in 12 hours. Note that in some instances, CPLEX is unable to provide any bound even the aforementioned CPU time.

| Per | CPLEX 12h | CPLEX Time | LNS Avg | LNS Bst |
|---|---|---|---|---|
| 2 | 0.0495 | 0.0495 | 0.0540 | 0.0528 |
| 4 | 0.0594 | 0.0609 | 0.0681 | 0.0615 |
| 6 | 0.0648 | 0.0674 | 0.0730 | 0.0698 |
| 8 | 0.0834 | No result | No result | No result |
| 10 | 0.0956 | No result | No result | No result |
| 12 | No result | No result | No result | No result |
| 14 | No result | No result | No result | No result |
| 16 | No result | No result | No result | No result |

Table 5.3: HYDROGEN GAP's. GAP's are calculated respect to lower bound found by CPLEX for 12h. No result means that in the given time CPLEX not able to obtain a solution.

### 5.1.2 Ethanol

The information contained in Figures 5.3 and 5.4 are equivalent to Figures 5.1 and 5.2. The Figures A.9 to A.16 are equivalent to Figures A.1 to A.8.

As shown, for low time periods (less than 6), CPLEX outperforms the proposed algorithm, finding better solutions in shorter CPU times. For more than 8 time periods, none of the methods performs better than the other one for the whole range of CPU times. Particularly, LNS tend to behave better for short CPU times, whereas CPLEX provides better solutions when long CPU times are considered. The Table 5.4 is equivalent to Table 5.2.

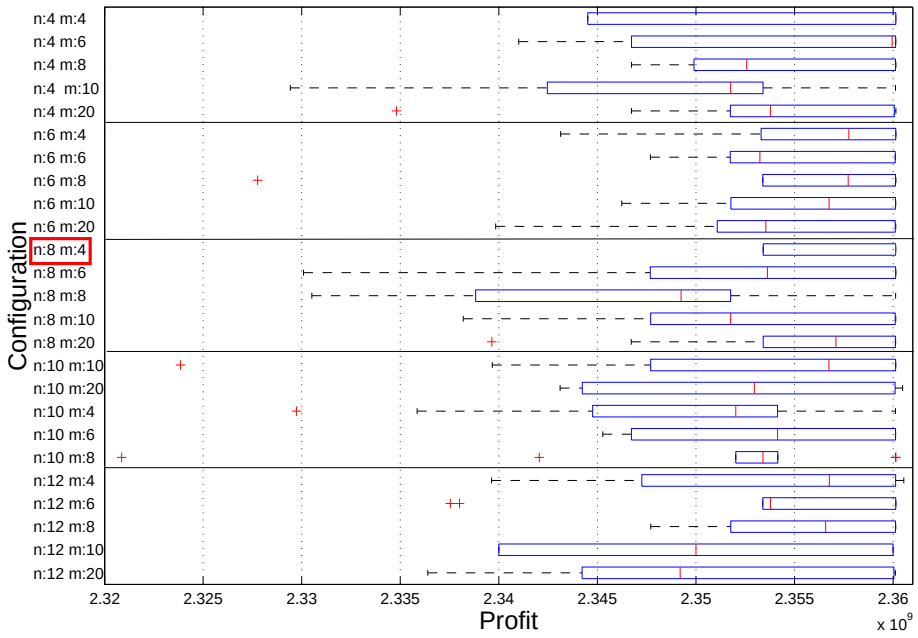


Figure 5.3: Tuning of ETHANOL results sorted by $n$ and $m$

Finally, Table 5.5 is equivalent to Table 5.3. As observed, LNS provides better solutions (i.e., with better optimality gaps) in periods 12,14 and 16.

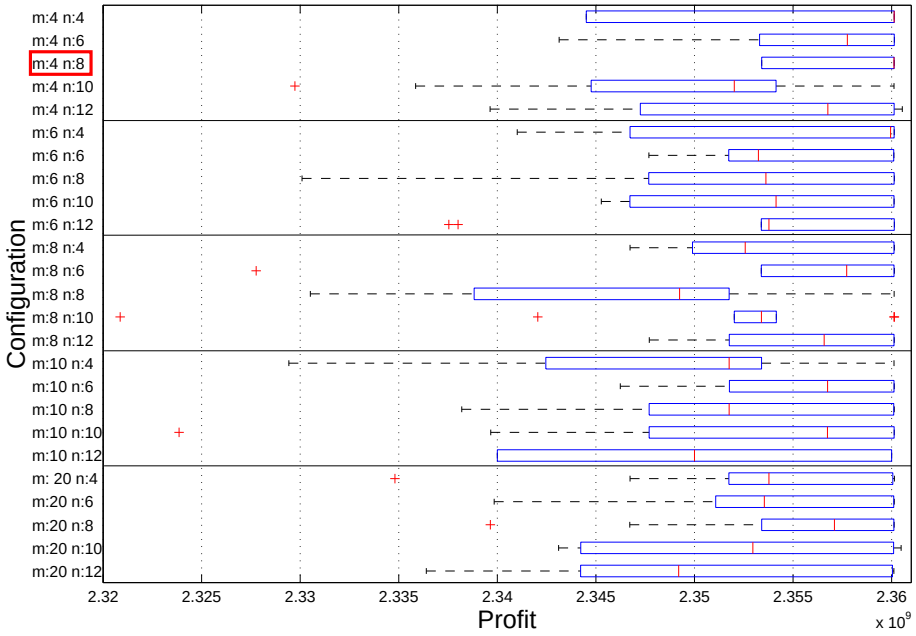


Figure 5.4: Tuning of ETHANOL results sorted by $m$ and $n$

| Per | Time | Bst Time | Bst Profit | Avg Profit | Std Profit | Avg Time | Std Time |
|---|---|---|---|---|---|---|---|
| 2 | 500.00 | 151.36 | 347359830 | 345279290.8 | 2685964.558 | 248.204 | 118.2521557 |
| 4 | 2000.00 | 406.64 | 1080116900 | 1080116900 | 0 | 687.211 | 303.2816757 |
| 6 | 3500.00 | 1304.48 | 1780058700 | 1778863300 | 1764525.812 | 1564.232 | 765.0026542 |
| 8 | 5000.00 | 4654.22 | 2360142100 | 2359543790 | 1891389.92 | 3905.375 | 1049.625741 |
| 10 | 6500.00 | 1705.22 | 2770025600 | 2767586790 | 5477665.745 | 5208.404 | 1460.014114 |
| 12 | 8000.000 | 6862.200 | 3182514600 | 3182511122.222 | 2329.580 | 7147.720 | 974.898 |
| 14 | 9500.000 | 9530.280 | 3538394900 | 3535963255.556 | 5111604.059 | 8450.841 | 1312.555 |
| 16 | 11000.000 | 9527.750 | 3812615700 | 3806304877.778 | 7596078.119 | 10104.947 | 892.877 |

Table 5.4: LNS algorithm results with the problem of ETHANOL

| Per | CPLEX 12h | CPLEX Time | LNS Avg | LNS Bst |
|---|---|---|---|---|
| 2 | 0.00 | 0.00 | 5.67 | 5.04 |
| 4 | 0.00 | 0.00 | 2.13 | 2.13 |
| 6 | 0.35 | 0.35 | 1.55 | 1.48 |
| 8 | 0.83 | 1.08 | 1.82 | 1.80 |
| 10 | 1.38 | 1.60 | 2.30 | 2.21 |
| 12 | 2.09 | 3.43 | 2.74 | 2.74 |
| 14 | 1.92 | 3.10 | 2.80 | 2.73 |
| 16 | 2.06 | 3.32 | 2.87 | 2.70 |

Table 5.5: ETHANOL GAP's. GAP's are calculated respect to upper bound found by CPLEX for 12h

# Chapter 6

# Conclusions and future work

This work has addressed the optimal design and planning of Hydrogen/Ethanol Scs. The design task was formulated as a mixed-integer programming problem that seeks to minimize the expected total cost in the case of HYDROGEN and maximize the net present value in the ETHANOL case. To overcome the large computational burden of solving these MILPs, we proposed a hybrid metaheuristic that combine the large neighborhood search with CPLEX. The capabilities of the proposed mathematical model and solver methodology were shown through two cases studies based on the Argentina sugar cane industry and the Hydrogen supply chain in Spain.

From the computational point of view, the hybrid metaheuristic algorithm provides near optimal solutions in a fraction of the time spent by CPLEX; even when CPLEX did not find a solution, our algorithm obtained a solution. In particular, the proposed algorithm is more promising for large instances.

Future work will follow different pathways: on the one hand to exploit the mathematical formulation of the cases to take profit of particular property; on the other, to initialize CPLEX with the solution obtained from our algorithm.

# Appendix A

# Appendix

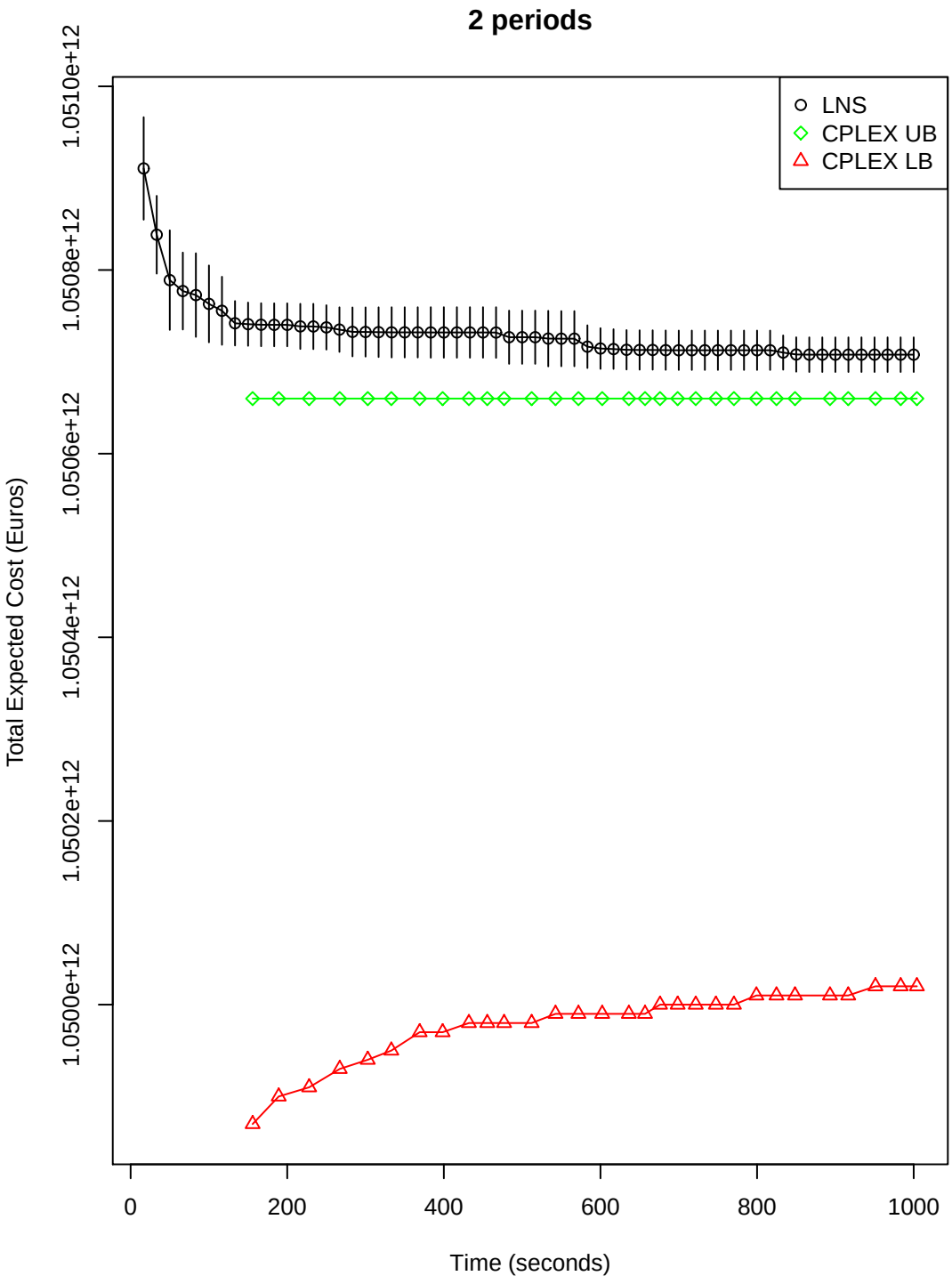


Figure A.1: LNS compared with CPLEX on the problem of HYDROGEN for $t = 2$. The vertical bars show the standard deviation of LNS over 10 runs.

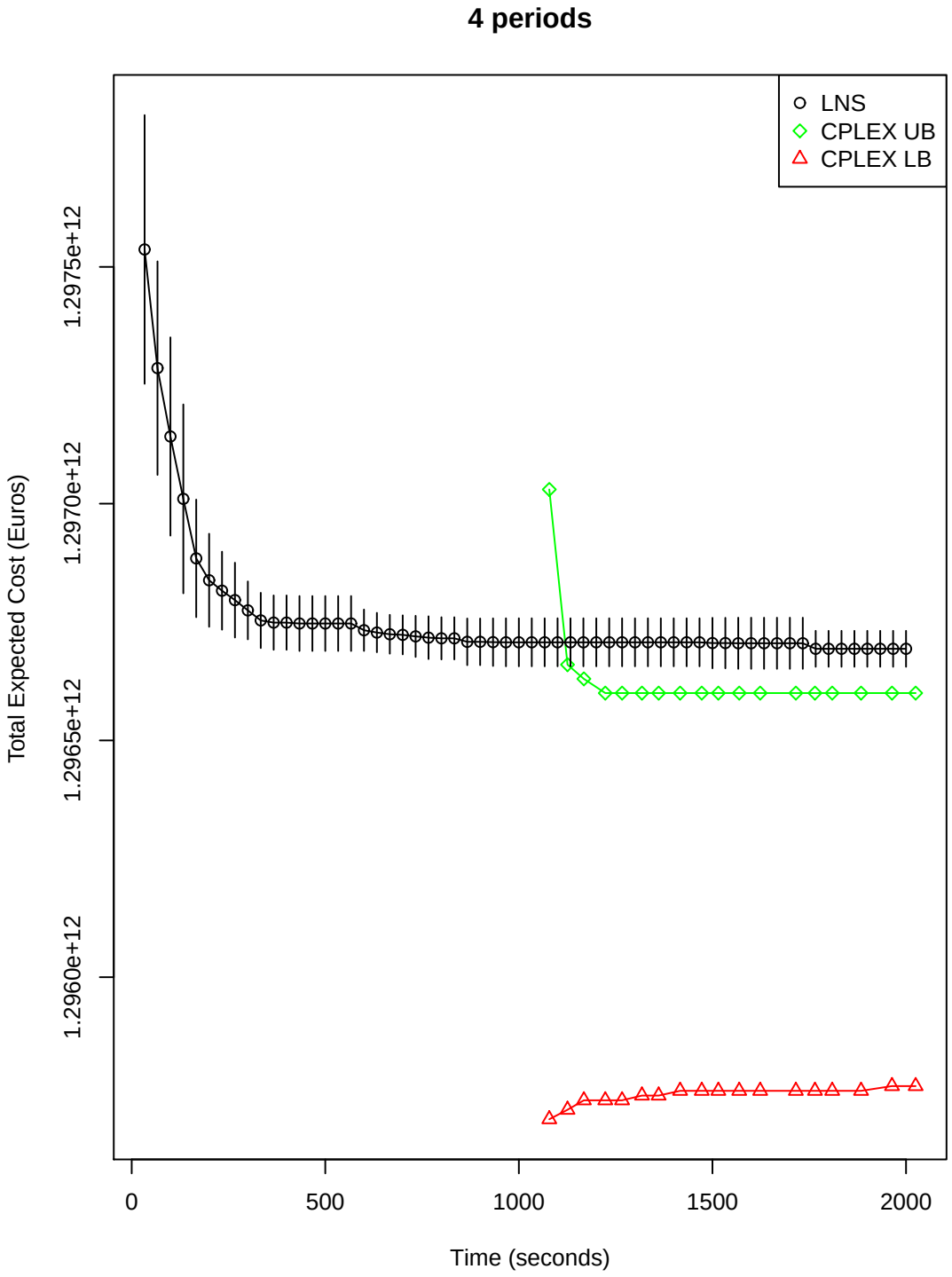


Figure A.2: LNS compared with CPLEX on the problem of HYDROGEN for $t = 4$. The vertical bars show the standard deviation of LNS over 10 runs.

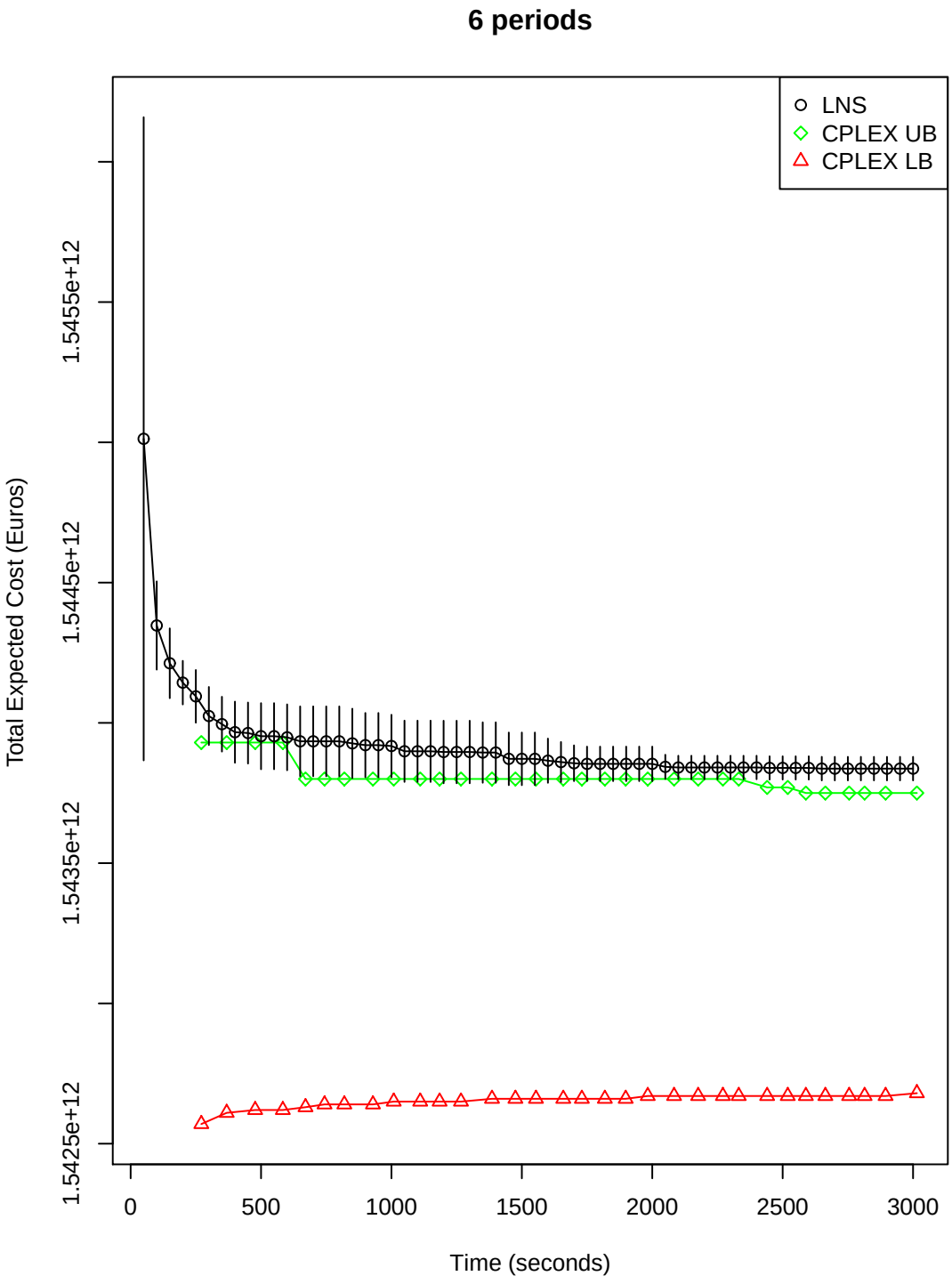


Figure A.3: LNS compared with CPLEX on the problem of HYDROGEN for $t = 6$. The vertical bars show the standard deviation of LNS over 10 runs.

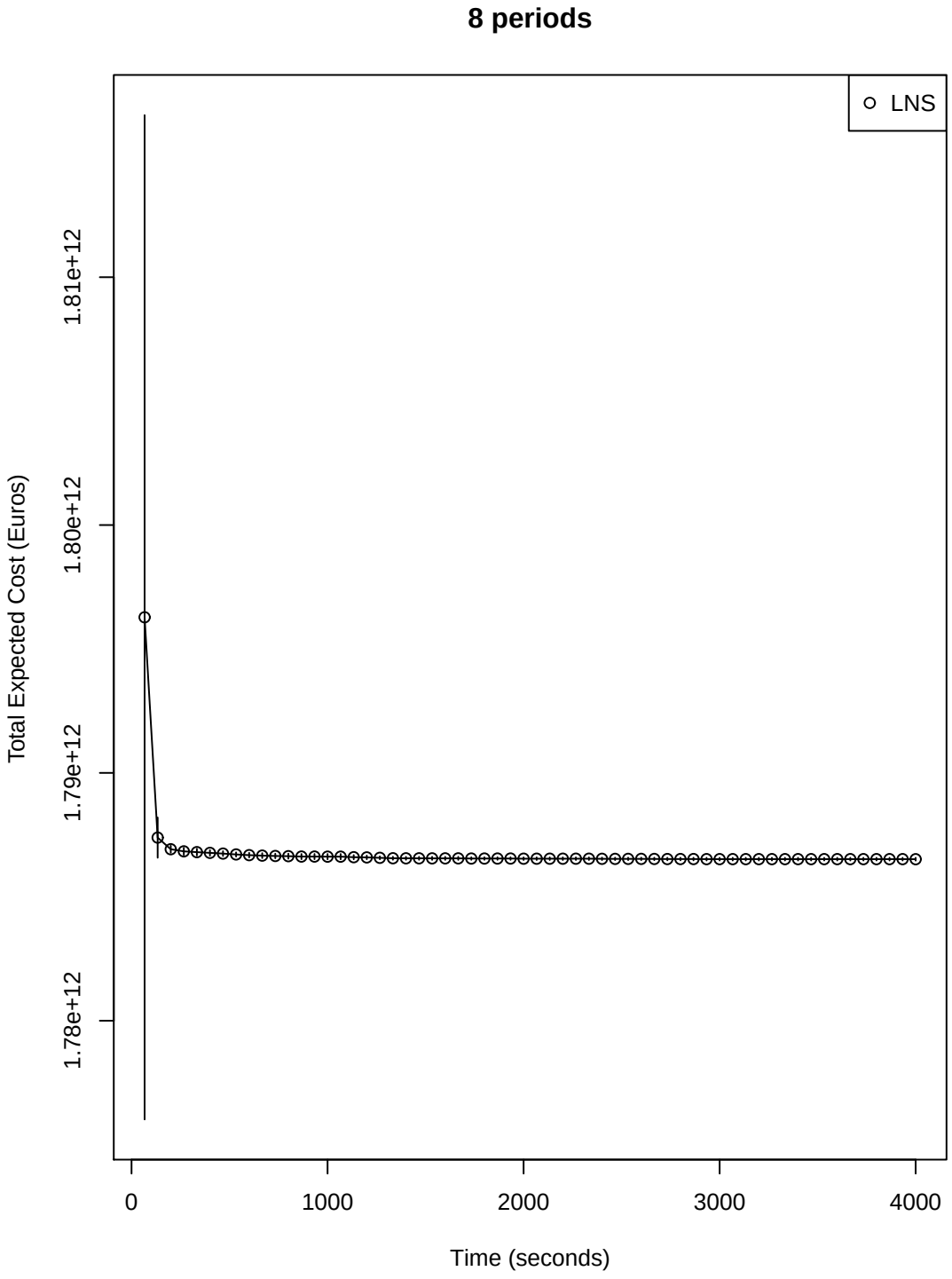


Figure A.4: LNS compared with CPLEX on the problem of HYDROGEN for $t = 8$. The vertical bars show the standard deviation of LNS over 10 runs. In the time given CPLEX was not able to obtain a solution.

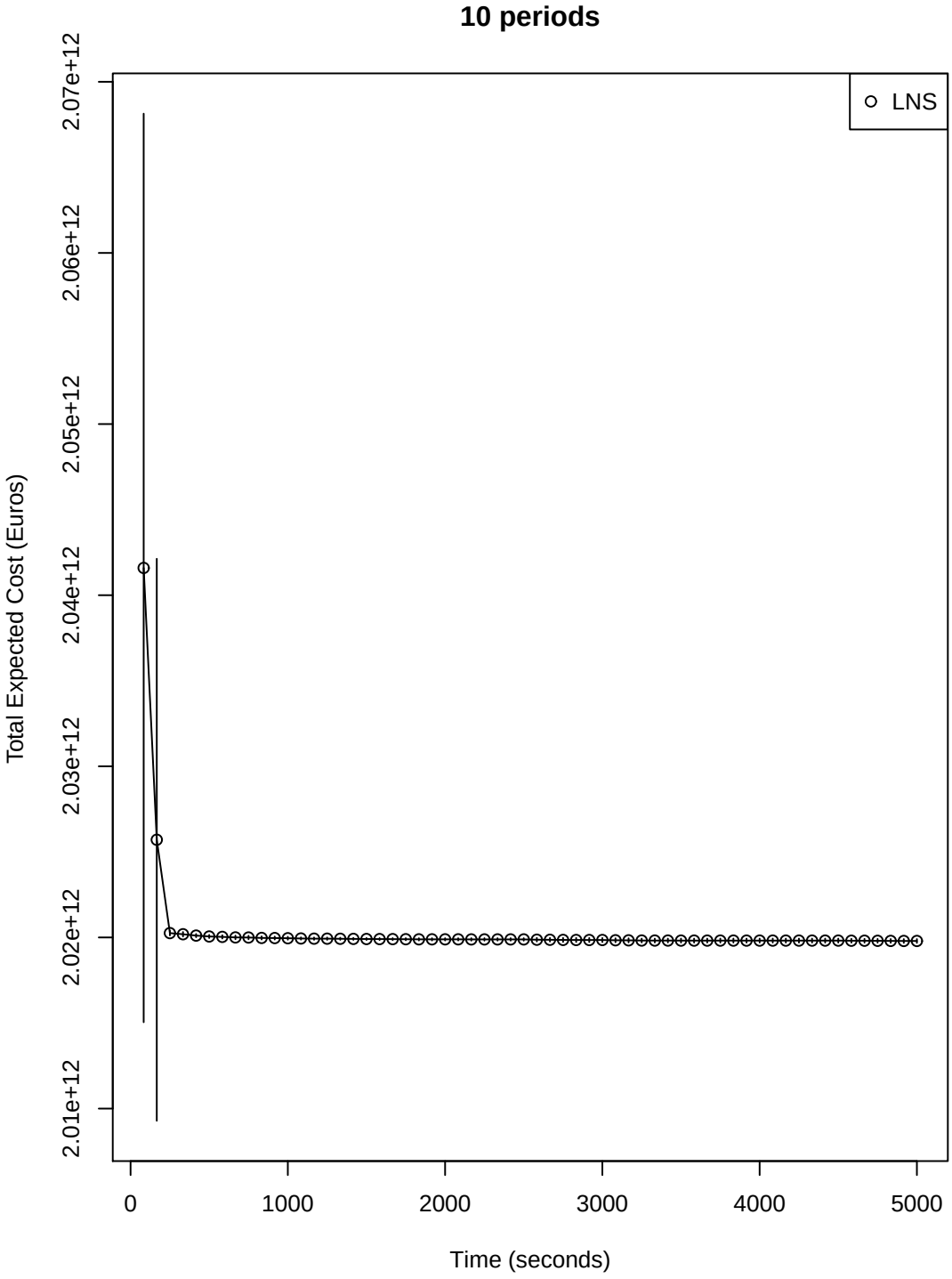


Figure A.5: LNS compared with CPLEX on the problem of HYDROGEN for $t = 10$. The vertical bars show the standard deviation of LNS over 10 runs. In the time given CPLEX was not able to obtain a solution.

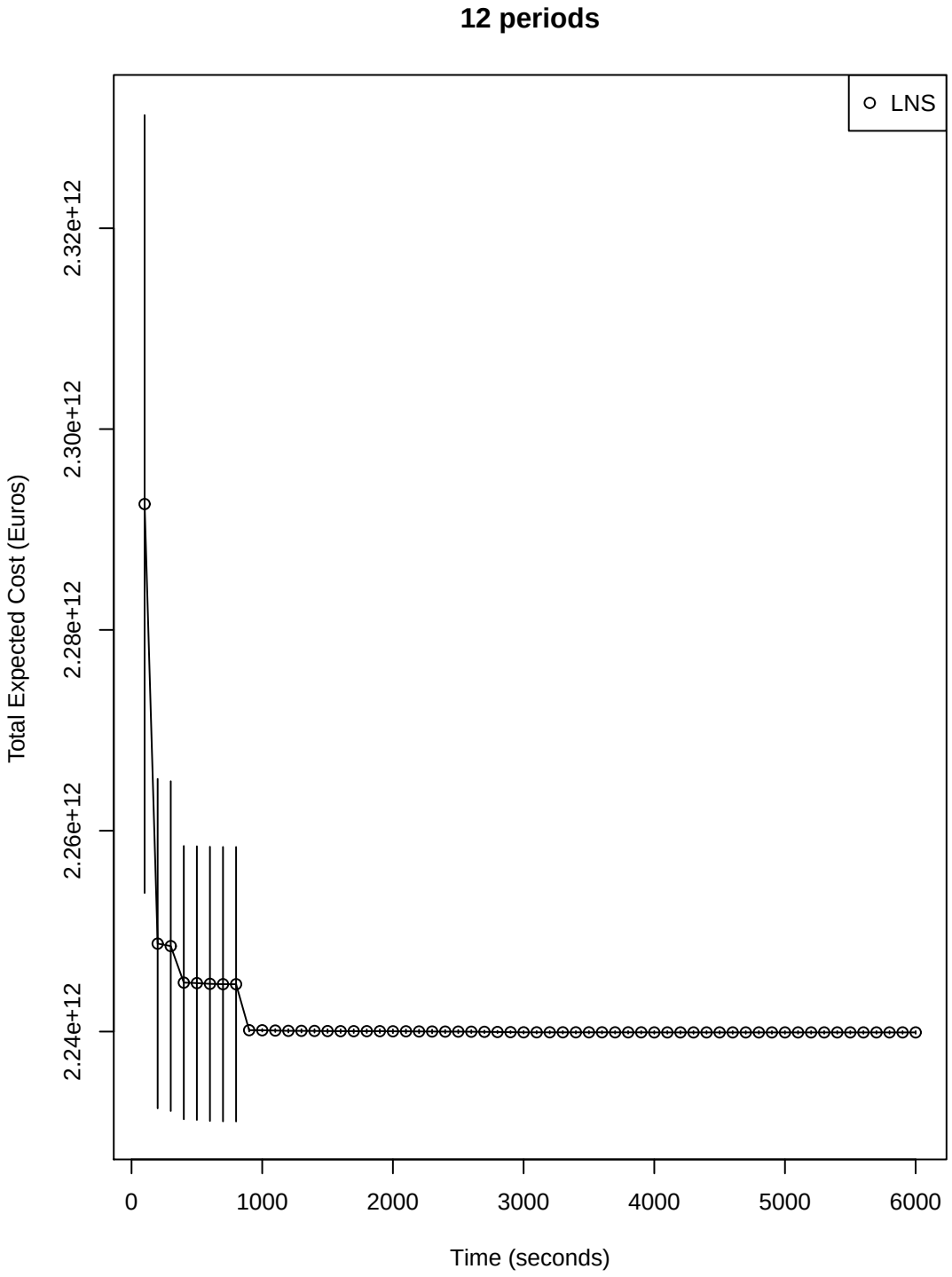


Figure A.6: LNS compared with CPLEX on the problem of HYDROGEN for $t = 12$. The vertical bars show the standard deviation of LNS over 10 runs. In the time given CPLEX was not able to obtain a solution.

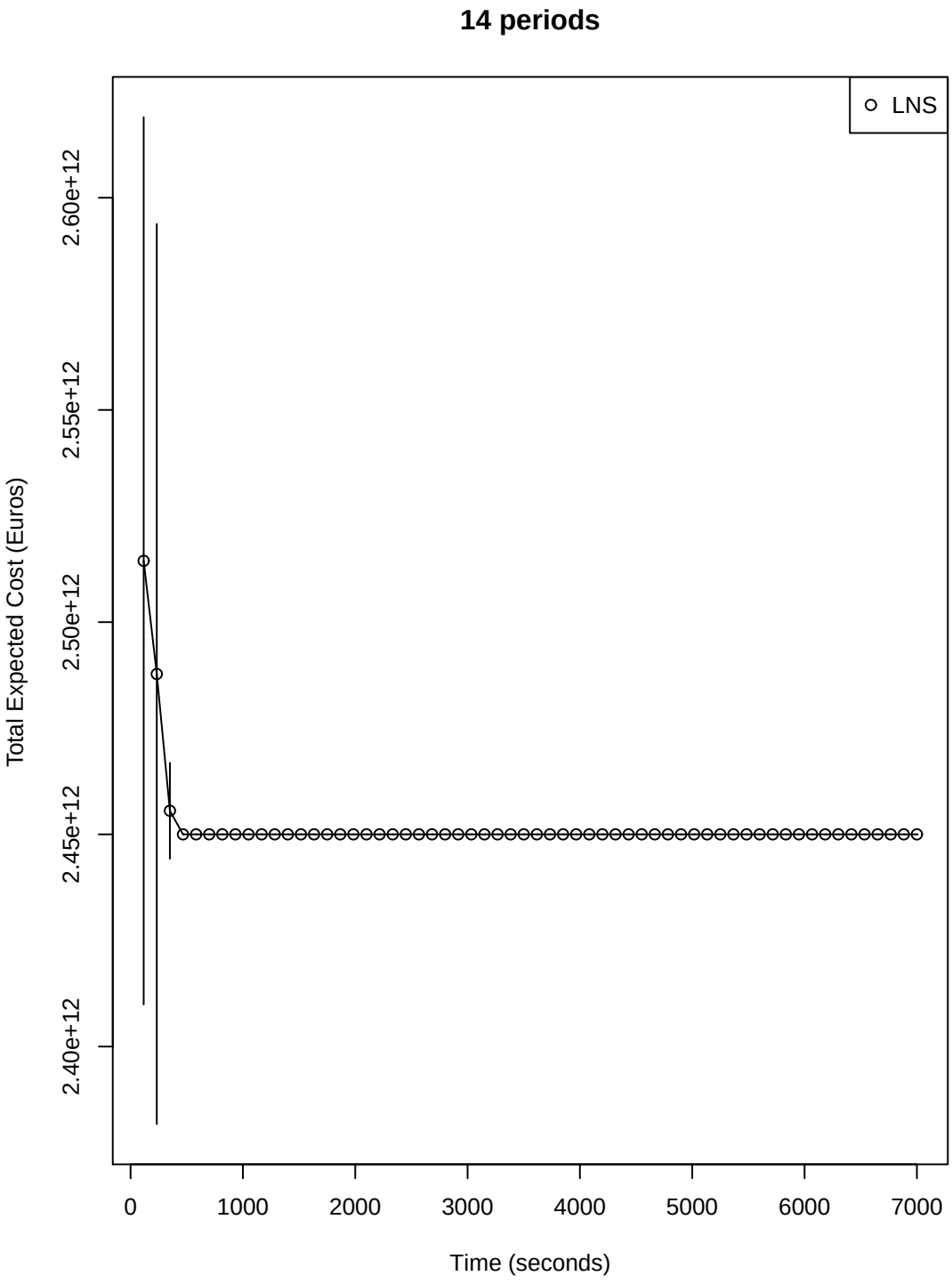


Figure A.7: LNS compared with CPLEX on the problem of HYDROGEN for $t = 14$. The vertical bars show the standard deviation of LNS over 10 runs. In the time given CPLEX was not able to obtain a solution.

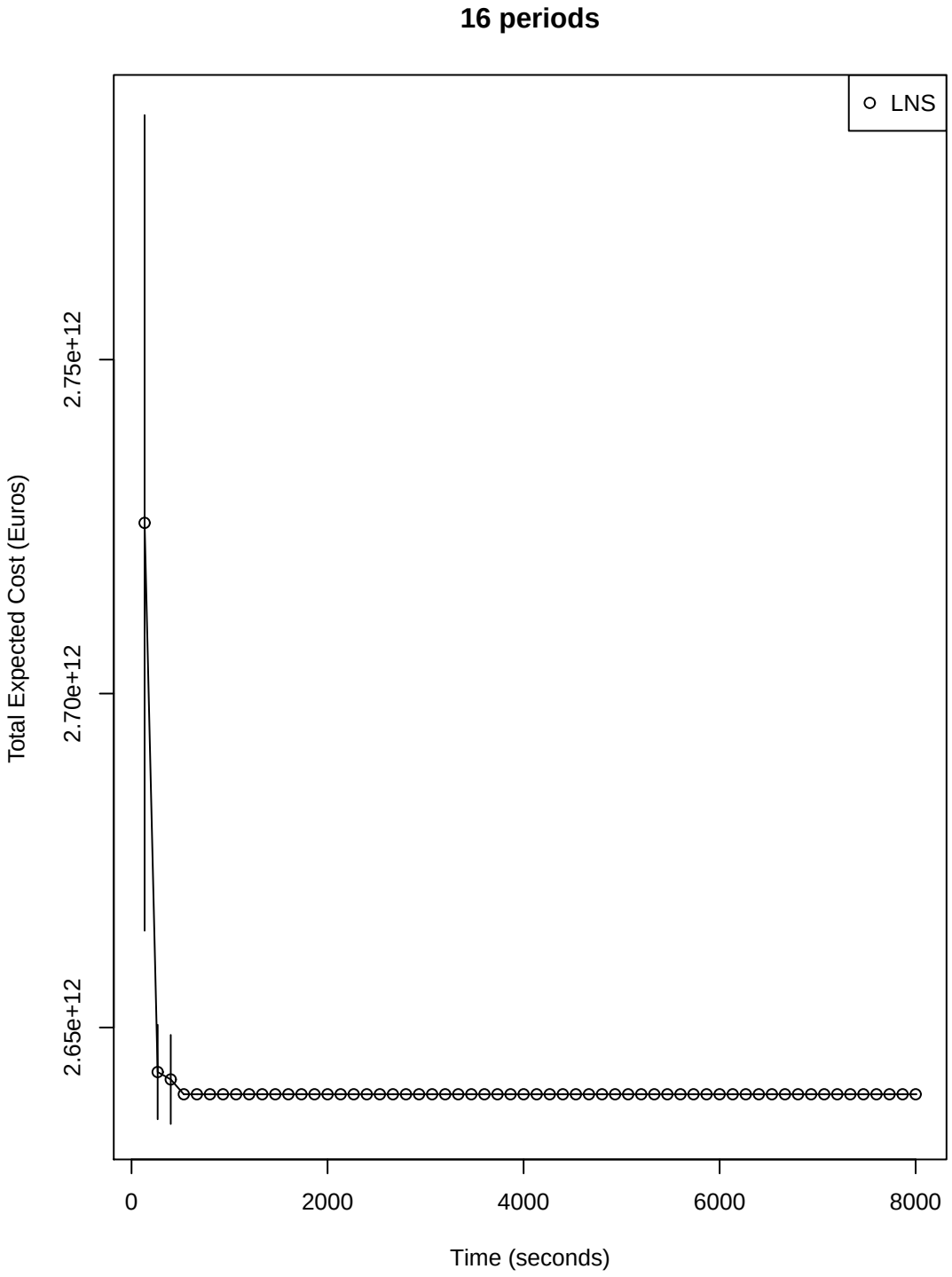


Figure A.8: LNS compared with CPLEX on the problem of HYDROGEN for $t = 16$. The vertical bars show the standard deviation of LNS over 10 runs. In the time given CPLEX was not able to obtain a solution.

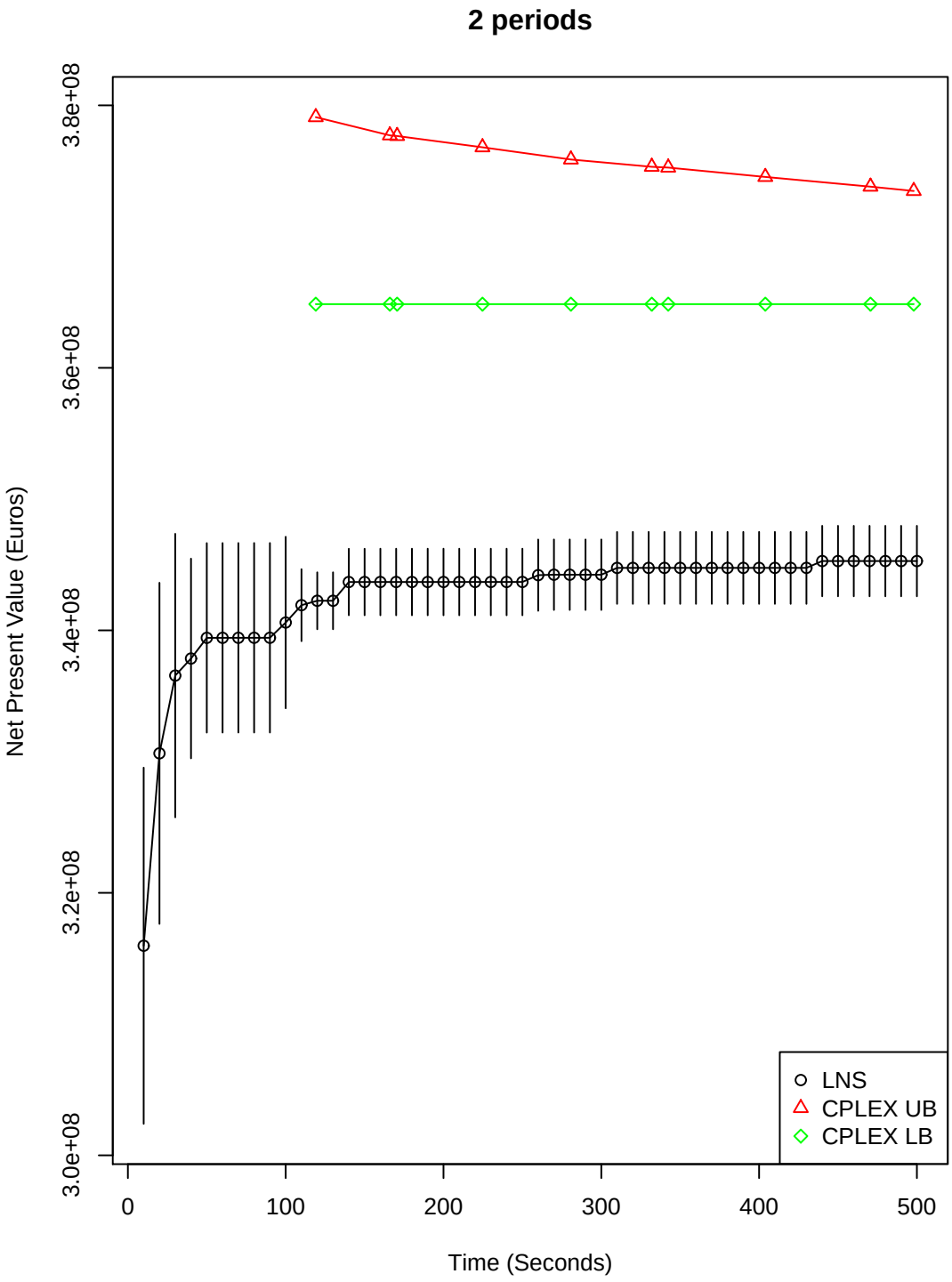


Figure A.9: LNS compared with CPLEX on the problem of ETHANOL for $t = 2$. The vertical bars show the standard deviation of LNS over 10 runs.

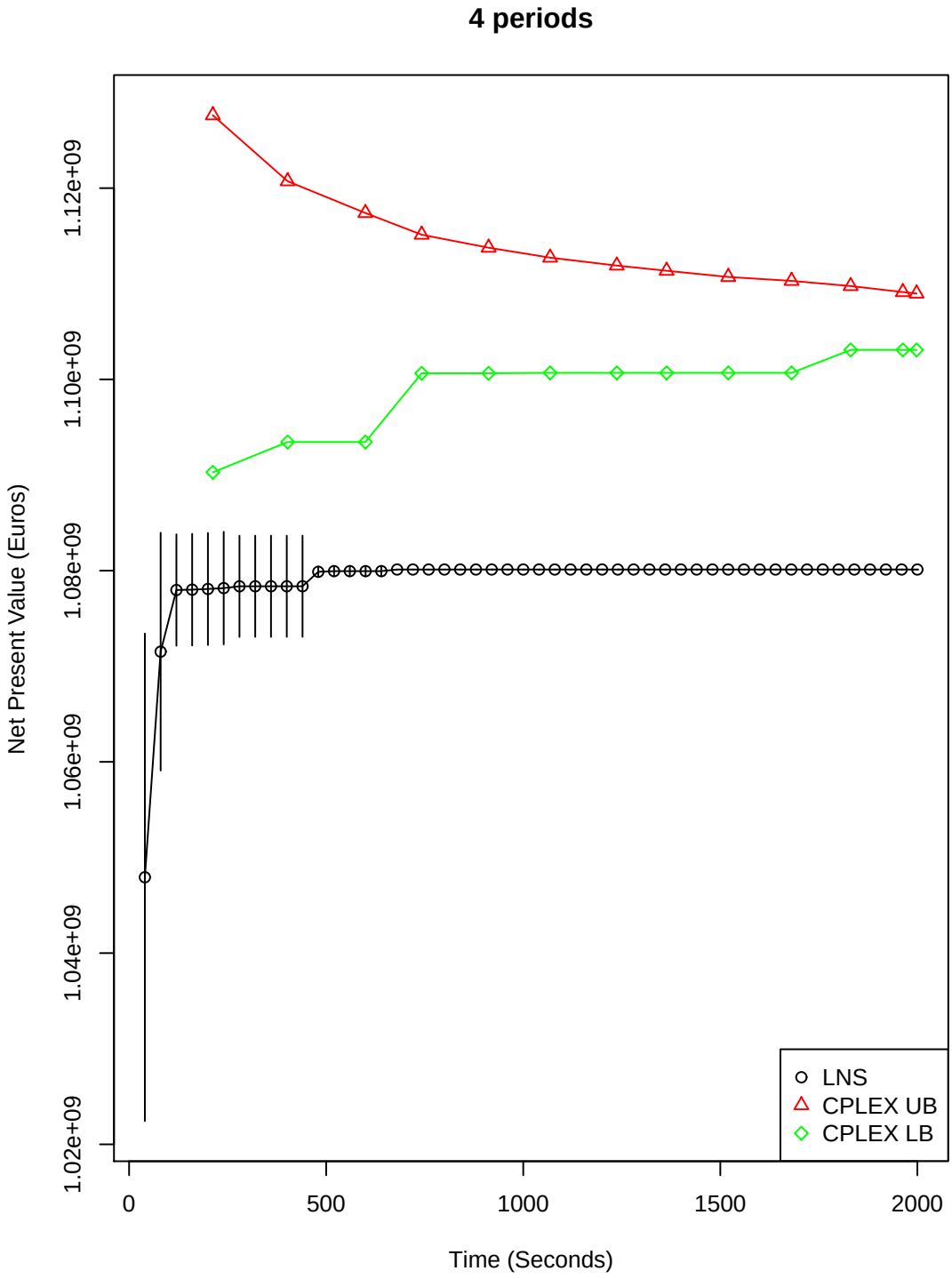


Figure A.10: LNS compared with CPLEX on the problem of ETHANOL for $t = 4$. The vertical bars show the standard deviation of LNS over 10 runs.

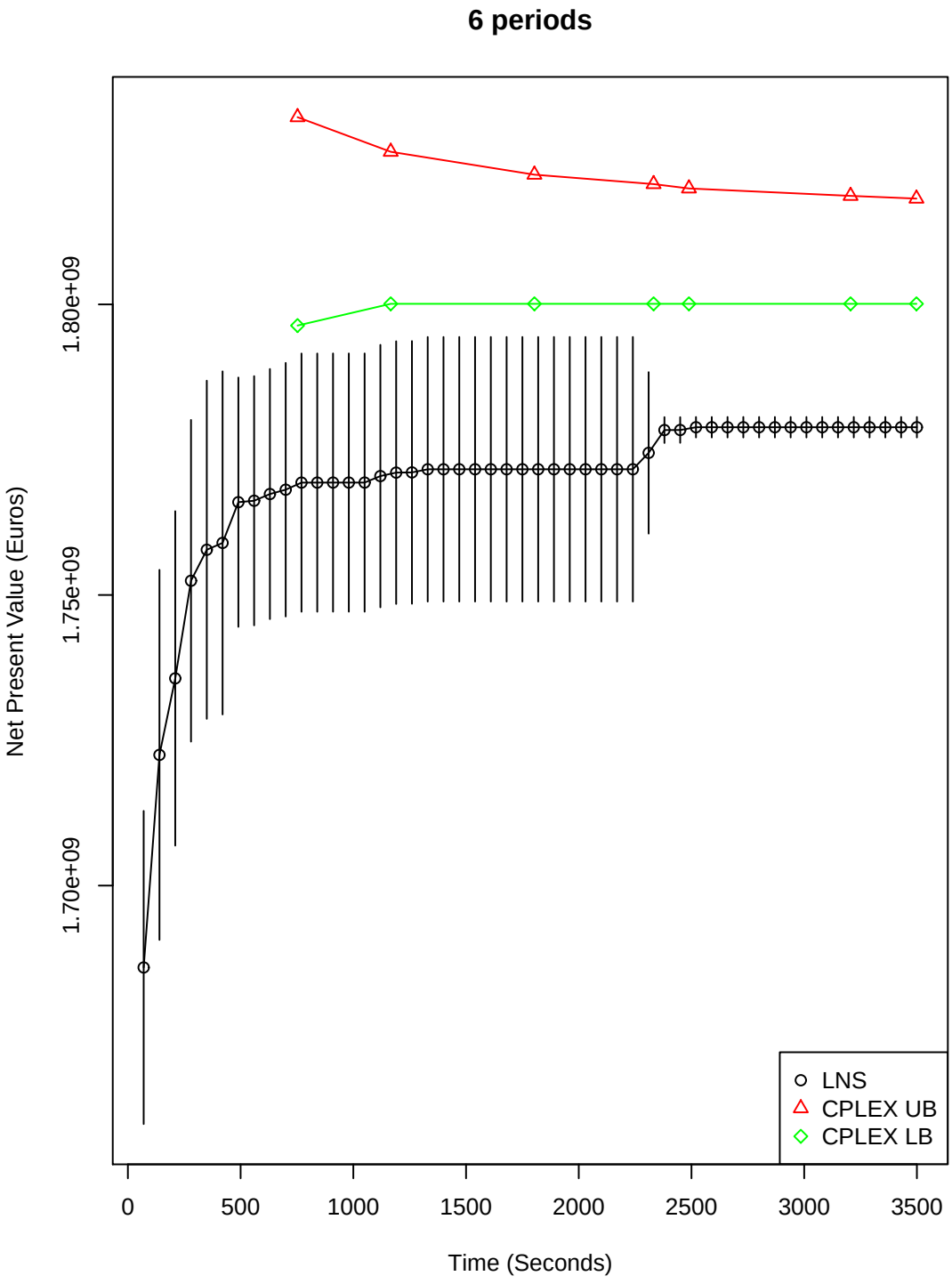


Figure A.11: LNS compared with CPLEX on the problem of ETHANOL for $t = 6$. The vertical bars show the standard deviation of LNS over 10 runs.

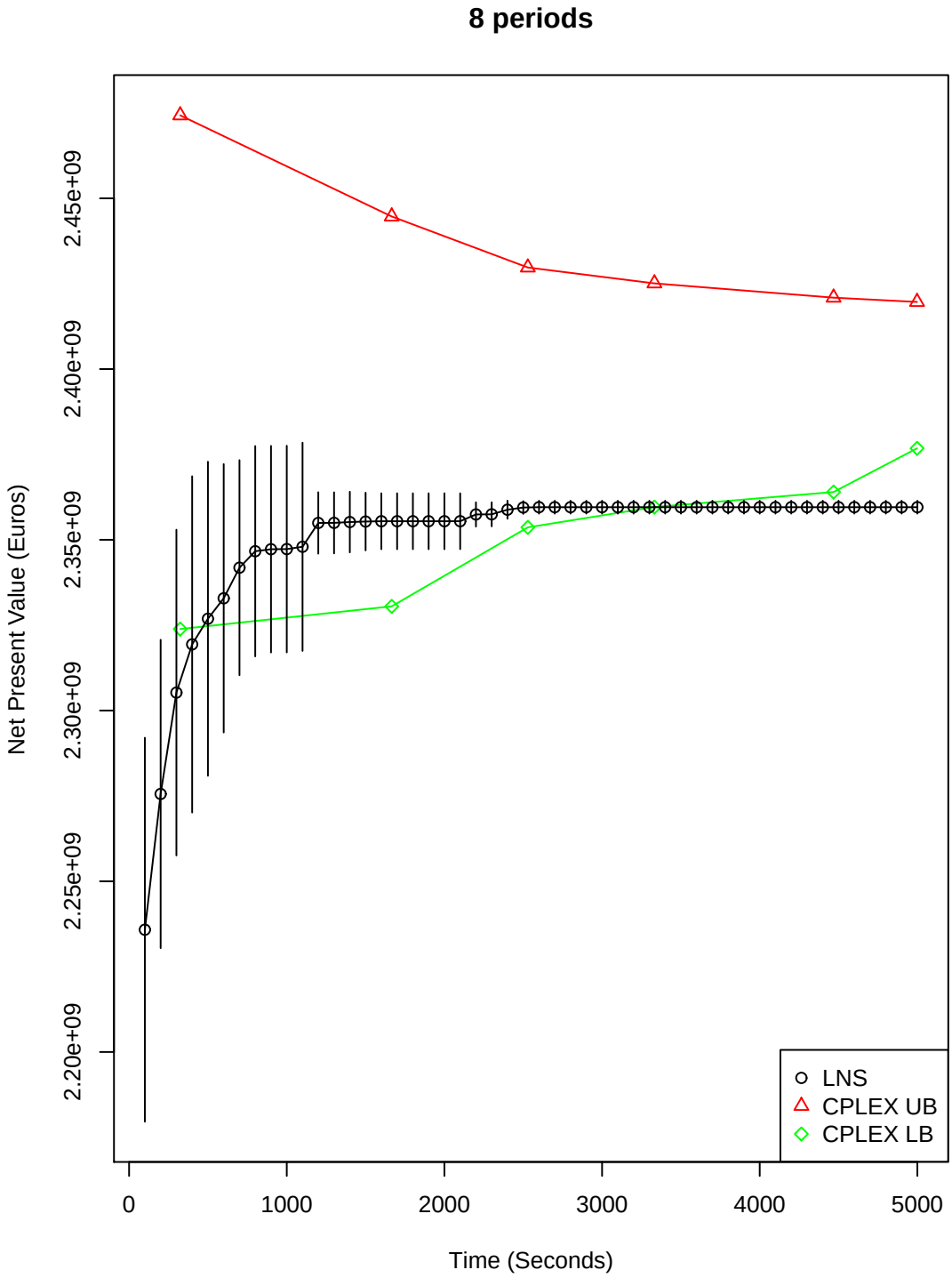


Figure A.12: LNS compared with CPLEX on the problem of ETHANOL for $t = 8$. The vertical bars show the standard deviation of LNS over 10 runs.

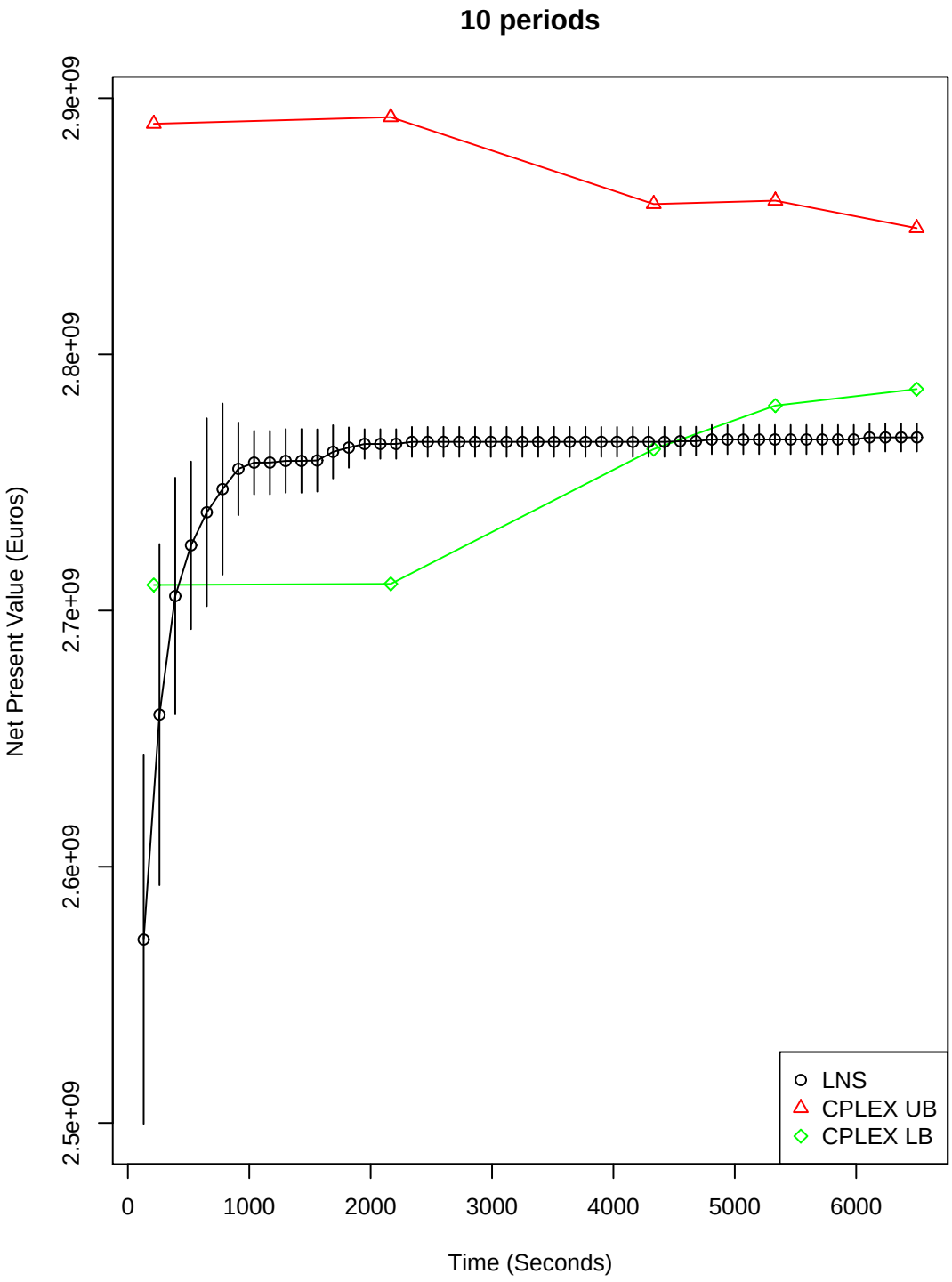


Figure A.13: LNS compared with CPLEX on the problem of ETHANOL for $t = 10$. The vertical bars show the standard deviation of LNS over 10 runs.

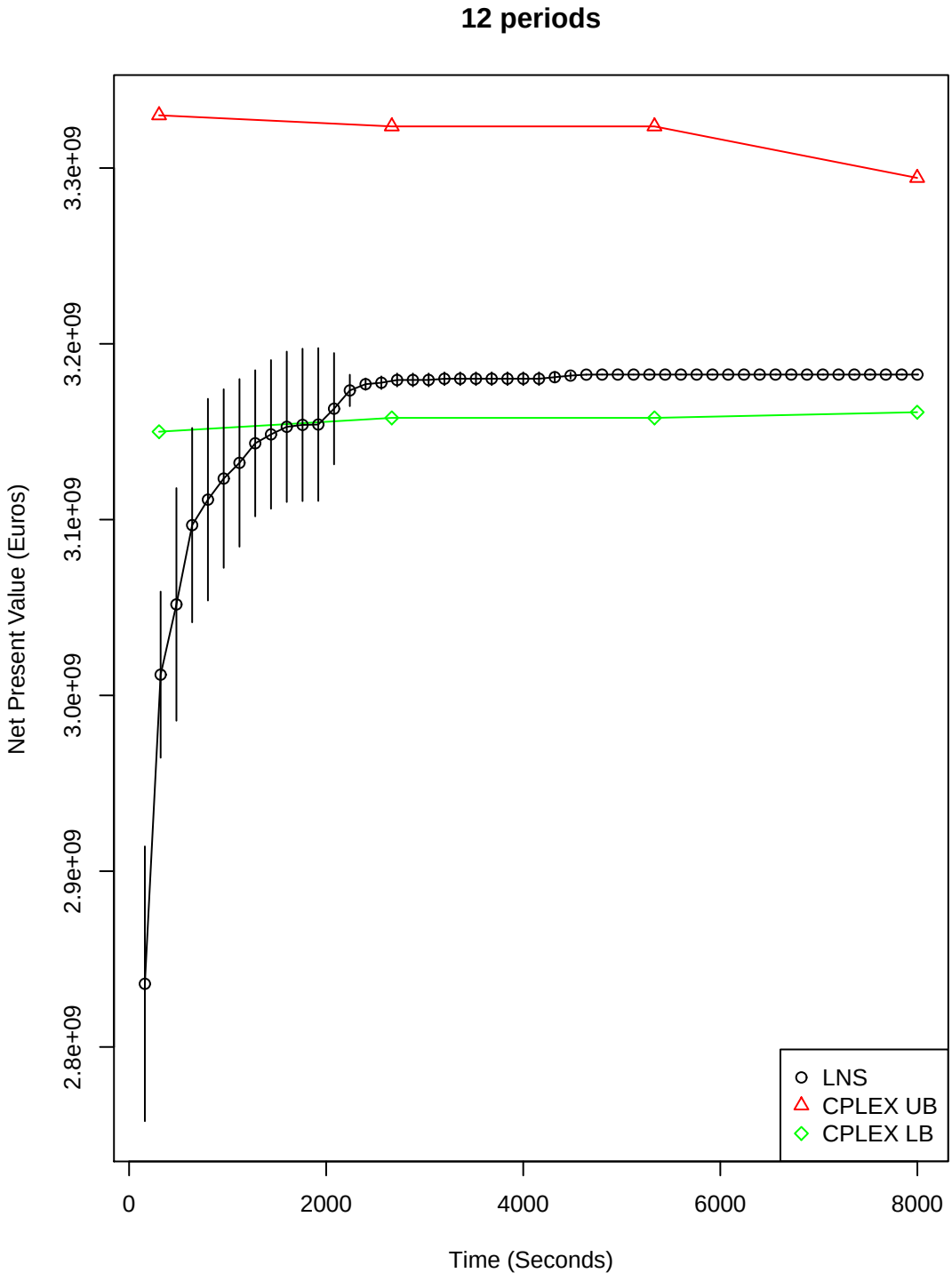


Figure A.14: LNS compared with CPLEX on the problem of ETHANOL for $t = 12$. The vertical bars show the standard deviation of LNS over 10 runs.

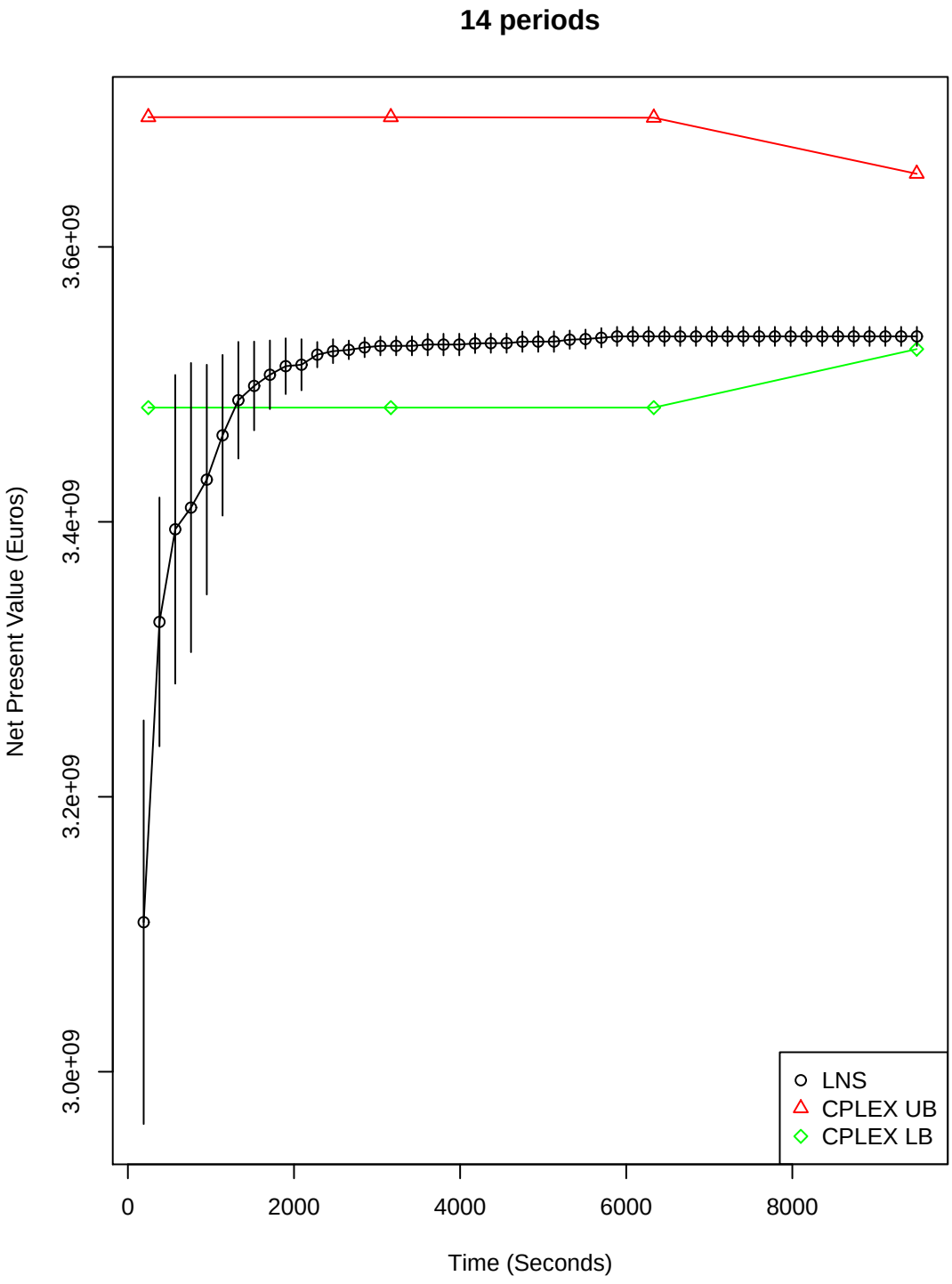


Figure A.15: LNS compared with CPLEX on the problem of ETHANOL for $t = 14$. The vertical bars show the standard deviation of LNS over 10 runs.

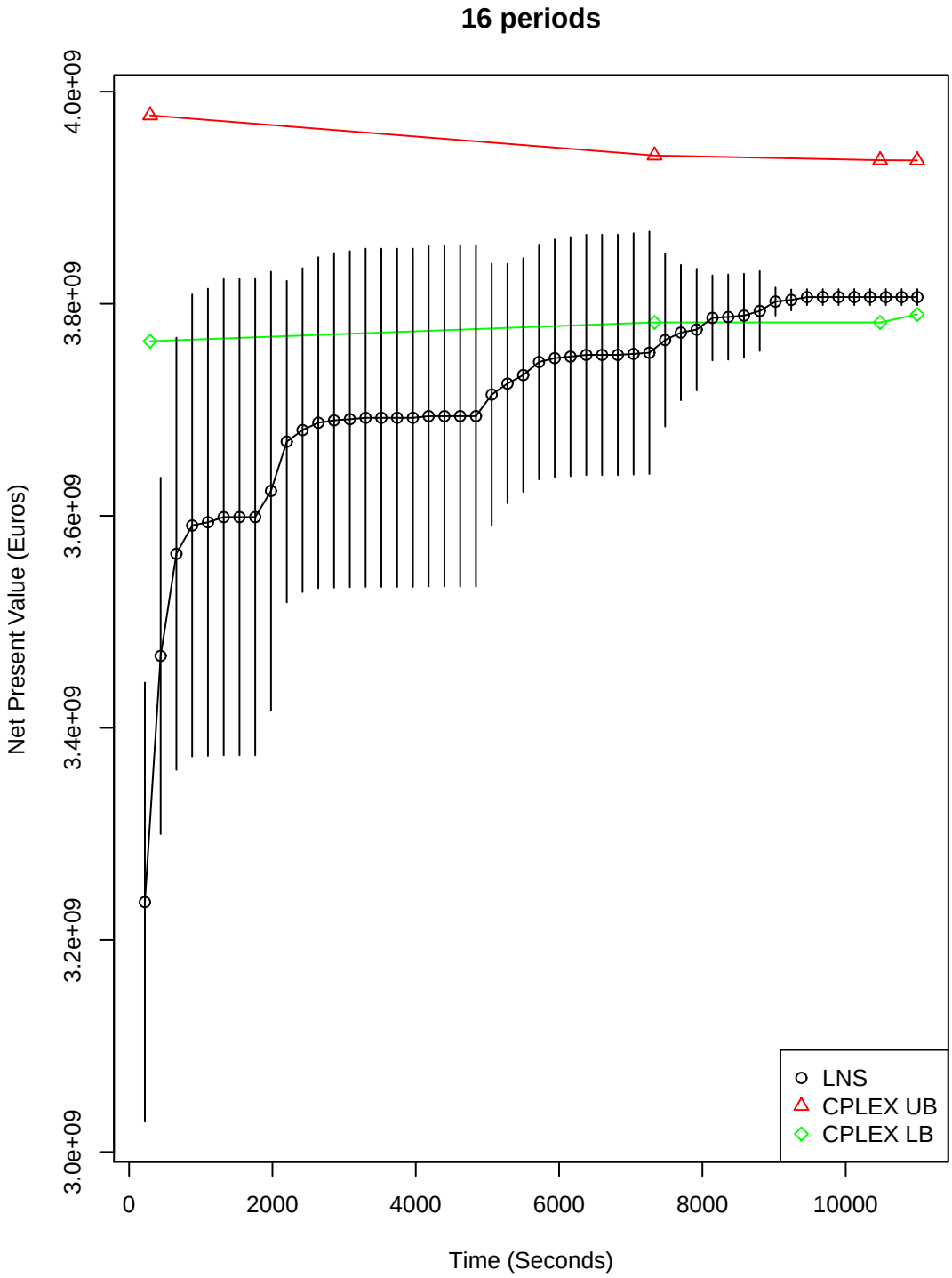


Figure A.16: LNS compared with CPLEX on the problem of ETHANOL for $t = 16$. The vertical bars show the standard deviation of LNS over 10 runs.

# Bibliography

Ahuja, R., Ergun,
"O., Orlin, J., Punnen, A., 2002. A survey of very large-scale neighborhood search techniques. Discrete Applied Mathematics 123 (1-3), 75–102.

Alba, E. (Ed.), 2005. Parallel Metaheuristics: A New Class of Algorithms. John Wiley.

Almansoori, A., Shah, N., 2006. Design and Operation of a Future Hydrogen Supply Chain:: Snapshot Model. Chemical Engineering Research and Design 84 (6), 423–438.

Bagajewicz, M., 2008. On the use of net present value in investment capacity planning models. Industrial & Engineering Chemistry Research 47 (23), 9413–9416.

Bazaraa, M., Sherali, H., Shetty, C., 2006. Nonlinear programming: theory and algorithms. John Wiley and Sons.

Blum, C., 2004. Theoretical and practical aspects of ant colony optimization. IOS Press.

Blum, C., 2005a. Ant colony optimization: Introduction and recent trends. Physics of Life reviews 2 (4), 353–373.

Blum, C., 2005b. Beam-ACO—Hybridizing ant colony optimization with beam search: An application to open shop scheduling. Computers & Operations Research 32 (6), 1565–1591.

Blum, C., Roli, A., 2003. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computing Surveys (CSUR) 35 (3), 268–308.

Bonfill, A., Bagajewicz, M., Espu na, A., Puigjaner, L., 2004. Risk management in the scheduling of batch plants under uncertain market demand. Ind. Eng. Chem. Res 43 (3), 741–750.

Dantzig, G., 1963. Linear programming and extensions. Princeton University, Princeton, NJ.

Della Croce, F., T'kindt, V., 2002. A Recovering Beam Search algorithm for the one machine dynamic total completion time scheduling problem. Journal of the Operational Research Society 53 (11), 1275–1280.

Dorigo, M., Blum, C., 2005. Ant colony optimization theory: A survey. Theoretical computer science 344 (2-3), 243–278.

Duran, M., Grossmann, I., 1986. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. Mathematical Programming 36 (3), 307–339.

Ehrgott, M., 2005. Multicriteria optimization.

Feo, T., Resende, M., 1995. Greedy randomized adaptive search procedures. Journal of Global Optimization 6 (2), 109–133.

Focacci, F., Laburthe, F., Lodi, A., 2002. Local Search and Constraint Programming. In: Glover, F., Kochenberger, G. (Eds.), Handbook of Metaheuristics. Vol. 57 of International Series in Operations Research & Management Science. Kluwer Academic Publishers, Norwell, MA.

Fogel, L., Owens, A., Walsh, M., et al., 1966. Artificial intelligence through simulated evolution 26.

Geoffrion, A., 1972. Generalized benders decomposition. Journal of optimization theory and applications 10 (4), 237–260.

Glover, F., 1986a. Future paths for integer programming and links to artificial intelligence. Computers & Operations Research 13 (5), 533 – 549.

Glover, F., 1986b. Future paths for integer programming and links to artificial intelligence. Computers & Operations Research 13 (5), 533–549.

Guillén-Gosálbez, G., Grossmann, I., 2009. Optimal design and planning of sustainable chemical supply chains under uncertainty. AIChE Journal 55 (1), 99–121.

Guillén-Gosálbez, G., Grossmann, I., 2010. A global optimization strategy for the environmentally conscious design of chemical supply chains under uncertainty in the damage assessment model. Computers & Chemical Engineering 34 (1), 42–58.

Guillén-Gosálbez, G., Mele, F., Grossmann, I., 2010. A bi-criterion optimization approach for the design and planning of hydrogen supply chains for vehicle use. AIChE Journal 56 (3), 650–667.

Gupta, O., Ravindran, A., 1985. Branch and bound experiments in convex nonlinear integer programming. Management Science 31 (12), 1533–1546.

Hansen, P., Mladenovi, N., 2001. Variable neighborhood search: Principles and applications. European journal of operational research 130 (3), 449–467.

Holland, J., 1992. Adaptation in natural and artificial systems.

Ibaraki, T., Nakamura, K., 2006. Packing problems with soft rectangles. In: Almeida, F., Blesa, M., Blum, C., Moreno, J. M., Pérez, M., Roli, A., Sampels, M. (Eds.), Proceedings of HM 2006 – 3rd International Workshop on Hybrid Metaheuristics. Vol. 4030 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Germany, pp. 13–27.

Kostina, A.M.; Guillén-Gosálbez, G., Mele, F., Bagajewicz, M., Jiménez, L., 2010. A novel rolling horizon strategy for the strategic 1 planning of supply chains. application to the sugar cane industry of argentina. Preprint submitted to Computers & Chemical Engineering, 54.

Lawler, E., Lenstra, J., Kan, A., Shmoys, D., 1985. The traveling salesman problem: a guided tour of combinatorial optimization. Vol. 3. Wiley New York.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., Teller, E., et al., 1953. Equation of state calculations by fast computing machines. The journal of chemical physics 21 (6), 1087.

Murtagh, B., Saunders, M., 1978. Large-scale linearly constrained optimization. Mathematical Programming 14 (1), 41–72.

Nemhauser, G., Wolsey, L., 1988a. Integer and combinatorial optimization 18.

Nemhauser, G. L., Wolsey, L. A., 1988b. Integer and Combinatorial Optimization. John Wiley & Sons.

Ow, P. S., Morton, T. E., 1988. Filtered beam search in scheduling. International Journal of Production Research 26, 297–307.

Powell, M., 1978. A fast algorithm for nonlinearly constrained optimization calculations. Numerical analysis, 144–157.

Raidl, G. R., 2006. A unified view on hybrid metaheuristics. In: Almeida, F., Blesa, M., Blum, C., Moreno, J. M., Pérez, M., Roli, A., Sampels, M. (Eds.), Proceedings of HM 2006 – 3rd International Workshop on Hybrid Metaheuristics. Vol. 4030 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Germany, pp. 1–12.

Rechenberg, I., Eigen, M., 1973. Evolutionsstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution.

Reeves, C., 1993. Modern heuristic techniques for combinatorial problems. John Wiley & Sons, Inc. New York, NY, USA.

Ropke, S., Pisinger, D., 2006. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation Science 40 (4), 455–472.

Sabio, N., Gadalla, M., Guilln-Goslbez, G., Jimnez, L., 2010. Strategic planning with risk control of hydrogen supply chains for vehicle use under uncertainty in operating costs: A case study of spain. International Journal of Hydrogen Energy 35 (13), 6836 – 6852.

Shaw, P., 1998a. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In: Maher, M., Puget, J.-F. (Eds.), Principle and Practice of Constraint Programming – CP98. Vol. 1520 of Lecture Notes in Computer Science. Springer, pp. 417–431.

Shaw, P., 1998b. Using constraint programming and local search methods to solve vehicle routing problems. Principles and Practice of Constraint ProgrammingCP98, 417–431.

Simchi-Levi, D., Kaminsky, P., Simchi-Levi, E., 2003. Designing and managing the supply chain: concepts, strategies, and case studies. Irwin/McGraw-Hill.

Stubbs, R., Mehrotra, S., 1999. A branch-and-cut method for 0-1 mixed convex programming. Mathematical Programming 86 (3), 515–532.

Stützle, D., 1999. Local Search Algorithms for Combinatorial Problems.

Talbi, E.-G., 2002. A Taxonomy of Hybrid Metaheuristics. Journal of Heuristics 8 (5), 541–564.

Voss, S., Osman, I., Roucairol, C., 1999. Meta-heuristics: Advances and trends in local search paradigms for optimization. Kluwer Academic Publishers Norwell, MA, USA.

Voudouris, C., Tsang, E., 1999. Guided local search and its application to the traveling salesman problem. European Journal of Operational Research 113 (2), 469–499.

Wolsey, L. A., 1998. Integer programming.

Wu, Y., Debs, A., Marsten, R., 2002. A direct nonlinear predictor-corrector primal-dual interior point algorithm for optimal power flows. Power Systems, IEEE Transactions on 9 (2), 876–883.